

Reachability Analysis of Communicating Pushdown Systems

Alex Heußner, Jérôme Leroux, Anca Muscholl, Grégoire Sutre

LaBRI Bordeaux, France

Models for Concurrent Software Systems

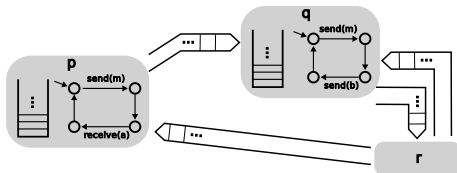
concurrent **pushdown processes** synchronizing over

- **shared variables**
global/local, atomic read/write, observation, . . .
[Quadeer/Rehof '05, Esparza/Bouajjani '05, Atig/Touili '08, . . .]
- **locks** / monitors [Kahlon/Gupta '07 '09]
- “**bags**” of tasks [Sen/Viswanathan '06]
- (fifo) **queues** [La Torre/Madhusudan/Parlato '08]

CPS Communicating Pushdown Systems

- model distributed programs consisting of single local processes with unbounded *recursion* and *asynchronous* communication based on TCP (*point-to-point, fifo, reliable, unbounded*)
- e.g., implemented on top of Berkeley Sockets API

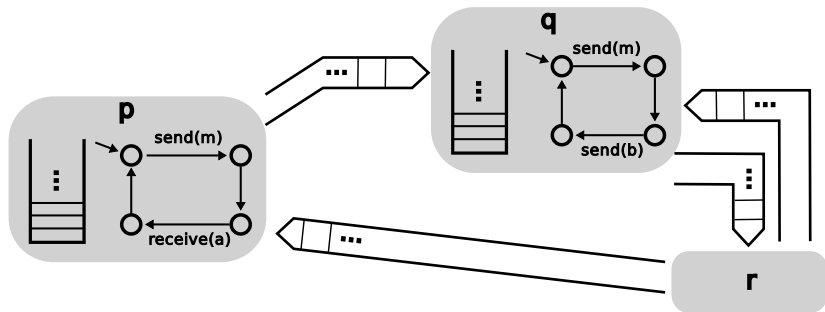
⇒ communication **architecture**:



- ⇒ local **pushdown automata** synchronizing over
- ⇒ **point-to-point, reliable, unbounded fifo** channels

Communicating
Pushdown
System

A Closer Look on CPS



Question

Investigate the border between decidability and undecidability of the **(control-state) reachability** question for CPS

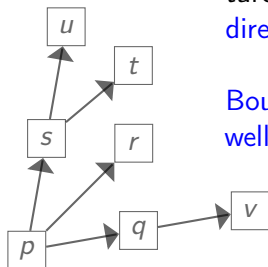
- ⇒ undecidability arises from *both*
 - pushdown stacks (emptiness of intersection of CFL,...)
 - unbounded communication channels
(communicating *finite* state machines are Turing powerful)
- ⇒ need restrictions on
 - pushdown operations, synchronization operations, communication architecture, the interplay of these three
 - the problem (e.g., bounded contexts)

Known Results for CPS

[La Torre, Madhusudan, Parlato: TACAS'08]

restriction channels are **well-queueing**:
can only receive messages when the local stack is empty

results Control state reachability over **well-queueing** architectures is **decidable** iff the architecture is a **root-to-leaf directed tree**. Complexity: **2ExpTime**



Bounded context control-state reachability over **well-queueing** architectures is **decidable** in **2ExpTime**.

Our Approach

ansatz focus the undecidability due to pushdown stacks
⇔ restrict communication by notion of **eagerness**

A run of a CPS is *eager* if each **receive action is directly following its matching send**.

