Analysis of Depth-Bounded Processes

Damien Zufferey[1] Thomas Wies[2] Thomas A. Henzinger[1]

[1]IST Austria

[2]New York University

MSR Redmond, December 14, 2011

Mobile Processes

Why mobile processes ?

- Mobile devices are ubiquitous (e.g. mobile phone).
- Mobility becomes common in PL abstraction (e.g. actor model [Hewitt et al., 1973]).

Interesting features of mobile processes:

- Process creation
- Mobility (communication channels as first class citizens)
- Assume no shared memory

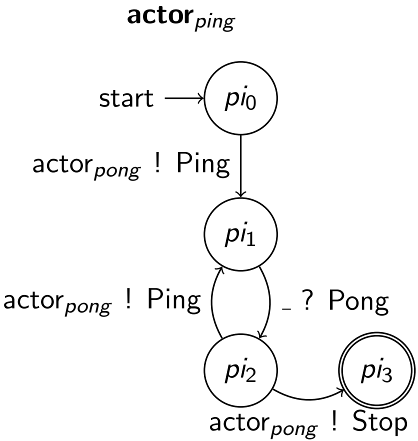
How common is mobility ? What are the use cases ?

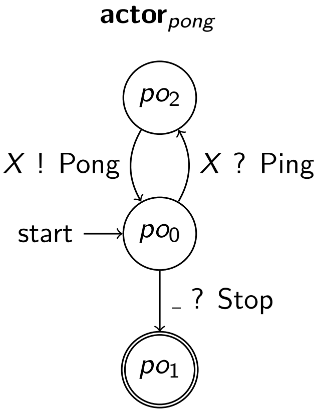
Example (1): scala/docs/examples/actors/pingpong.scala

```
class Ping(count: Int, pong: Actor) extends Actor {
  def act() {
    var pingsLeft = count - 1
    pong ! Ping
    loop {
      react {
        case Pong =>
          if (pingsLeft % 1000 == 0)
            println("Ping: pong")
          if (pingsLeft > 0) {
            pong ! Ping
            pingsLeft -= 1
          } else {
            println("Ping: stop")
            pong ! Stop
            exit()
          }
      }
    }
  }
}
```

```
class Pong extends Actor {
  def act() {
    var pongCount = 0
    loop {
      react {
        case Ping =>
          if (pongCount % 1000 == 0)
            println("Pong: ping "+pongCount)
          sender ! Pong
          pongCount += 1
        case Stop =>
          println("Pong: stop")
          exit()
      }
    }
  }
}
```

Example (2): scala/docs/examples/actors/pingpong.scala





- '?' means receive a message.
- '!' means send a message.

Communication primitives and mobility

In the case of the scala actor library:
(asynchronous communication with mailboxes)

- Send the address of an actor to another (within a message).
- Messages implicitly carry return address to `reply`.
- `forwarding` messages.
- Creating new process (also a new address and mailbox).
- Emulating synch. communication with shared mailboxes.

π-calculus or not ?

Shall I introduce the π-calculus, or can I continue with pictures ?

$A\pi$-calculus: Concepts

The π-calculus [Milner et al., 1992a, Milner et al., 1992b] is a process calculus able to describe concurrent computations whose configuration may change during the computation.
The *asynchronous* π-calculus [Honda and Tokoro, 1991] is a restriction of the π-calculus.

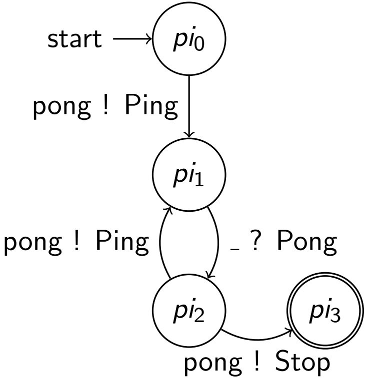
It is build around the notions of

- Names : channels as first class values.
- Threads : concurrent execution of parallel threads: $P \mid Q$.
- i/o prefixes : sending/receiving messages.

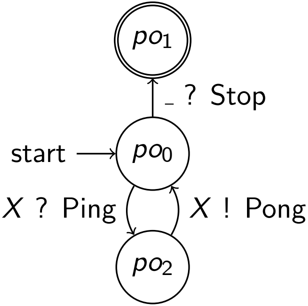
$A\pi$-calculus: Syntax

$$
\begin{array}{lcll}
P & ::= & x(y).P & \text{(input prefix)} \\
 & \mid & \overline{x}\langle y \rangle & \text{(output)} \\
 & \mid & \sum_i a_i(b_i).P_i & \text{(external choice)} \\
 & \mid & P \mid P & \text{(parallel composition)} \\
 & \mid & !P & \text{(replication)} \\
 & \mid & (\nu x)P & \text{(name creation)} \\
 & \mid & 0 & \text{(unit process)}
\end{array}
$$

$$
\begin{aligned}
pi_0 &= \overline{\text{pong}_{Ping}}\langle \text{ping}_{Pong} \rangle | pi_1 \\
pi_1 &= \text{ping}_{Pong}().pi_2 \\
pi_2 &= pi_{2a} \oplus pi_{2b} \\
pi_{2a} &= \overline{\text{pong}_{Ping}}\langle \text{ping}_{Pong} \rangle | pi_1 \\
pi_{2b} &= \overline{\text{pong}_{Stop}}\langle\rangle | pi_3 \\
pi_3 &= 0
\end{aligned}
$$



$$
\begin{array}{rcl}
po_0 & = & \mathrm{pong}_{Stop}().po_1 \\
 & + & \mathrm{pong}_{Ping}(X).po_2(X) \\
po_1 & = & 0 \\
po_2(X) & = & \overline{X}\langle\rangle | po_0
\end{array}
$$



$A\pi$-calculus: Semantics

Evaluating a formula in $A\pi$-calculus reduces to applying the rule:

$$\overline{a}\langle b\rangle \mid \sum_{i\in I} a_i(b_i).Q_i \quad \rightarrow \quad Q_x[b/b_x] \quad \text{where } a_x = a$$

What happens:

- channel a carries b;
- b is sent through a and replace b_x in the continuation Q_x.

$A\pi$-calculus: Semantics

Evaluating a formula in $A\pi$-calculus reduces to applying the rule:

$$\overline{a}\langle b\rangle \mid \sum_{i\in I} a_i(b_i).Q_i \quad \rightarrow \quad Q_x[b/b_x] \qquad \text{where } a_x = a$$

What happens:

- channel *a carries b*;
- b is sent through a and replace b_x in the continuation Q_x.

