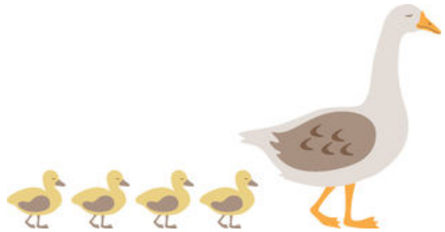
Applications of Unboundedness and Downward Closures to Concurrent Analysis

Matthew Hague
Royal Holloway, University of London

Shonan 2016

In The Beginning...



Theorem

The downward closures of languages defined by higher-order recursion schemes are computable.

- Proof: Igor's talk.
- Question: what can we do with the result?

Tools

Several Tools

Being able to construct a downward closure gives us several tools

- Can decide finiteness of a language.
- Can compute the downward closure of the Parikh Image.
- Can decide the diagonal problem.
- Separability by piecewise testable languages.

The Diagonal Problem

Theorem [Zetzsche, 2015]

If a class of languages C can be synchronised with regular languages, then

- Downward closures are computable iff
- The Diagonal Problem is decidable.

The Diagonal Problem

Theorem [Zetzsche, 2015]

If a class of languages C can be synchronised with regular languages, then

- Downward closures are computable iff
- The Diagonal Problem is decidable.

The Diagonal Problem:

- Given a language $\mathcal{L} \in C$ and a set of characters $\{a_1, \ldots, a_n\}$
- For all k is there a word in $\mathcal{L}$ containing,
  - more than k a_1s, more than k a_2s, and so on

Example

The language $\mathcal{L} = \{a\$b,\ aa\$bb,\ aaa\$bbb, \ldots\}$

Example

The language $\mathcal{L} = \{a\$b,\ aa\$bb,\ aaa\$bbb, \ldots\}$

Satisfies the diagonal problem for $\{a, b\}$

- For $k = 3$, take $aaa\$bbb$
- For every k, take $a^k\$b^k$.
- Note: no word in $\mathcal{L}$ contains an infinite number of as and bs.

Example

The language $\mathcal{L} = \{a\$b,\ aa\$bb,\ aaa\$bbb, \ldots\}$

Satisfies the diagonal problem for $\{a, b\}$

- For $k = 3$, take $aaa\$bbb$
- For every k, take $a^k\$b^k$.
- Note: no word in $\mathcal{L}$ contains an infinite number of as and bs.

Does not satisfy the diagonal problem for $\{\$\}$

- All words in $\mathcal{L}$ only contain one \$
- Fails for $k = 2$

Example

The language $\mathcal{L} = \{a\$b,\ aa\$bb,\ aaa\$bbb, \ldots\}$

Satisfies the diagonal problem for $\{a, b\}$

- For $k = 3$, take $aaa\$bbb$
- For every k, take $a^k\$b^k$.
- Note: no word in $\mathcal{L}$ contains an infinite number of as and bs.

Does not satisfy the diagonal problem for $\{\$\}$

- All words in $\mathcal{L}$ only contain one \$
- Fails for $k = 2$

Gives us a kind of unboundedness check.

Separability by Piecewise Testable Languages

A piecewise testable language is a finite boolean combination of regular expressions of the form

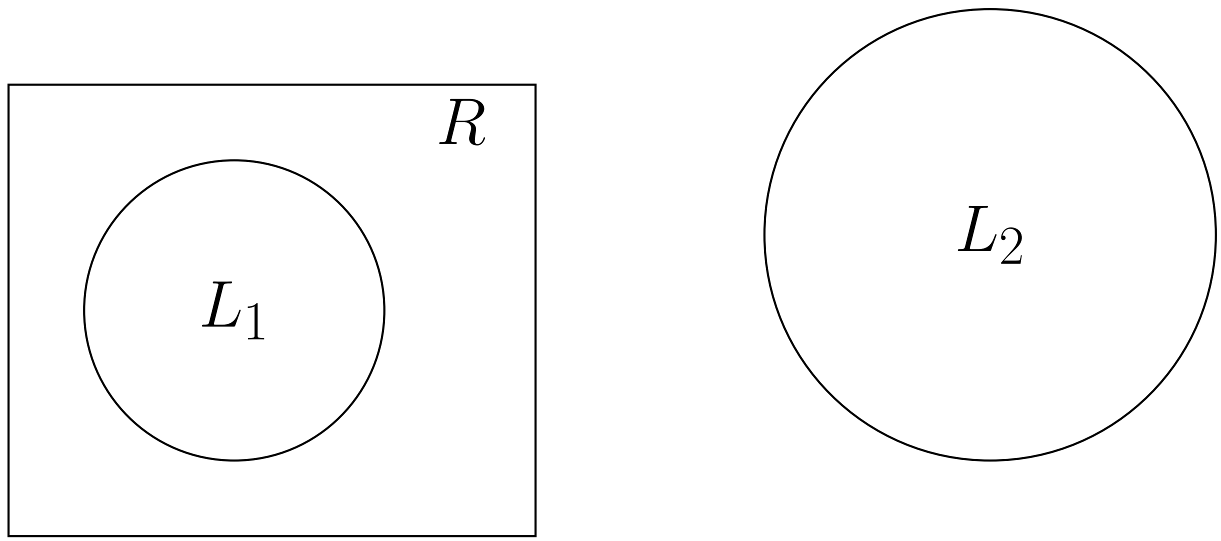
$$\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$$