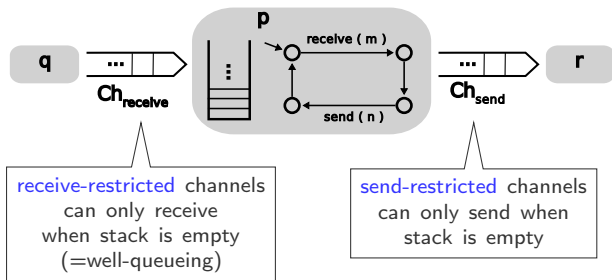
A CPS is *eager* if each configuration can be reached by an eager run.

remark ⇔ **finite state** communicating processes have **decidable** reachability on eager runs

⇔ any CPS is eager when the architecture is an **undirected forest**

Our Approach

ansatz add dual notion to “well-queueing”

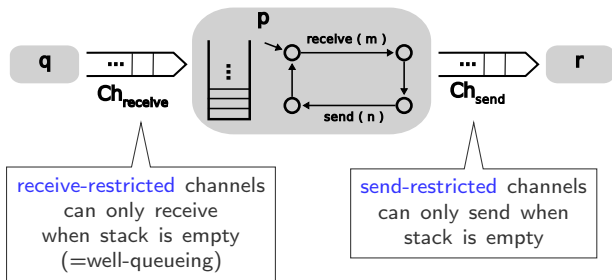


⇒ restrict type of channels allowed for CPS to either send-restricted, receive-restricted, or both send-&receive-restricted

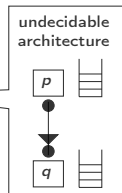


Our Approach

ansatz add dual notion to “well-queueing”



- ⇒ restrict type of channels allowed for CPS to either send-restricted, receive-restricted, or both send-&receive-restricted



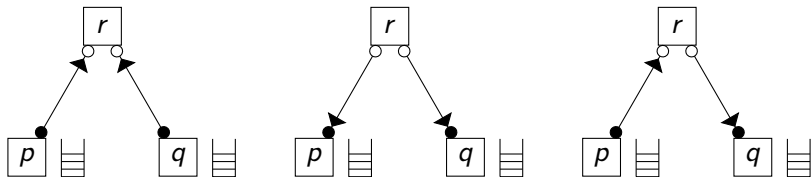
Decidable Architectures

Mutex

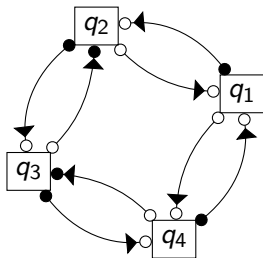
Bounded Phase Reachability

Decidability wrt. Architectures

⇒ undecidable architectures:



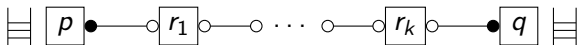
⇒ What about ?



Decidable Architectures

Definition

An architecture is *confluent* if it contains a pair of distinct processes p, q , together with a *simple, unoriented path* p, r_1, \dots, r_k, q between p and q , such that



Result

Control-state reachability is *decidable* on *eager* runs if and only if the architecture is *non confluent*. Complexity: *ExpTime-complete*.

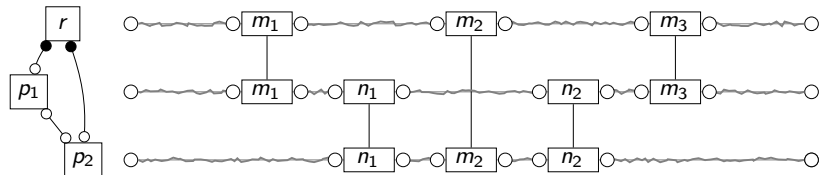
(proof)

Result

Control-state reachability is **decidable** on **eager** runs if and only if the architecture is **non confluent**. Complexity: **ExpTime-complete**.

proof idea:

can always find linearization (scheduling) of a run that can be simulated on **one** pushdown stack