$A\pi$-calculus: Semantics

Evaluating a formula in $A\pi$-calculus reduces to applying the rule:

$$\overline{a}\langle b\rangle \mid \sum_{i\in I} a_i(b_i).Q_i \quad\rightarrow\quad Q_x[b/b_x] \qquad \text{where } a_x = a$$

What happens:

- channel a carries b;
- b is sent through a and replace b_x in the continuation Q_x.

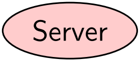
What are Depth-Bounded Processes (DBP) ?

As buzzwords: concurrent/distributed message-passing programs with process creation and mobility. (Warning restrictions may apply.)

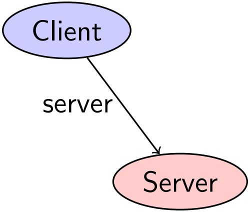
For the programmers: some class of programs using the actor model (Erlang, Scala, Akka, ActorFoundry, ...)

For the theoreticians: a fragment of the π-calculus.

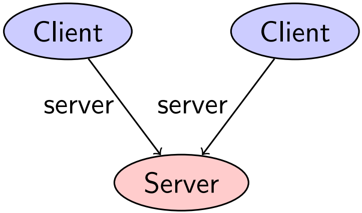
Example: client-server communication pattern

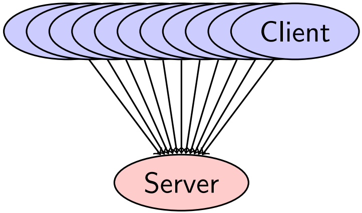


Example: client-server communication pattern

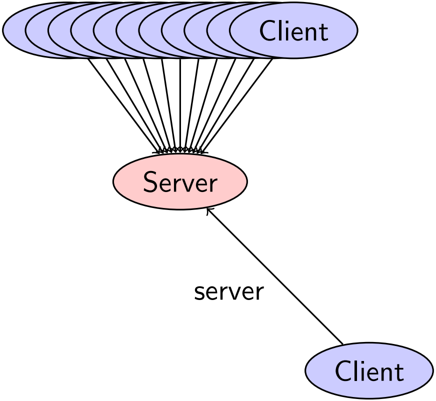


Example: client-server communication pattern

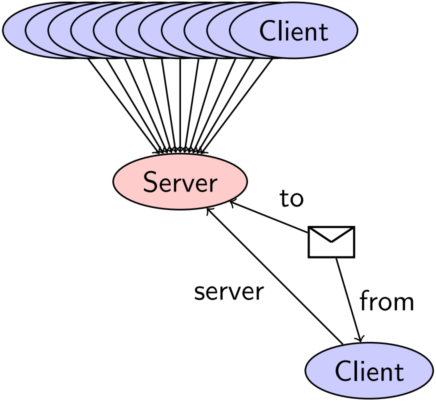



Client
Server

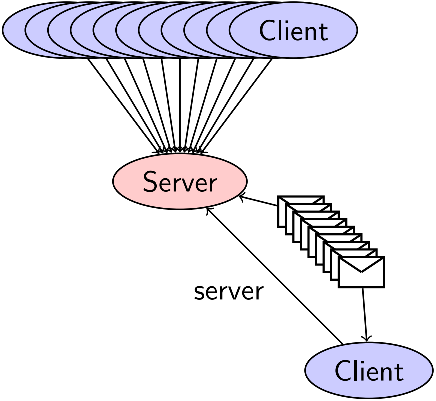
Example: client-server communication pattern



Example: client-server communication pattern

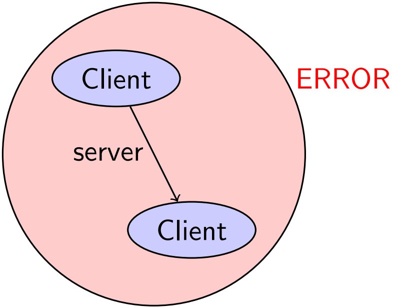


Example: client-server communication pattern



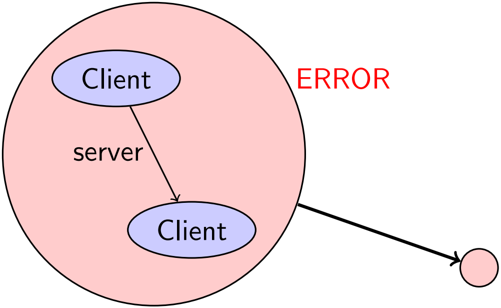
What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).



What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).



What kind of properties are we looking at ?

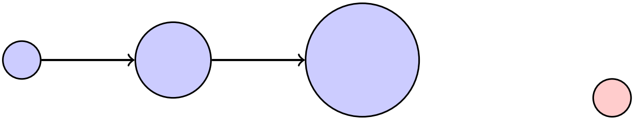
Safety properties, more precisely the control-state reachability problem (aka covering problem).

What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).

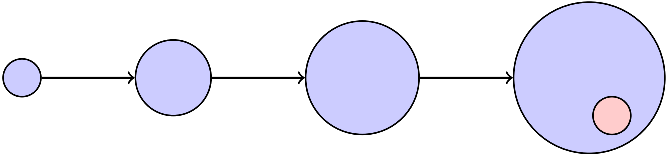
What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).



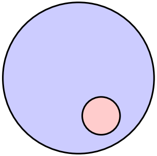
What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).

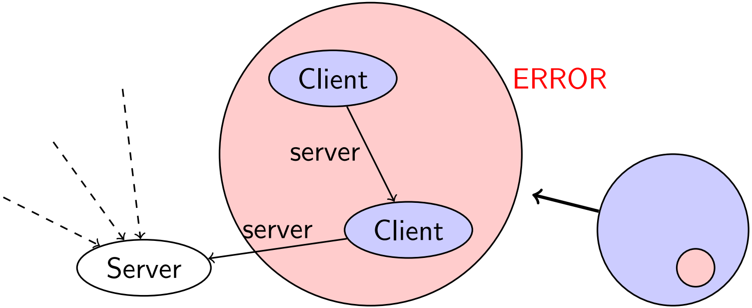


What kind of properties are we looking at ?

Safety properties, more precisely the control-state reachability problem (aka covering problem).



Safety properties, more precisely the control-state reachability problem (aka covering problem).