That is, only the order of $a_1, \ldots, a_n$ is important.

Separability by Piecewise Testable Languages

Separability asks

- Given languages L_1 and L_2,
- Does there exist piecewise testable R such that
  - $L_1 \subseteq R$, and
  - $R \cap L_2 = \emptyset$.



Decidability of Piecewise Testability

Theorem [Czerwinski et al]

If the downwards closure of L_1 and L_2 are computable

- separability by piecewise testable languages is decidable, and
- such an R can be constructed.

This gives us

- A "well-behaved" over-approximation of L_1,
- that is refined enough to avoid hitting L_2.
- (we could then get a similar approximation of L_2.)

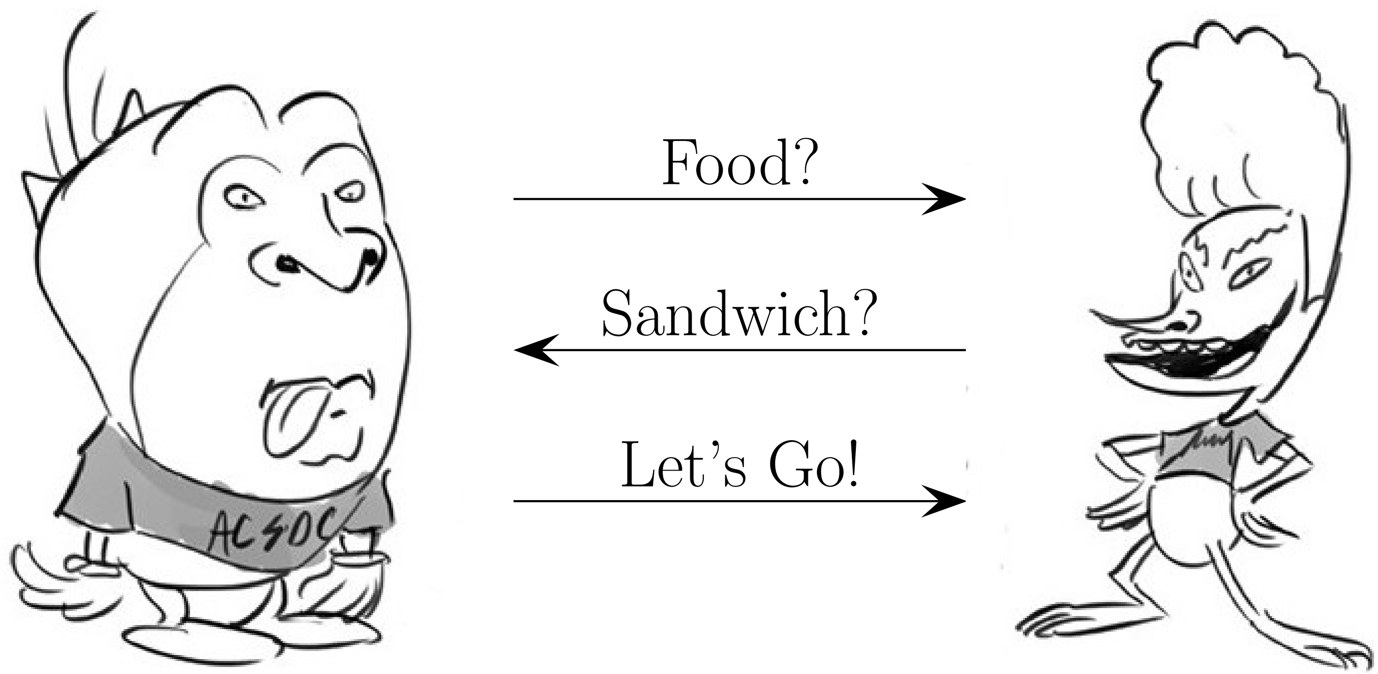
Applications

Example Applications

Can be applied to analysis of concurrent systems.

- Simple / vague idea.
  - To start thinking...
- Parameterised Asynchronous Shared-Memory Systems
  - Concrete result [La Torre *et al*]
- Asynchronous Atomic Methods
  - (in the style of Viswanation *et al*)

Lunch Protocols

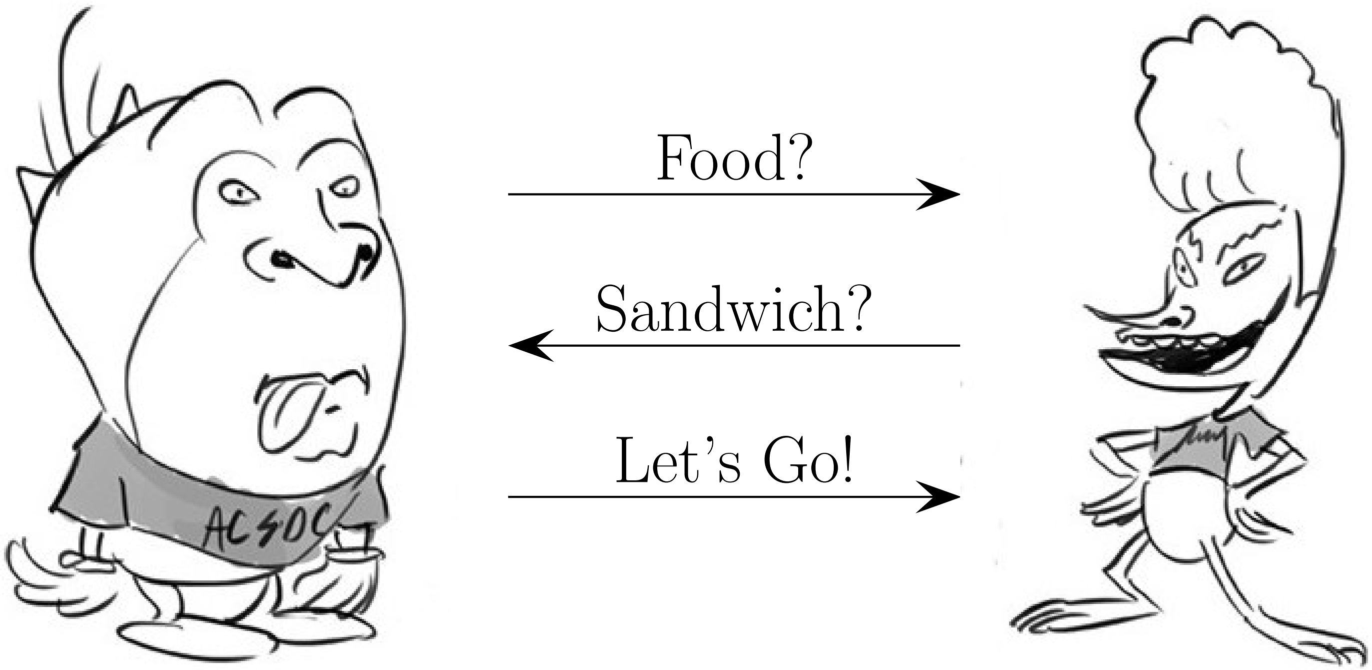


- Let A_V be a regular automaton specifying valid protocol runs.
- They go to lunch if there is a run in