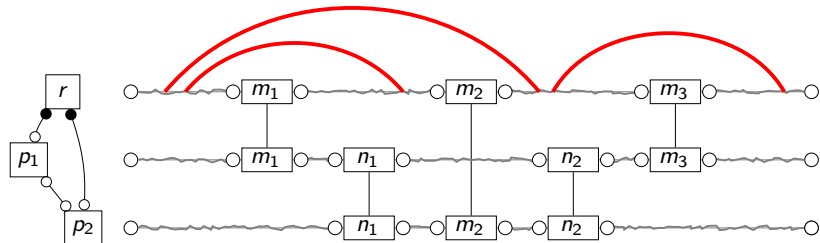
(proof)

Result

Control-state reachability is **decidable** on **eager** runs if and only if the architecture is **non confluent**. Complexity: **ExpTime-complete**.

proof idea:

can always find linearization (scheduling) of a run that can be simulated on **one** pushdown stack



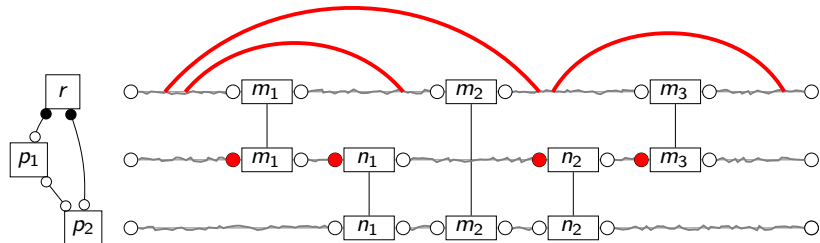
(proof)

Result

Control-state reachability is **decidable** on **eager** runs if and only if the architecture is **non confluent**. Complexity: **ExpTime-complete**.

proof idea:

can always find linearization (scheduling) of a run that can be simulated on **one** pushdown stack



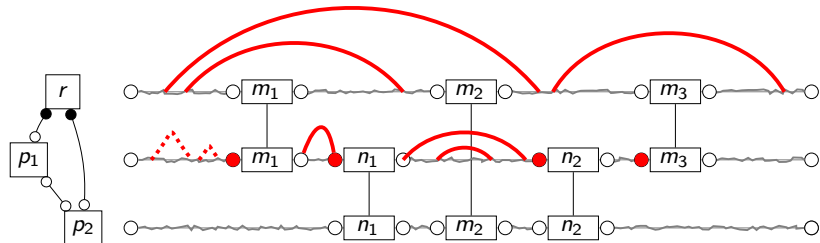
(proof)

Result

Control-state reachability is **decidable** on **eager** runs if and only if the architecture is **non confluent**. Complexity: **ExpTime-complete**.

proof idea:

can always find linearization (scheduling) of a run that can be simulated on **one** pushdown stack



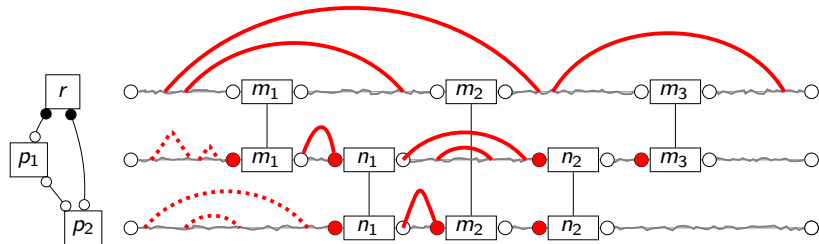
(proof)

Result

Control-state reachability is **decidable** on **eager** runs if and only if the architecture is **non confluent**. Complexity: **ExpTime-complete**.

proof idea:

can always find linearization (scheduling) of a run that can be simulated on **one** pushdown stack



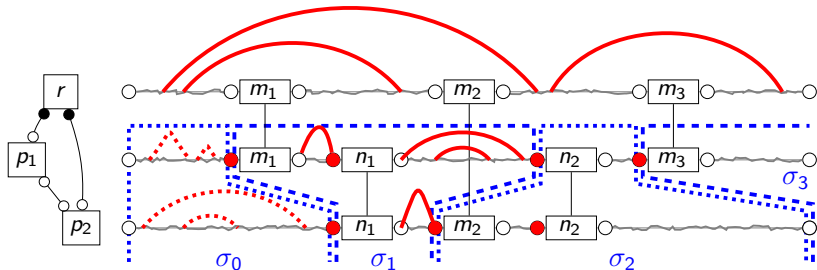
(proof)