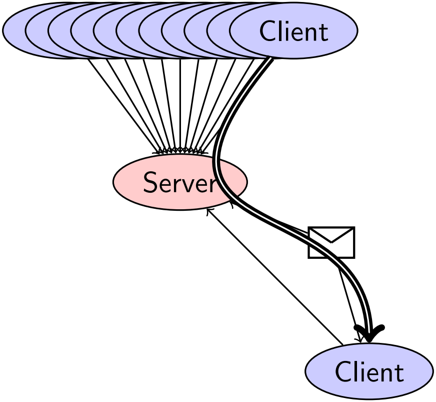
A well-structured transition system (WSTS) is a transition system $\langle S, \rightarrow, \leq \rangle$ such that:

- $\leq$ is a well-quasi-ordering (wqo),
  i.e. well-founded + no infinite antichain.
- compatibility of $\leq$ w.r.t. $\rightarrow$ (simulation)

$$\forall \quad \begin{tikzcd} t \arrow[r, "*"] & t' \\ s \arrow[u, phantom, "\rotatebox{90}{$\leq$}"] \arrow[r] & s' \arrow[u, phantom, "\rotatebox{90}{$\leq$}"] \end{tikzcd} \quad \exists$$

For more detail see:
[Finkel and Schnoebelen, 2001, Abdulla et al., 1996]

System with a bound on the longest acyclic path.
(Concretely: it is not possible to encode an infinite memory.)

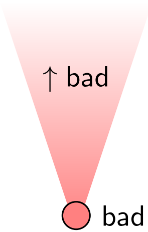


Why DBP are WSTS ?

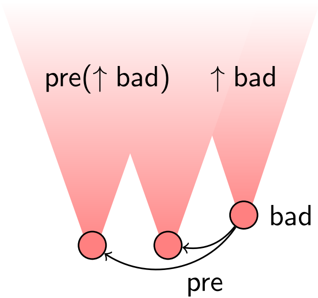
WQO subgraph isomorphism for graphs of specific shapes (proof by a generalisation of Kruskal tree theorem).

Compatibility The π-calculus has no fairness constraints. Additional processes (greater in the ordering) can simply be ignored.

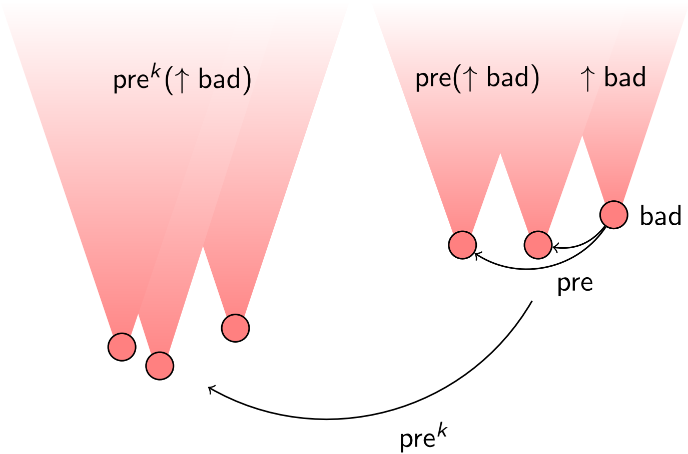
Backward algorithm for covering



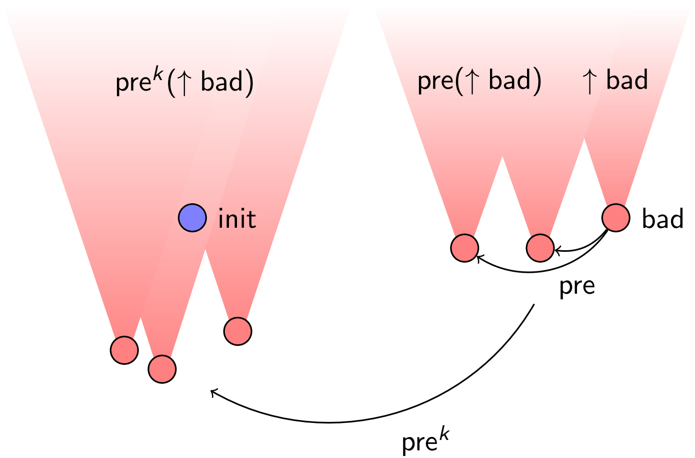
Backward algorithm for covering



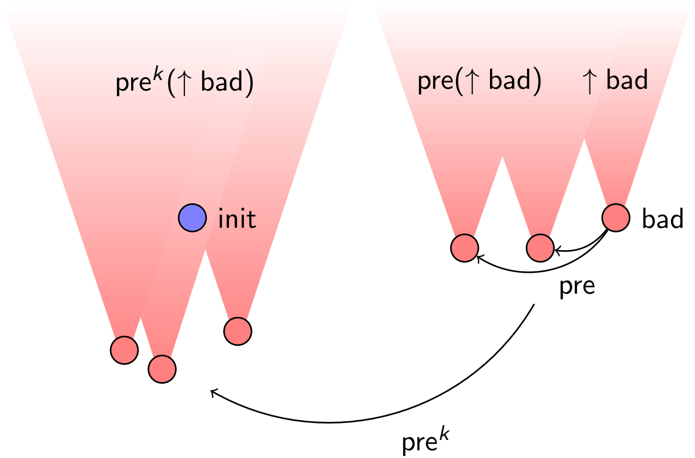
Backward algorithm for covering



Backward algorithm for covering

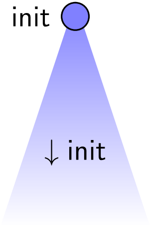


Backward algorithm for covering

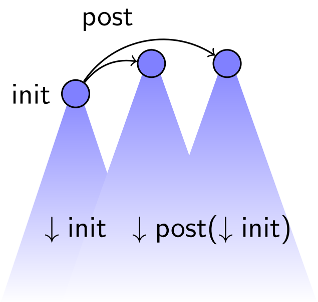


Computing the pre for DBP is not practical (aliasing problem)!
Also the theoretical complexity is terrible.