$\mathcal{L}\Big($  $\Big) \cap \mathcal{L}\Big($  $\Big) \cap \mathcal{L}(A_V)$

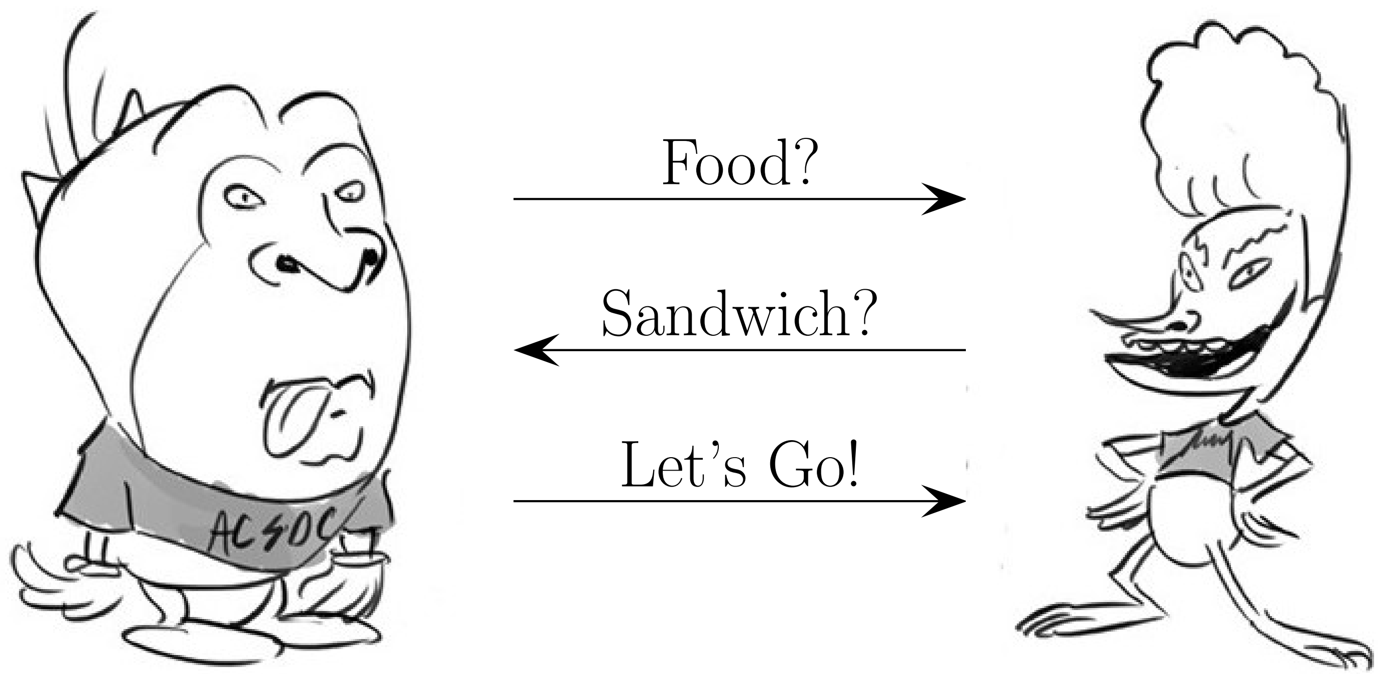
(Both execute a valid run of the protocol.)