Result

Control-state reachability is **decidable** on **eager** runs if and only if the architecture is **non confluent**. Complexity: **ExpTime-complete**.

proof idea:

can always find linearization (scheduling) of a run that can be simulated on **one** pushdown stack



(proof)

Result

Control-state reachability is **decidable** on **eager** runs if and only if the architecture is **non confluent**. Complexity: **ExpTime-complete**.

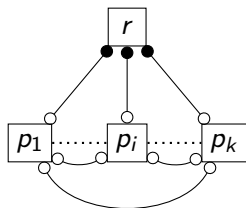
proof sketch:

induction over the architecture (on number of channels and of \bullet)

\Downarrow_1 all channels are send-&receive-restricted ($\circ-\circ$)

\Leftrightarrow check emptiness of **1** pushdown automaton
with **exponentially** many control states

\Downarrow_2 only one process communicates always unrestricted



\Rightarrow reorder each runs as follows

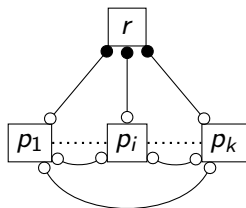


\Rightarrow calculate summaries of control-state reachability for $\{p_1, \dots, p_k\}$ analogous to \Downarrow_1

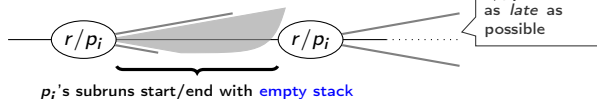
\Rightarrow use these summaries and combine with r 's (pushdown) actions

\Rightarrow simulate by 1 pushdown automaton with exponentially many control-states

\Downarrow_2 only one process communicates always unrestricted

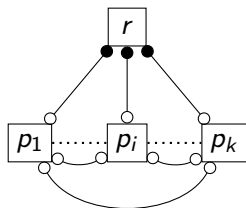


\Rightarrow reorder each runs as follows

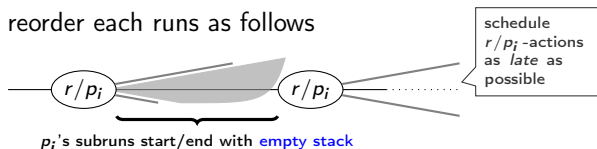


- \Rightarrow calculate summaries of control-state reachability for $\{p_1, \dots, p_k\}$ analogous to \Downarrow_1
- \Rightarrow use these summaries and combine with r 's (pushdown) actions
- \Rightarrow simulate by 1 pushdown automaton with exponentially many control-states

\Downarrow_2 only one process communicates always unrestricted



\Leftrightarrow reorder each runs as follows

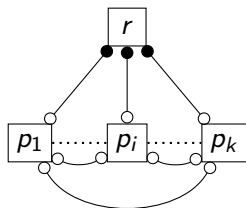


\Leftrightarrow calculate summaries of control-state reachability for $\{p_1, \dots, p_k\}$ analogous to \Downarrow_1

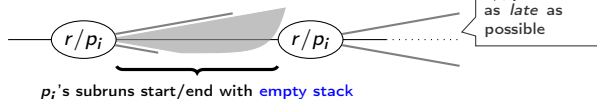
\Leftrightarrow use these summaries and combine with r 's (pushdown) actions

\Leftrightarrow simulate by 1 pushdown automaton with exponentially many control-states