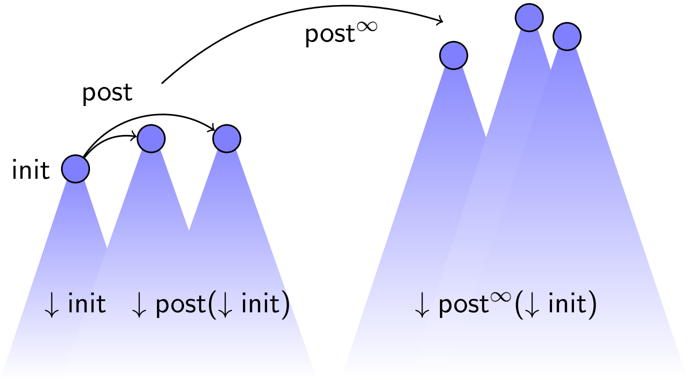
Forward algorithm for covering using acceleration



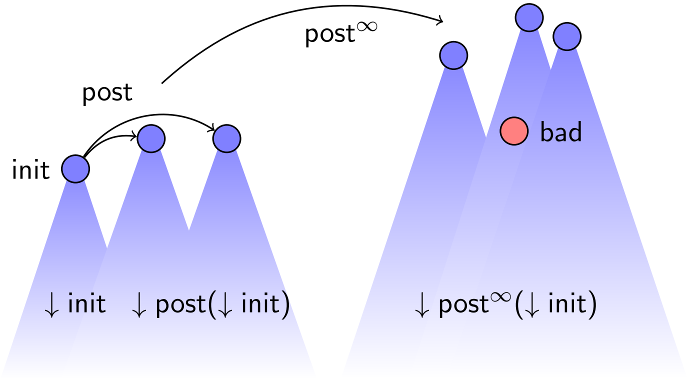
Forward algorithm for covering using acceleration



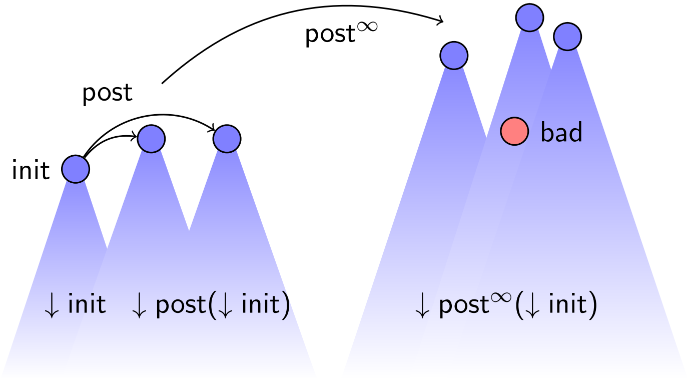
Forward algorithm for covering using acceleration



Forward algorithm for covering using acceleration

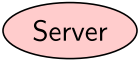


Forward algorithm for covering using acceleration



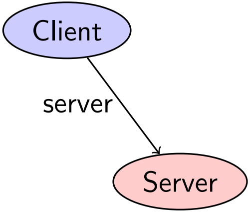
Successfully applied to Petri nets (+extensions) and lossy channels systems. Even though computing the covering set is not decidable [Dufourd et al., 1998].

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]



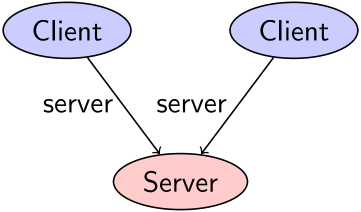
Adequate domain of limits

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]



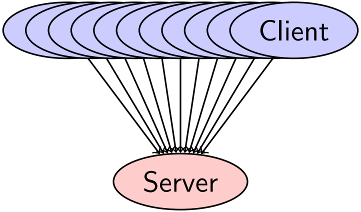
Adequate domain of limits

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]



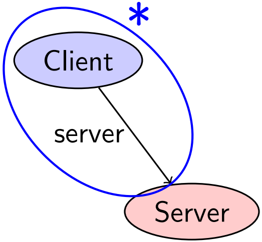
Adequate domain of limits

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]



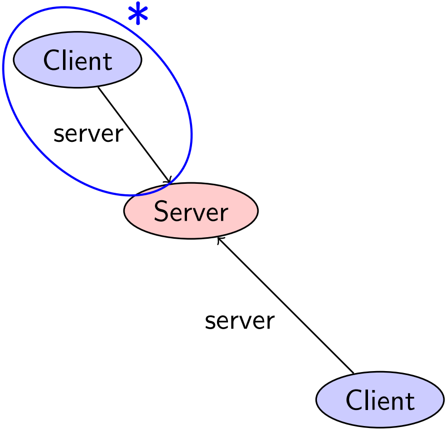
Adequate domain of limits

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]



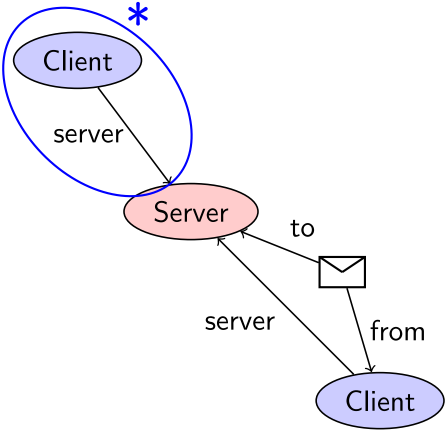
Adequate domain of limits

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]



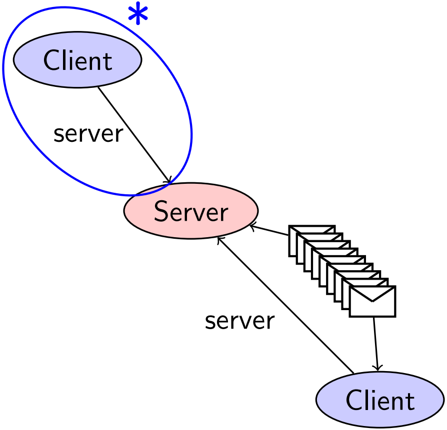
Adequate domain of limits

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]



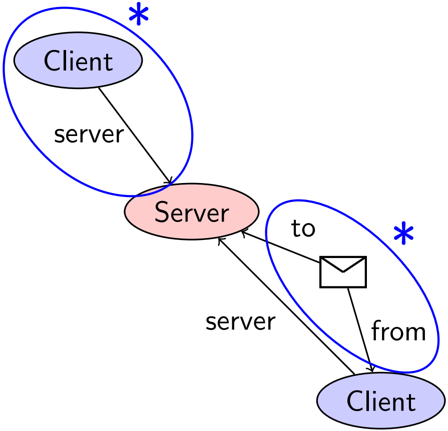
Adequate domain of limits

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]



