Verification of Concurrent Asynchronous Message-passing Programs

Tom Henzinger Thomas Wies Damien Zufferey

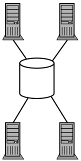
École Polytechnique Fédérale de Lausanne

June 24, 2009

Outline

Shared memory

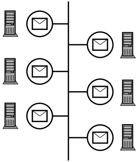
Communication using a memory that every process can access (read and write).



+ Fast
− Limited scaling
− Hard to program (deadlocks, races, ...)

Message passing

Processes exchange messages.



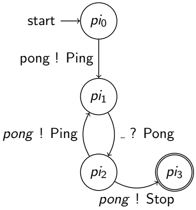
+ Scales well
− Slower
∼ Hard to program (easier than shared memory ?)

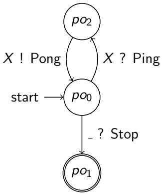
Example (1): scala/docs/examples/actors/pingpong.scala

```
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





Objectives and Contributions

Objectives

- Identifying fragments:
  - interesting for the programmer;
  - where verification related problems are decidable.
- Algorithms *fast enough* to check properties of those fragments.

Contributions

- An heuristic to check for Deadlock freedom of static systems.
- A new kind of system: Star Topologies
- A semi-algorithm for control reachability in Star Topologies.

Outline

$A\pi$-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 & Tokoro, 1991, Boudol et al., 1992] is a restriction of the π-calculus.

It is build around the notions of

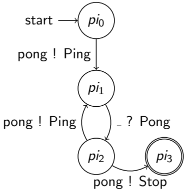
- Names : channels as first class values.
- Threads : concurrent execution of parallel threads: $P \mid Q$.
- i/o prefixes : sending/receiving messages.

$A\pi$-calculus: Syntax

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

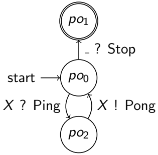
$A\pi$-calculus: Example (1)

$$
\begin{array}{lcl}
pi_0 & = & \overline{pong_{Ping}}\langle ping_{Pong}\rangle | pi_1 \\
pi_1 & = & ping_{Pong}().pi_2 \\
pi_2 & = & pi_{2a} \oplus pi_{2b} \\
pi_{2a} & = & \overline{pong_{Ping}}\langle ping_{Pong}\rangle | pi_1 \\
pi_{2b} & = & \overline{pong_{Stop}}\langle\rangle | pi_3 \\
pi_3 & = & 0
\end{array}
$$



$A\pi$-calculus: Example (2)

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



$A\pi$-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:

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

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:

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

What happens:

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

Overview

The Actor Model [Hewitt et al., 1973, Clinger, 1981, Agha, 1986] uses *actors* and their interactions to build concurrent softwares.

An actor can:

- send finitely many messages to other actors.
- create a finite number of new actors.
- receive a message from its mailbox and continue with a specified behaviour.

General Actor Systems

What can an actor do ?

Receive	$P(\vec{a}_I;\vec{a}_O) = \sum_{i\in I} a_i(\vec{b}_i).P_i(\vec{a}_I;\vec{a}_O,\vec{b}_i) \quad \forall i\in I, a_i \in \vec{a}_I$
Send	$P(\vec{a}_I;\vec{a}_O) = \bar{a}\langle\vec{b}\rangle \mid P'(\vec{a}_I;\vec{a}_O)$
Branch	$P(\vec{a}_I;\vec{a}_O) = A(\vec{a}_I;\vec{a}_O) \oplus B(\vec{a}_I;\vec{a}_O)$
New channel	$P(\vec{a}_I;\vec{a}_O) = (\nu\vec{a}).P'(\vec{a}_I,\vec{a};\vec{a}_O)$
New actor	$P(\vec{a}_I;\vec{a}_O) = (\nu\vec{n})(Q(\vec{n};\emptyset) \mid P'(\vec{a}_I;\vec{a}_O,\vec{n}))$

Unique Receiver Condition [Amadio, 2000]

In π-calculus threads and names are independent.
In the actor model an actor does not share its mailbox.

The unique receiver condition links channels with a thread. It is a syntactic restriction where names that belong to a thread are kept separately.

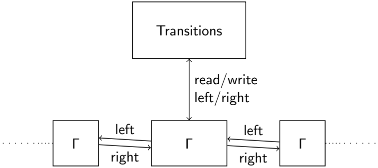
General Actor Systems

What can an actor do ?

Receive	$P(\vec{a}_I;\vec{a}_O) = \sum_{i\in I} a_i(\vec{b}_i).P_i(\vec{a}_I;\vec{a}_O,\vec{b}_i) \quad \forall i\in I, a_i \in \vec{a}_I$