\Downarrow_2 only one process communicates always unrestricted



\Leftrightarrow reorder each runs as follows

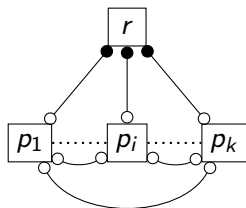


\Leftrightarrow calculate summaries of control-state reachability for $\{p_1, \dots, p_k\}$ analogous to \Downarrow_1

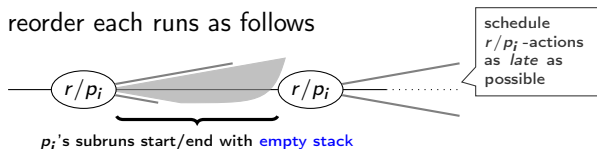
\Leftrightarrow use these summaries and combine with r 's (pushdown) actions

\Leftrightarrow simulate by 1 pushdown automaton with exponentially many control-states

\Downarrow_2 only one process communicates always unrestricted



\Leftrightarrow reorder each runs as follows




\Leftrightarrow calculate summaries of control-state reachability for $\{p_1, \dots, p_k\}$ analogous to \Downarrow_1

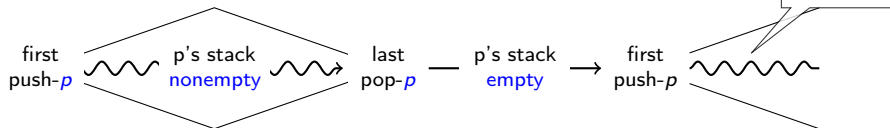
\Leftrightarrow use these summaries and combine with r 's (pushdown) actions

\Leftrightarrow simulate by 1 pushdown automaton with exponentially many control-states

(proof)

step architecture is 

⇔ decompose runs with respect to p :



⇔ channel $p \circ \bullet q$ not used by p in-between \diamond

⇔ q 's stack is empty at push/pop frontiers inside \diamond

key idea: (induction on \diamond)

transform \diamond -subruns into summaries that do not use pushdowns

⇔ this results in  which is conform to $\Downarrow 2$

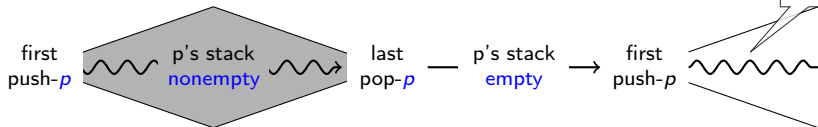
⇔ lower bound: intersection of one pushdown with n finite automata

(proof)

step architecture is



⇨ decompose runs with respect to p :



⇨ channel $p \circ \bullet q$ not used by p in-between \diamond

⇨ q 's stack is empty at push/pop frontiers inside \diamond


key idea: (induction on \diamond)

transform \diamond -subruns into summaries that do not use pushdowns

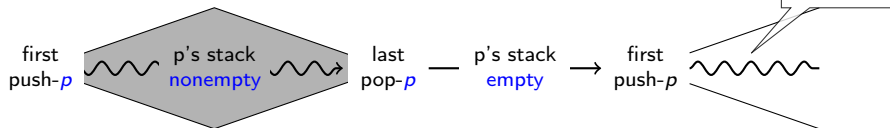
⇨ this results in  which is conform to $\Downarrow 2$

⇨ lower bound: intersection of one pushdown with n finite automata

(proof)

step architecture is 

⇔ decompose runs with respect to p :

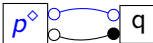


⇔ channel $p \circ \bullet q$ not used by p in-between \diamond

⇔ q 's stack is empty at push/pop frontiers inside \diamond


key idea: (induction on \diamond)

transform \diamond -subruns into summaries that do not use pushdowns

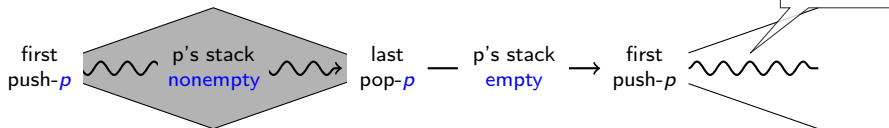
⇔ this results in  which is conform to $\Downarrow 2$