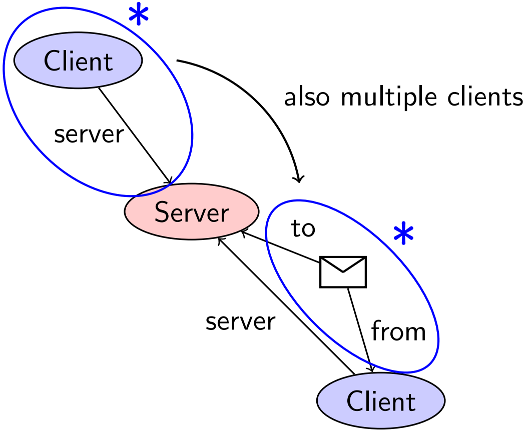
Adequate domain of limits

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]



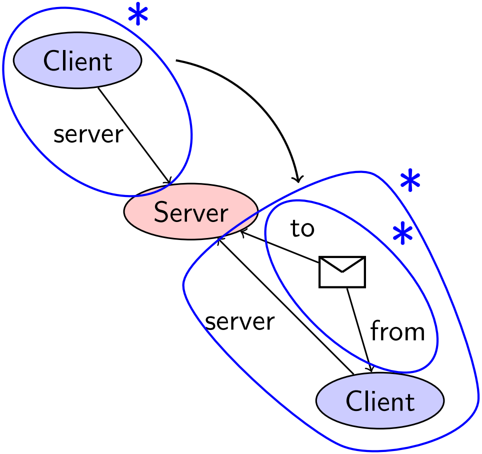
Adequate domain of limits

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]



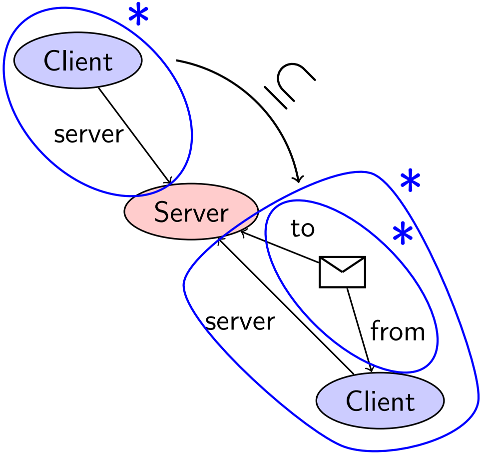
Adequate domain of limits

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]

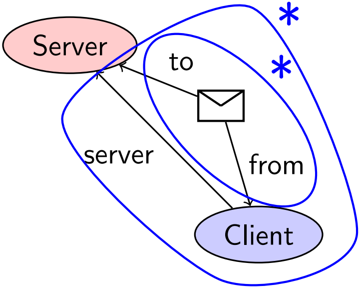


Adequate domain of limits

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]



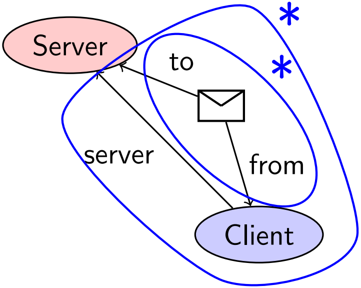
ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]



Adequate domain of limits

ADL: [Geeraerts et al., 2006]
Further developed in [Finkel and Goubault-Larrecq, 2009]
Applied to DBP in [Wies et al., 2010]

$$(\nu x)(Server(x) \mid !(\nu y)(Client(y, x) | !Messages(x, y)))$$



When does acceleration work ? (flat systems)

Usually forward algorithms are based on acceleration. By acceleration we mean computing the result of executing a loop infinitely many time.

We can see this as computing the result of execution traces of length $< \omega^2$. Concretely, it means that the algorithm can saturate the covering set by executing only simple loops (see [Bardin et al., 2005]). This condition is known as flattability.

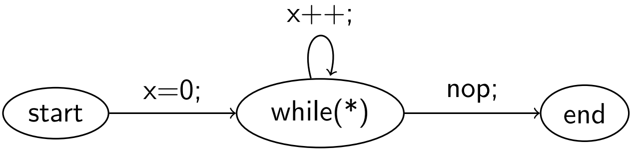
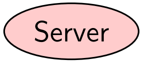


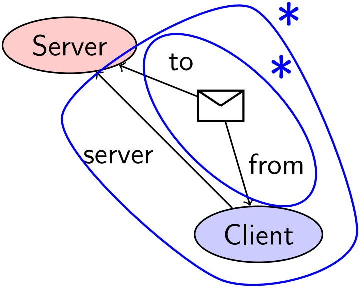
Figure: Example of a flat program

DBP are intrinsically not flat.

initial configuration:



covering set:

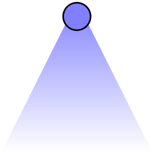


How many steps are there between the initial configuration and the final configuration ?
ω^2 steps

Hence, we need to consider nested loops if we want to compute the covering set.

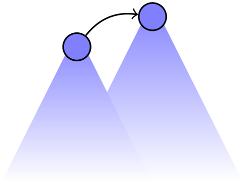
From acceleration to widening

Acceleration considers transitions. Widening only states.

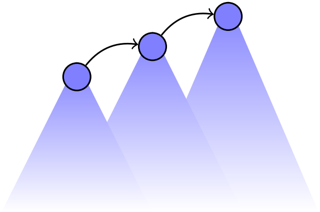


From acceleration to widening

Acceleration considers transitions. Widening only states.

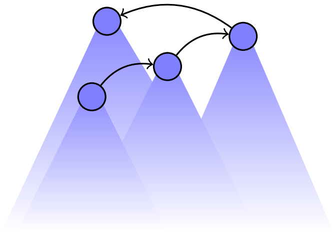


Acceleration considers transitions. Widening only states.

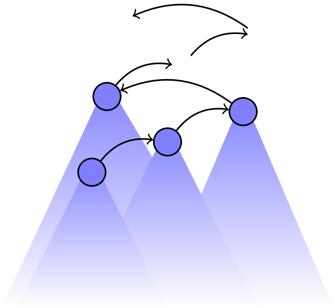


From acceleration to widening

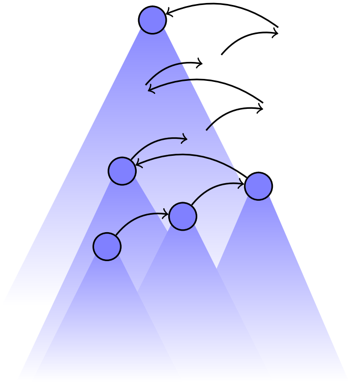
Acceleration considers transitions. Widening only states.