Lunch Protocols



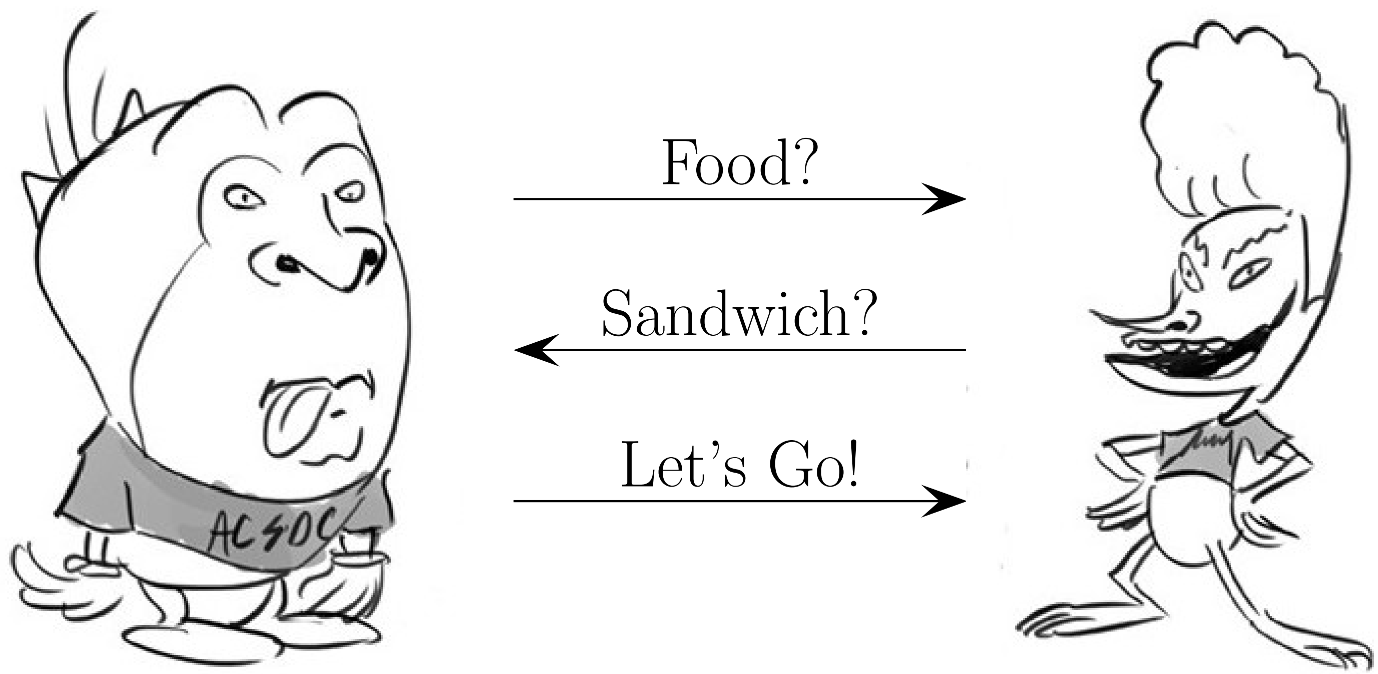
- If  and  are Turing machines
  - Can't even tell if there's a run in $\mathcal{L}\big($$\big)$.
- If  and  are pushdown automata
  - Can test $\mathcal{L}\big($$\big)$ but not $\mathcal{L}\big($$\big) \cap \mathcal{L}\big($$\big)$.

Lunch Protocols



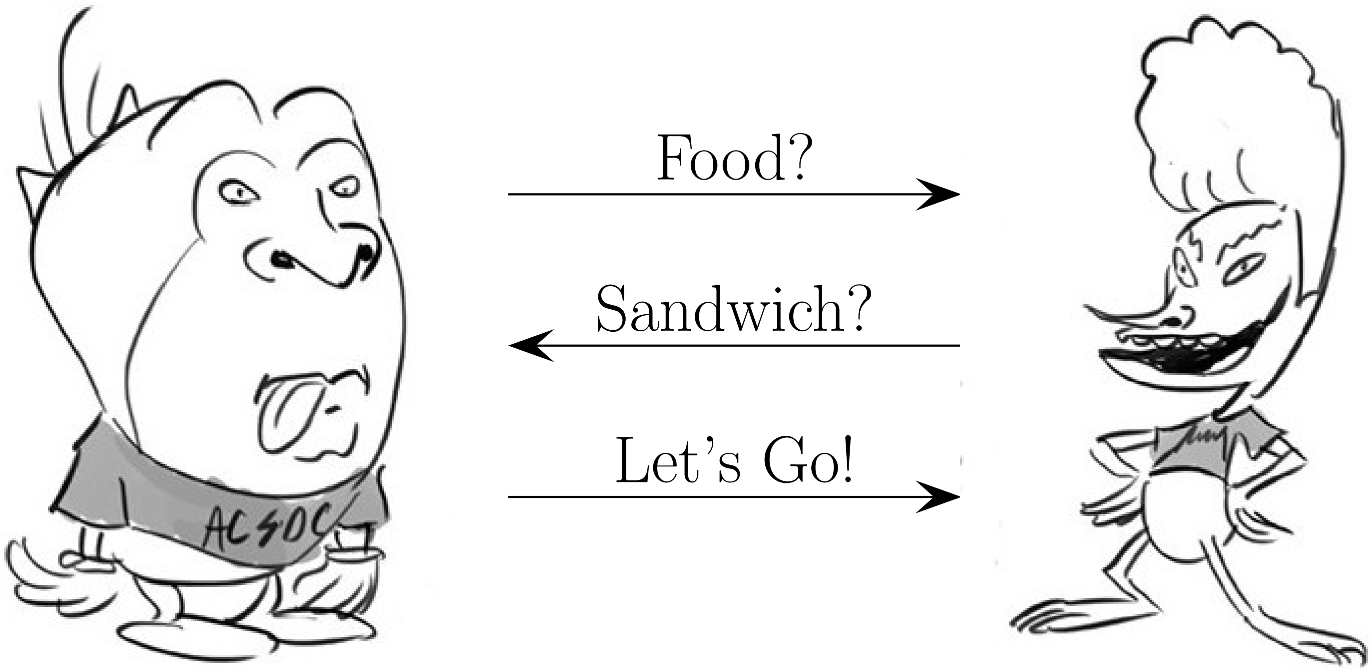
- If  and  are regular we can test $\mathcal{L}\Big($$\Big) \cap \mathcal{L}\Big($$\Big) \cap \mathcal{L}(A_V)$