⇔ lower bound: intersection of one pushdown with n finite automata

(proof)

step architecture is 

⇔ decompose runs with respect to p :

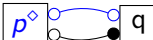


⇔ channel $p \circ \bullet q$ not used by p in-between \diamond

⇔ q 's stack is empty at push/pop frontiers inside \diamond

key idea: (induction on \diamond)

transform \diamond -subruns into summaries that do not use pushdowns

⇔ this results in  which is conform to $\Downarrow 2$

⇔ lower bound: intersection of one pushdown with n finite automata

Decidable Architectures

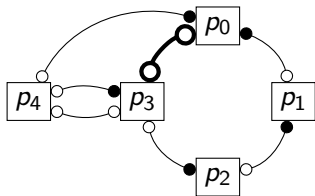
Mutex

Bounded Phase Reachability

Mutex

(A Semantic Restriction on Cycles)

A CPS is *mutex* if for all reachable configurations (local control-states & content of queues) **at most one** channel **per simple cycle** in the architecture is **non-empty**.



- ⇔ polyforest architectures are mutex
- ⇔ for two processes this equals the **half-duplex** property of [Finkel & Cécé CAV'97, ...]

Result

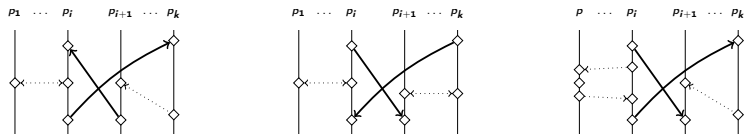
Given a CPS that is *mutex* wrt. a given architecture then each of its runs can be **reordered** into an equivalent **eager** run.

Result

Given a CPS that is **mutex** wrt. a given architecture then each of its runs can be **reordered** into an equivalent **eager** run.

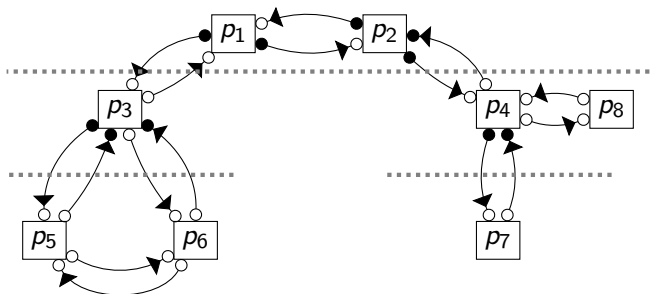
proof idea:

- can always schedule a “minimal” communication event first
- if not, there is a cyclic dependency between all pending events



- but this is prohibited by mutex !

Application to Master-Worker Protocols



“master-worker” distributed computing

- hierarchical network
- can receive tasks only when local stack is empty, can send result when local calculation finished (empty stack)

note: mutex
& non-confluent

Decidable Architectures

Mutex

Bounded Phase Reachability

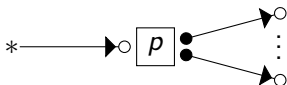
⇒ basic idea:

under-approximation of concurrent processes by bounding the number of switches between processes [Qadeer/Rehof]

Phases

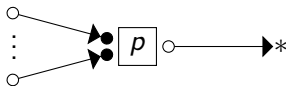
A *phase* of a run of a CPS is defined as:

- a *contiguous* subrun
- where all actions are of a *unique process*, say $p \in \mathcal{P}$
- and p only communicates over channels either like



well-queueing contexts
of [LMP08]; "*m-phase*";

- or like



dual notion; "*n-phase*";