Send	$P(\vec{a}_I;\vec{a}_O) = \bar{a}\langle\vec{b}\rangle \mid P'(\vec{a}_I;\vec{a}_O)$
Branch	$P(\vec{a}_I;\vec{a}_O) = A(\vec{a}_I;\vec{a}_O) \oplus B(\vec{a}_I;\vec{a}_O)$
New channel	$P(\vec{a}_I;\vec{a}_O) = (\nu\vec{a}).P'(\vec{a}_I,\vec{a};\vec{a}_O)$
New actor	$P(\vec{a}_I;\vec{a}_O) = (\nu\vec{n})(Q(\vec{n};\emptyset) \mid P'(\vec{a}_I;\vec{a}_O,\vec{n}))$

Reachability is undecidable for General Actor Systems.

Encoding a Turing machine into an actor system.



Outline

Static Actor Systems

Systems of actors without creation of channels or actors are *static*. It corresponds to $A\pi$-calculus without creation of names. This fragment of $A\pi$-calculus reduces to Petri nets [Amadio & Meyssonnier, 2002].

Remark

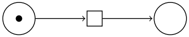
Without name creation, a general `!?` + `reply` mechanism is not possible. However, we can emulate it when there is no more than one `reply` per received message.

Petri Nets

Petri nets are modeling language for discrete distributed systems. A Petri net (S, T, F, M) is a directed bipartite graph where

- S a finite set of *places*;
- T a finite set of *transitions*;
- F is the flow relation, $F \subseteq (S \times T) \bigcup (T \times S) \rightarrow \{0, 1\}$;
- M is a marking, $M : S \rightarrow \mathbb{N}$.

Places may contain some tokens, that are consumed and created by transitions.

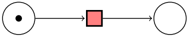


Petri Nets

Petri nets are modeling language for discrete distributed systems.
A Petri net (S, T, F, M) is a directed bipartite graph where

- S a finite set of *places*;
- T a finite set of *transitions*;
- F is the flow relation, $F \subseteq (S \times T) \bigcup (T \times S) \rightarrow \{0, 1\}$;
- M is a marking, $M : S \rightarrow \mathbb{N}$.

Places may contain some tokens, that are consumed and created by transitions.



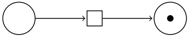
Outline
Introduction
Actor Systems
Deadlock Freedom of *Static* Actor Systems
Extensions to *Dynamic* Actor Systems
Static Actor Systems
Petri Nets
Structural Analysis of Petri Nets

Petri Nets

Petri nets are modeling language for discrete distributed systems.
A Petri net (S, T, F, M) is a directed bipartite graph where

- S a finite set of *places*;
- T a finite set of *transitions*;
- F is the flow relation, $F \subseteq (S \times T) \bigcup (T \times S) \rightarrow \{0, 1\}$;
- M is a marking, $M : S \rightarrow \mathbb{N}$.

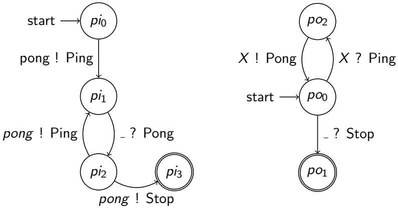
Places may contain some tokens, that are consumed and created by transitions.



Tom Henzinger, Thomas Wies, Damien Zufferey Verifying actors 22 / 54

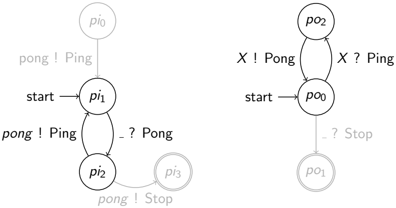
Outline
Introduction
Actor Systems
Deadlock Freedom of *Static* Actor Systems
Extensions to *Dynamic* Actor Systems
Static Actor Systems
Petri Nets
Structural Analysis of Petri Nets

Reachability: *Static* Actor Systems to Petri Nets (1)



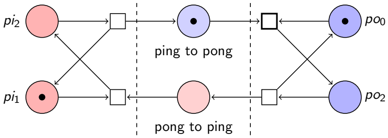
Outline
Introduction
Actor Systems
Deadlock Freedom of *Static* Actor Systems
Extensions to *Dynamic* Actor Systems
Static Actor Systems
Petri Nets
Structural Analysis of Petri Nets

Reachability: *Static* Actor Systems to Petri Nets (1)



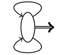
Tom Henzinger, Thomas Wies, Damien Zufferey Verifying actors 23 / 54

Reachability: *Static* Actor Systems to Petri Nets (2)

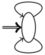


Structural Properties

Siphon : Set of places S such that $\bullet S \subseteq S \bullet$



Trap : Set of places T such that $T \bullet \subseteq \bullet T$.

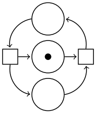


P-Invariant : (nonzero integer vector I, integer X) such that

$$\forall M \text{ marking}, M_0 \to^* M : \sum_{p \in P} I(p) \cdot M(p) = X$$

Necessary Condition for Deadlock [Commoner, 1972]

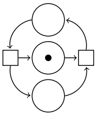