Acceleration considers transitions. Widening only states.

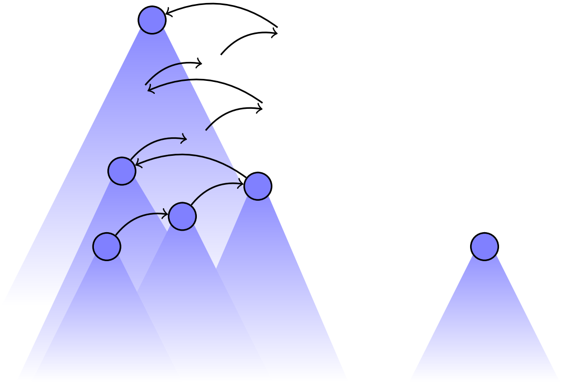


Acceleration considers transitions. Widening only states.

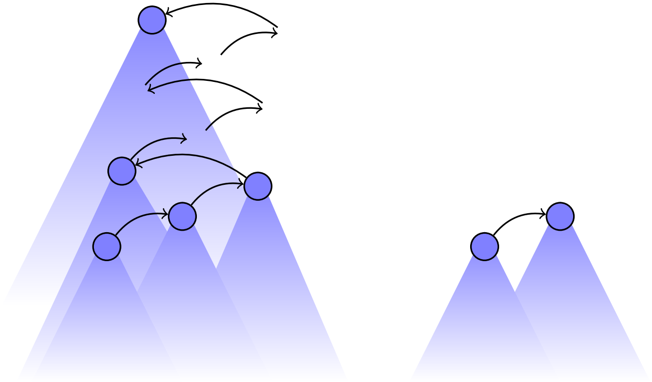


From acceleration to widening

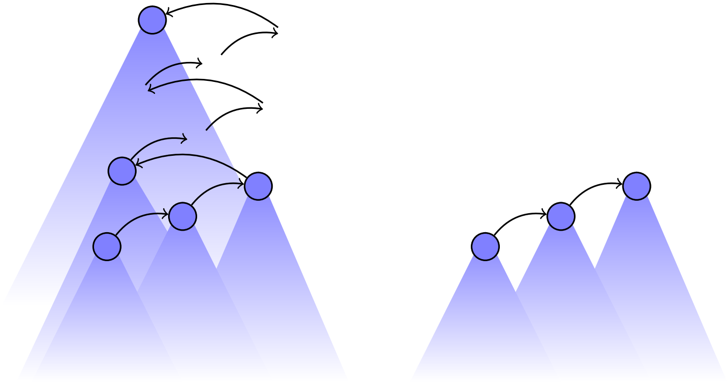
Acceleration considers transitions. Widening only states.



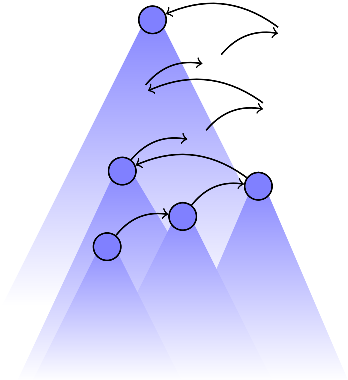
Acceleration considers transitions. Widening only states.

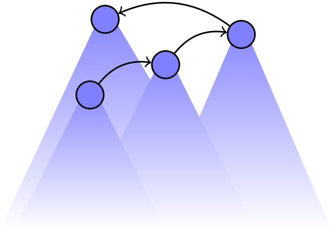


Acceleration considers transitions. Widening only states.



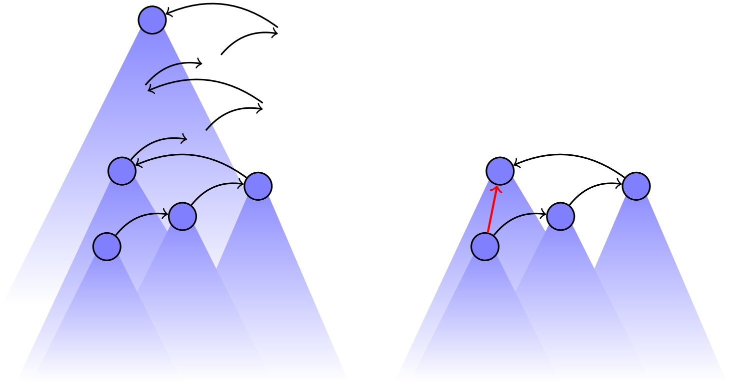
Acceleration considers transitions. Widening only states.



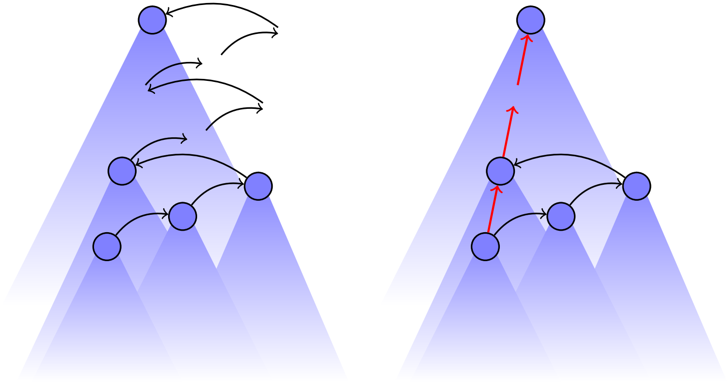


From acceleration to widening

Acceleration considers transitions. Widening only states.



Acceleration considers transitions. Widening only states.



Abstract interpretation: Domains

- Concrete domain: $D = \mathcal{P}(S)$
- Abstract domain: $D_{\downarrow} = \{ \downarrow X \mid X \subseteq S \}$

The abstract domain can be further refined from the set of downward-closed set to the set of ideals (downward-closed and *directed*).

- Abstract domain 2: D_{Idl}

An arbitray downward-closed set can be represented as the finite union of ideals.

Widening (1)

Goal: try to mimic acceleration (when possible), and force termination

A set-widening operator (∇) for a poset X is partial function ($\mathcal{P}(X) \to X$) that satisfies:

- Covering : for all $Y \subseteq X$, $y \in Y \Rightarrow y \leq \nabla(Y)$;
- Termination : widening of any ascending chain stabilizes.

Reason of using a set-widening operator: we need the history.