Bounded Phase Reachability

Result

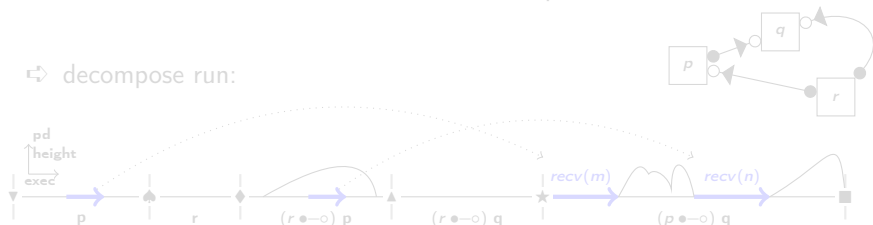
Given a CPS and an integer K , the K phase bounded control-state reachability problem is solvable in time doubly exponential in K and polynomial in the size of the CPS. Lower bound : 2ExpTime.

proof idea: reduce k -bounded question to the $(k - 1)$ -bounded one

assert: only m -phases and empty pushdowns at phase boundaries

? test whether \blacksquare is reachable from \blacktriangledown in ≤ 5 phases on the CPS

\Rightarrow decompose run:



Bounded Phase Reachability

Result

Given a CPS and an integer K , the K phase bounded control-state reachability problem is solvable in time doubly exponential in K and polynomial in the size of the CPS. Lower bound : 2ExpTime.

proof idea: reduce k -bounded question to the $(k - 1)$ -bounded one

assert: only m -phases and empty pushdowns at phase boundaries

? test whether \blacksquare is reachable from \blacktriangledown in ≤ 5 phases on the CPS

⇒ decompose run:



Bounded Phase Reachability

Result

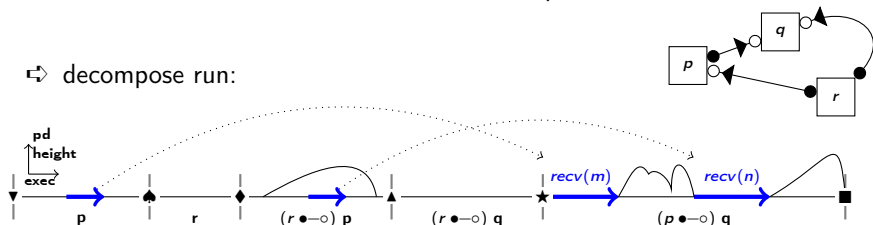
Given a CPS and an integer K , the K phase bounded control-state reachability problem is solvable in time doubly exponential in K and polynomial in the size of the CPS. Lower bound : 2ExpTime.

proof idea: reduce k -bounded question to the $(k - 1)$ -bounded one

assert: only m -phases and empty pushdowns at phase boundaries

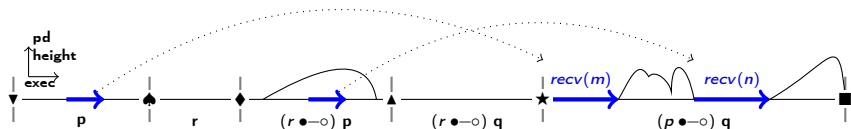
? test whether \blacksquare is reachable from \blacktriangledown in ≤ 5 phases on the CPS

\Leftrightarrow decompose run:



proof (continued)

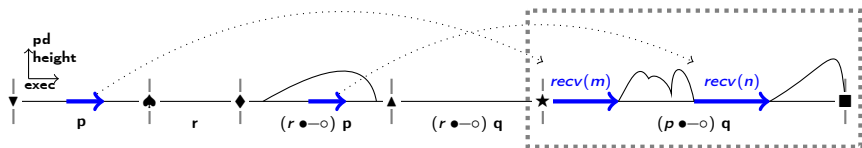
- ⇒ guess intermediate control states ♠, ♦, ▲, ★
- ⇒ guess for each phase its process & input channel



- ⇒ focus last phase
- ⇒ internalize sends of last phase (never received)
- ⇒ receive actions only possible when stack is empty
- ⇒ calculate summaries for subruns with non-empty pushdown
- ⇒ generate finite automaton representation for last phase
- ⇒ test whether $\blacksquare|_p$ is reachable from $\star|_p$
- ⇒ synchronize phases $1..k - 1$ with this automaton
- ⇒ control structure grows in worst case double exponentially

proof (continued)