Lunch Protocols



- If  and  are regular we can test $\mathcal{L}\Big($$\Big) \cap \mathcal{L}\Big($$\Big) \cap \mathcal{L}(A_V)$
- The downward closure of  and  are regular!

Lunch Protocols



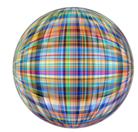
- If  and  are regular we can test $\mathcal{L}\big($$\big) \cap \mathcal{L}\big($$\big) \cap \mathcal{L}(A_V)$
- The downward closure of  and  are regular!
- The intersection with A_V may eliminate "faulty" runs caused by deleting characters.

Parameterized Asynchronous Shared-Memory

Suppose a model where we have

Parameterized Asynchronous Shared-Memory

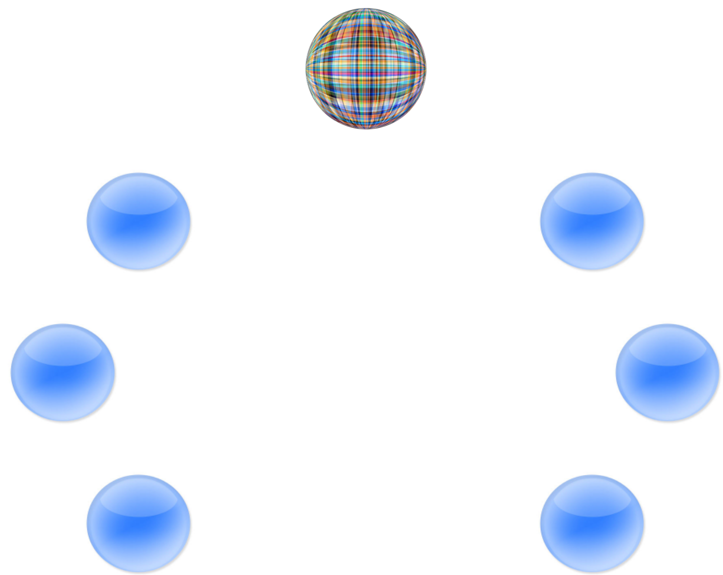
Suppose a model where we have



- A master process.

Parameterized Asynchronous Shared-Memory

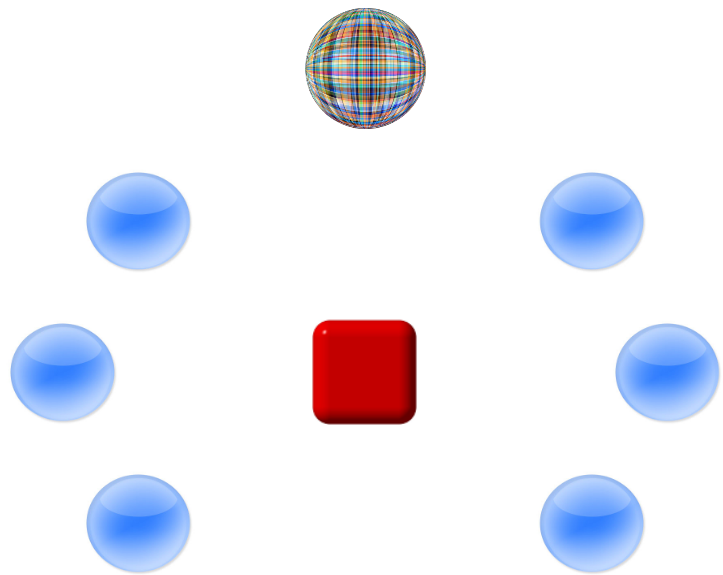
Suppose a model where we have



- A master process.
- Any number of identical slave processes.