Widening (2)

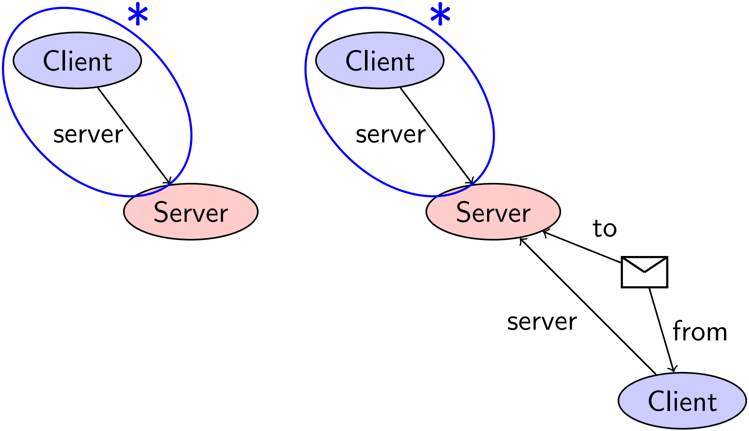
Lifting a widening operators from $Idl(S)$ to D_{Idl}: going from elements of the domain to finite powerset is non-trivial. We assume that the ordering is a *bqo*. Thus $Idl(S)$ is also a bqo.

Given an ascending chain: $C = \{L_i\}_{0 \leq i \leq n}, C \subseteq D_{Idl}$ (history)

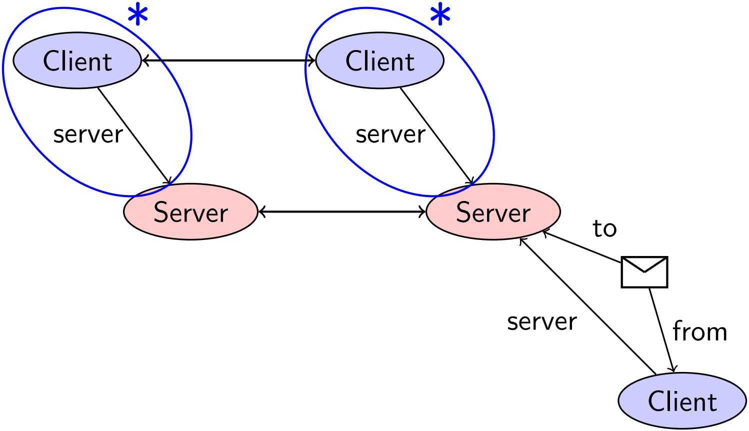
- $\nabla(\{L_0\}) = \{L_0\}$
- $\nabla(\{L_0, \ldots, L_i\}) = \nabla(\{L_0, \ldots, L_{i-1}\}) \sqcup \{\nabla_S(\mathcal{I}) \ \mid \mathcal{I} \text{ max ascending chain in} \nabla(\{L_0, \ldots, L_{i-1}\})\}$

Why a *bqo* ? To avoid having an infinite antichain in $Idl(S)$.

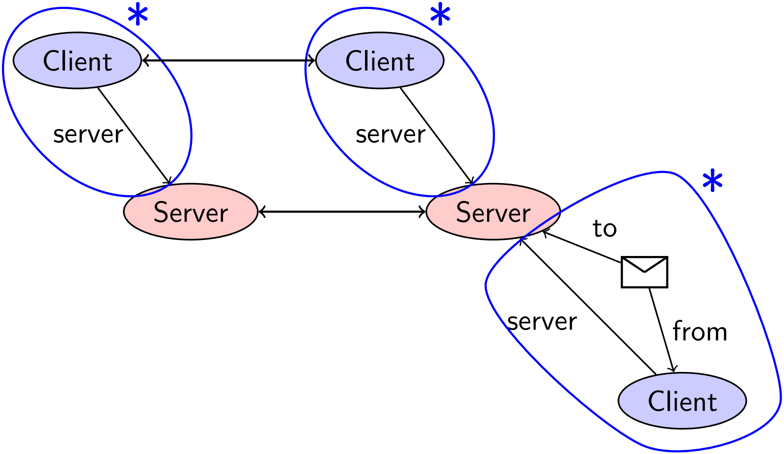
Set-widening for DBP (1)

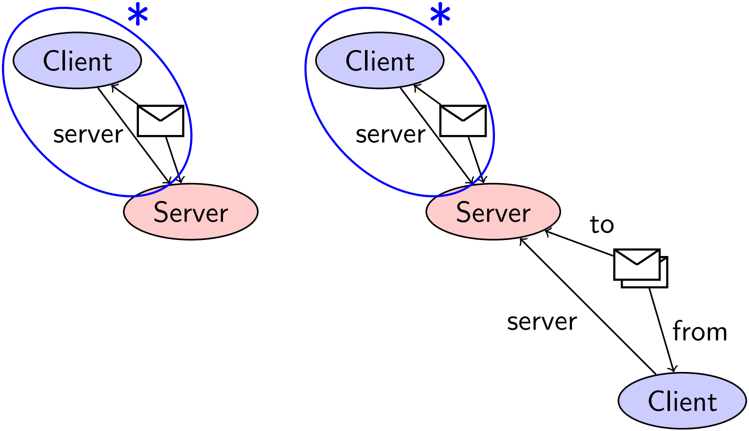


Set-widening for DBP (1)

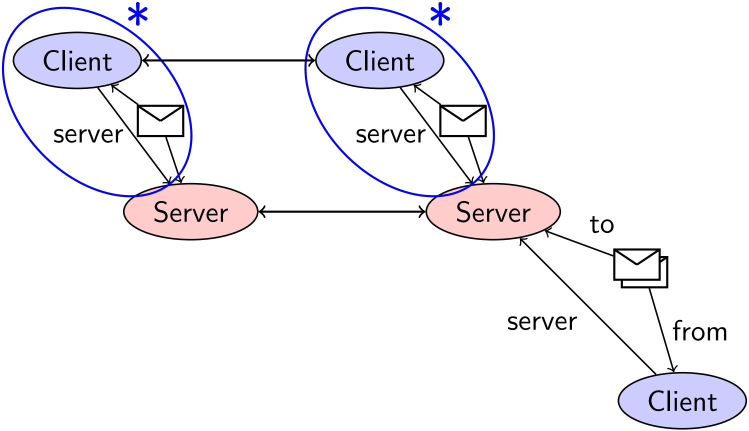


Set-widening for DBP (1)