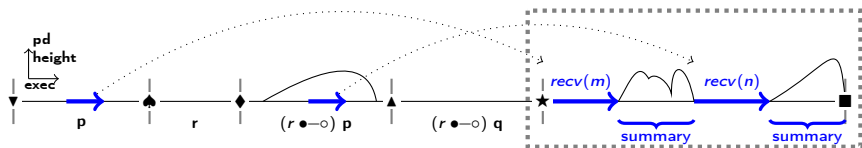
- ⇒ guess intermediate control states ♠, ♦, ▲, ★
- ⇒ guess for each phase its process & input channel



- ⇒ focus last phase
- ⇒ internalize sends of last phase (never received)
- ⇒ receive actions only possible when stack is empty
- ⇒ calculate summaries for subruns with non-empty pushdown
- ⇒ generate finite automaton representation for last phase
- ⇒ test whether $\blacksquare|_p$ is reachable from $\star|_p$
- ⇒ synchronize phases $1..k - 1$ with this automaton
- ⇒ control structure grows in worst case double exponentially

proof (continued)

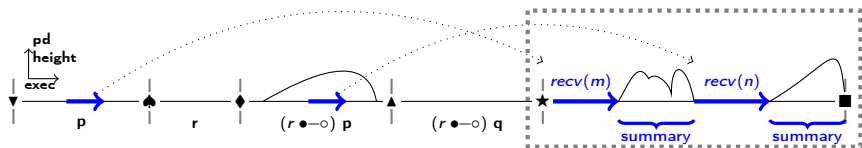
- ⇒ guess intermediate control states ♠, ♦, ▲, ★
- ⇒ guess for each phase its process & input channel



- ⇒ focus last phase
- ⇒ internalize sends of last phase (never received)
- ⇒ receive actions only possible when stack is empty
- ⇒ calculate summaries for subruns with non-empty pushdown
- ⇒ generate finite automaton representation for last phase
- ⇒ test whether $\blacksquare|_p$ is reachable from $\star|_p$
- ⇒ synchronize phases $1..k - 1$ with this automaton
- ⇒ control structure grows in worst case double exponentially

proof (continued)

- ⇒ guess intermediate control states ♠, ♦, ▲, ★
- ⇒ guess for each phase its process & input channel



- ⇒ focus last phase
- ⇒ internalize sends of last phase (never received)
- ⇒ receive actions only possible when stack is empty
- ⇒ calculate **summaries** for subruns with non-empty pushdown
- ⇒ generate **finite automaton** representation for last phase
- ⇒ test whether $\blacksquare|_p$ is reachable from $\star|_p$
- ⇒ synchronize phases $1..k - 1$ with this automaton
- ⇒ control structure grows in worst case **double exponentially**

[LMP'08]

CPS

control-state reachability

well-queueing (=receive-restr.)	$\xrightarrow{\text{dual}}$	receive-, send-, and send&receive-restricted
directed forests	$\xrightarrow{\text{extend}}$	non-confluent architectures with cycles (+mutex)
(all runs implicitly eager)		eager runs
2ExpTime	$\xrightarrow{\text{ameliorate}}$	ExpTime-complete

bounded phase control-state reachability

w-q phase (=m-phase)	$\xrightarrow{\text{dual}}$	m- and n-phases
2ExpTime	$\xrightarrow{\text{direct proof}}$	2ExpTime-complete (ExpTime subclass)

Perspectives

- prototypical implementation
 - need additional heuristics
 - “sub-exponential” sub-models
- verify other properties: liveness, ...
- hierarchical models
 - local models are multiple threads that synchronize by locks, shared variables, etc.
 - share connections between local threads
- model dynamical aspects
 - capture local dynamic multi-threaded environments, e.g., by dynamic pushdown systems