Parameterized Asynchronous Shared-Memory

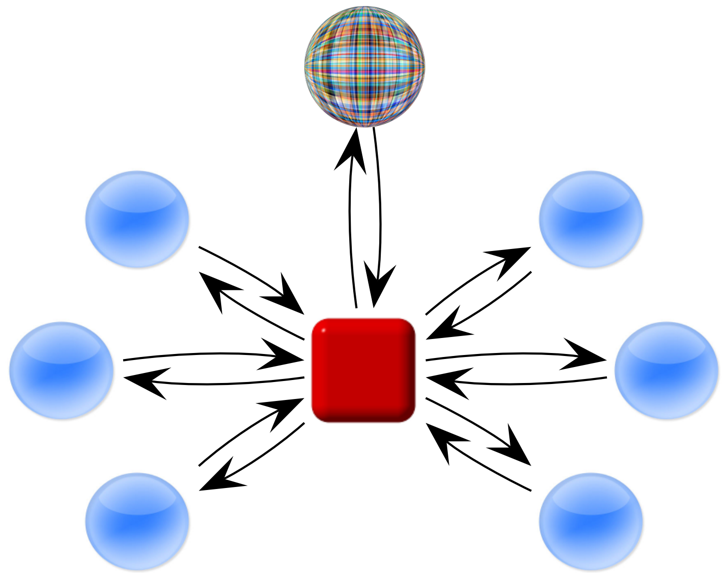
Suppose a model where we have



- A master process.
- Any number of identical slave processes.
- A global shared-memory.

Parameterized Asynchronous Shared-Memory

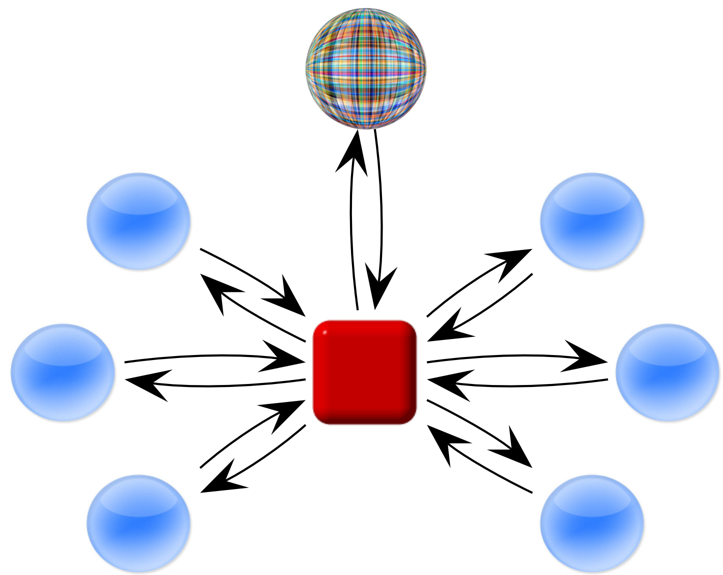
Suppose a model where we have



- A master process.
- Any number of identical slave processes.
- A global shared-memory.
- Processes can read and write to the memory.