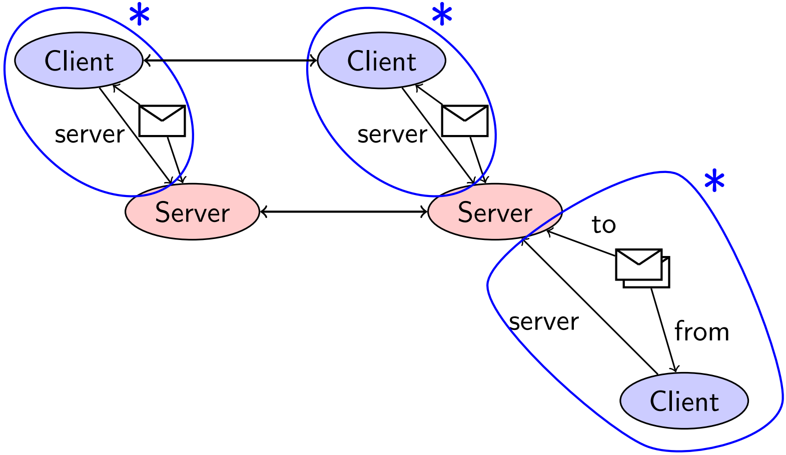



*
Client
server
Server
*
Client
server
Server
to
server
from
Client

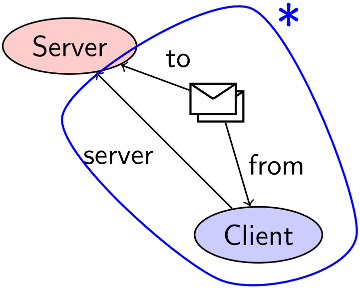
Set-widening for DBP (2)



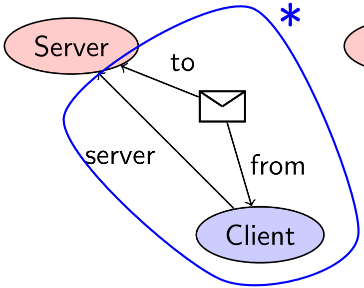
Set-widening for DBP (2)

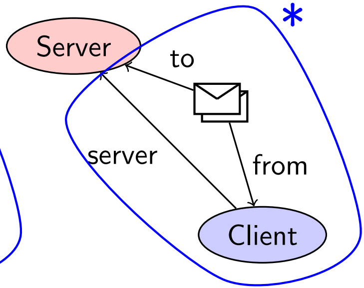


Set-widening for DBP (3)

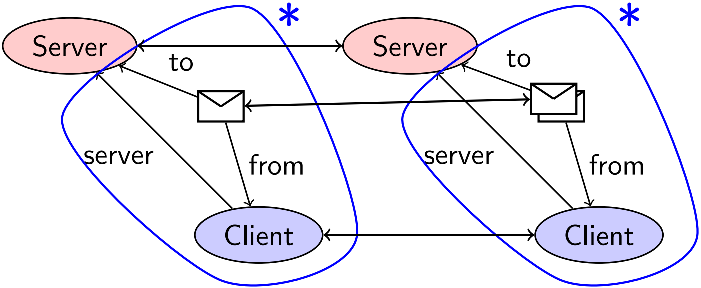


Set-widening for DBP (3)

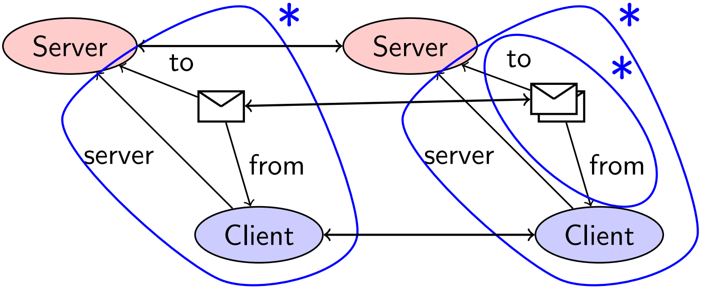




Set-widening for DBP (3)



Set-widening for DBP (3)



What about the precision ?

- Acceleration and widening seems like the *extreme* ends of some spectrum.
- Is there a class of nested loops for which we can compute exactly the result ?
- Can we get a good characterisation of the programs for which this kind of widening matches acceleration ?

- DBP is one of the largest fragment of the π-calculus for which interesting verification questions are still decidable.
- Not yet clear what is the right way of handling features such as process creation and mobility.
- WSTS approach gives decidability a result, now we are working on an efficient analysis.

Questions ?

References I

Abdulla, P. A., Cerans, K., Jonsson, B. and Tsay, Y.-K. (1996).
General Decidability Theorems for Infinite-State Systems.
In LICS pp. 313–321,.

Bardin, S., Finkel, A., Leroux, J. and Schnoebelen, P. (2005).
Flat Acceleration in Symbolic Model Checking.
In ATVA pp. 474–488,.

Dufourd, C., Finkel, A. and Schnoebelen, P. (1998).
Reset Nets Between Decidability and Undecidability.
In ICALP pp. 103–115,.

Finkel, A. and Goubault-Larrecq, J. (2009).
Forward Analysis for WSTS, Part I: Completions.
In STACS vol. 09001, of Dagstuhl Sem. Proc. pp. 433–444,.

Finkel, A. and Schnoebelen, P. (2001).
Well-structured transition systems everywhere!
Theor. Comput. Sci. *256*, 63–92.

Geeraerts, G., Raskin, J.-F. and Van Begin, L. (2006).
Expand, Enlarge and Check: New algorithms for the coverability problem of WSTS.
J. Comput. Syst. Sci. *72*, 180–203.

Hewitt, C., Bishop, P. and Steiger, R. (1973).
A Universal Modular ACTOR Formalism for Artificial Intelligence.
In IJCAI pp. 235–245,.

References II

Honda, K. and Tokoro, M. (1991).
An Object Calculus for Asynchronous Communication.
In ECOOP pp. 133–147,.

Meyer, R. (2008).
On Boundedness in Depth in the pi-Calculus.
In IFIP TCS vol. 273, of IFIP pp. 477–489, Springer.

Milner, R., Parrow, J. and Walker, D. (1992a).
A Calculus of Mobile Processes, I.
Inf. Comput. *100*, 1–40.

Milner, R., Parrow, J. and Walker, D. (1992b).
A Calculus of Mobile Processes, II.
Inf. Comput. *100*, 41–77.

Wies, T., Zufferey, D. and Henzinger, T. A. (2010).
Forward Analysis of Depth-Bounded Processes.
In FoSSaCS 2010 vol. 4349, of LNCS pp. 94–108, Springer.