Parameterized Asynchronous Shared-Memory

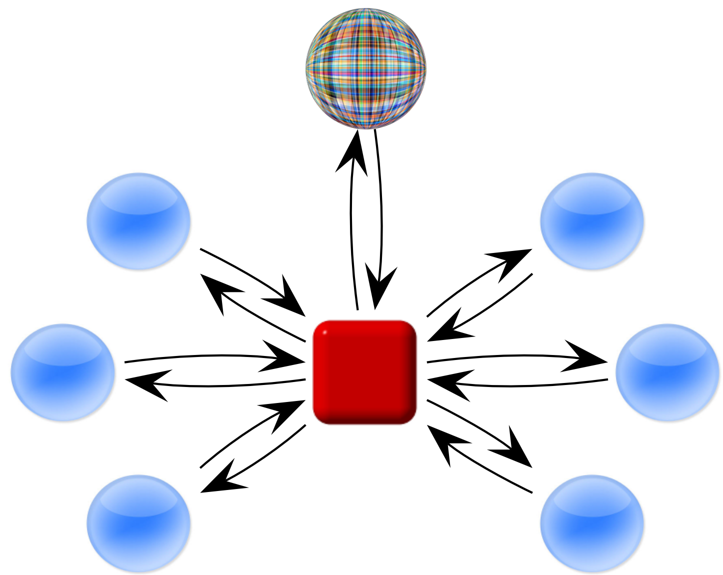
Suppose a model where we have



- A master process.
- Any number of identical slave processes.
- A global shared-memory.
- Processes can read and write to the memory.
- But cannot read and write atomically.

Parameterized Asynchronous Shared-Memory

Suppose a model where we have



Theorem [La Torre et al, 2015]

Reachability is decidable whenever

- Can synchronise all processes with regular automata.
- Reachability of the slave is decidable.
- The downward closure of the master can be computed.

Intuition

We can see some intuition behind the result as follows:

- Each process's view of the global store can be its downward closure.
- Observation:
  - if a slave writes a symbol
  - an arbitrary number of slaves can also write it
  - at any point in the future
- Only "precious" resource are master writes.
  - but we only need the downward closure of the master
- Boils down to a number of reachability checks on the client combined with the master.

Asynchronous Atomic Methods

Suppose we have a system with

- A set of processes $P_1, \ldots, P_n$
- A global control state from a finite set
- Processes may spawn further processes
  - E.g. a run of P_1 may spawn P_3 and two copies of P_7.
- When a process terminates, another spawned process is scheduled
  - Only communication is by control state after termination.

History:

- Pushdown systems [Sen & Viswanathan]
- Generic systems [Chadha & Viswanation]

Runs with Asynchronous Atomic Methods

A configuration is

$$(q, P, M)$$

where

- q is the global control state
- P is the state of the currently exectuting process
- M is a multiset of waiting processes

Transitions of The System

A transition

$$(q, P, M) \longrightarrow (q', P', M')$$

- updates the control state and the running process
- P may add new elements to M

Transitions of The System

A transition

$$(q, P, M) \longrightarrow (q', P', M')$$

- updates the control state and the running process
- P may add new elements to M

If the process has terminated

$$(q, \perp, M) \longrightarrow (q, P, M - P)$$

- A $P \in M$ is chosen non-deterministically to run next

Deciding Reachability of a Control State

Can model as a vector addition system (almost)

$$(q, P, c_1, \ldots, c_n)$$

- q is the control state
- P as before (hence not a real VASS)
- c_i counts the number of P_i waiting to run

Deciding Reachability of a Control State

Can model as

$$(q, P, c_1, \ldots, c_n)$$

Key Observation:

- If we only care if some q can be reached, then
- non-deterministically forgetting scheduled processes does not affect analysis.
- P can be approximated by its downward closure
  - Since this is regular, the entire system is a VASS
  - We can decide coverability of a VASS.

Bounded Synchronisation and Thread Spawning

Atig *et al* give the following model

- Allow context switches.
  - Current P suspends to let another P' run
  - P is rescheduled at most k times
  - for some a priori fixed k
  - P pushdown systems

Reachability is decidable

- Atig *et al*'s proof relies on downward closures
- Can we generalise P to any process for which we have the downward closure?

Do We Need to Compute the Downward Closure?

Chadha and Viswanathan give a generic algorithm for certain "well-behaved" systems of the form

$$(q, P, c_1, \ldots, c_n)$$

- This includes asynchronous atomic method calls
- Their algorithm is by repeated approximation
- Broadly speaking, only membership of the downward closure is required
  - I'm glossing over a lot of details here...

Hence we can ask if our algorithms really need to compute the downward closure, or merely test it.

When is the Downward Closure Not Enough

Take our asynchronous atomic methods system

$$(q, P, M)$$

Fairness question [Majumdar]: are all processes eventually run?

- Taking the downward closure "forgets processes"
- It's easy to be fair if we can forget inconvenient processes...
- Replacing P with its downward closure is too inaccurate.

Conclusion

Conclusions

Downward closure results have given us new tools for reasoning about HORS

- Downward closure, separability, Parikh image (approx)

How can we apply them?

We covered two applications

- Parameterised asynchronous shared-memory
- Asynchronous method calls