Deadlock: at least one incoming place per transition is empty.



Necessary Condition for Deadlock [Commoner, 1972]

Deadlock: at least $\underbrace{\text{one incoming place per transition}}_{\downarrow}$ is empty.

global siphon

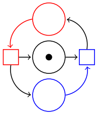


Necessary Condition for Deadlock [Commoner, 1972]

Deadlock: at least one incoming place per transition is empty.

global siphon

empty global siphon



Trap, Invariant, ILP, SAT

Algorithm 1 Check Petri net for deadlock freedom

Require: $P = (S, T, F, M)$ a petri net
Ensure: returns Yes if P is deadlock free

$\phi \leftarrow \bigwedge_{s\in S}\left(s \Rightarrow \bigwedge_{t\in\bullet s}\left(\bigvee_{p\in\bullet t} p\right)\right) \wedge finish$
while $R = \text{solve}(\phi)$ **do**
  if is maximal trap in R marked ? **then**
    $t \leftarrow$ smallest marked trap in R
    $\phi \leftarrow \phi \wedge \neg t$
  else if $\exists(I, X)$ controlling R **then**
    $t \leftarrow \bigwedge_{s\in R \wedge I(s)\neq 0} s$
    $\phi \leftarrow \phi \wedge \neg t$
  else if ILP approximation has a solution **then**
    $\phi \leftarrow \phi \wedge \neg R$
  else
    return NO
  end if
end while
return Yes

Trap, Invariant, ILP, SAT

Algorithm 2 Check Petri net for deadlock freedom

Require: $P = (S, T, F, M)$ a petri net
Ensure: returns Yes if P is deadlock free

$\phi \leftarrow \bigwedge_{s\in S}\left(s \Rightarrow \bigwedge_{t\in\bullet s}\left(\bigvee_{p\in\bullet t} p\right)\right) \wedge finish$ ← Enumerating siphons with SAT solver
while $R = \text{solve}(\phi)$ **do**
  if is maximal trap in R marked ? **then**
    $t \leftarrow$ smallest marked trap in R
    $\phi \leftarrow \phi \wedge \neg t$
  else if $\exists(I, X)$ controlling R **then**
    $t \leftarrow \bigwedge_{s\in R \wedge I(s)\neq 0} s$
    $\phi \leftarrow \phi \wedge \neg t$
  else if ILP approximation has a solution **then**
    $\phi \leftarrow \phi \wedge \neg R$
  else
    return NO
  end if
end while
return Yes

Trap, Invariant, ILP, SAT

Algorithm 3 Check Petri net for deadlock freedom

Require: $P = (S, T, F, M)$ a petri net
Ensure: returns Yes if P is deadlock free

$\phi \leftarrow \bigwedge_{s\in S}\left(s \Rightarrow \bigwedge_{t\in\bullet s}\left(\bigvee_{p\in\bullet t} p\right)\right) \wedge finish$
while $R = \text{solve}(\phi)$ **do**
  if is maximal trap in R marked ? **then**
    $t \leftarrow$ smallest marked trap in R
    $\phi \leftarrow \phi \wedge \neg t$
  else if $\exists(I, X)$ controlling R **then** ← Checking siphon for emptiness
    $t \leftarrow \bigwedge_{s\in R \wedge I(s)\neq 0} s$
    $\phi \leftarrow \phi \wedge \neg t$
  else if ILP approximation has a solution **then**
    $\phi \leftarrow \phi \wedge \neg R$
  else
    return NO
  end if
end while
return Yes

Trap, Invariant, ILP, SAT

Algorithm 4 Check Petri net for deadlock freedom

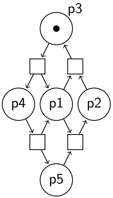
Require: $P = (S, T, F, M)$ a petri net
Ensure: returns Yes if P is deadlock free

$$\phi \leftarrow \bigwedge_{s \in S} \left(s \Rightarrow \bigwedge_{t \in \bullet s} \left(\bigvee_{p \in \bullet t} p \right) \right) \wedge \mathit{finish}$$

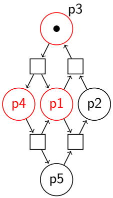
while $R = \text{solve}(\phi)$ **do**
  if is maximal trap in R marked ? **then**
    $t \leftarrow$ smallest marked trap in R
    $\phi \leftarrow \phi \wedge \neg t$
  else if $\exists (I, X)$ controlling R **then**
    $t \leftarrow \bigwedge_{s \in R \wedge I(s) \neq 0} s$
    $\phi \leftarrow \phi \wedge \neg t$
  else if ILP approximation has a solution **then**
    $\phi \leftarrow \phi \wedge \neg R$
  else
    return NO
  end if
end while
return Yes

Refining enumeration

Example



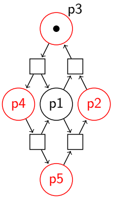
Example



Siphon p1,p3,p4

- has no marked trap.
- has no place invariant.
- has no ILP solution.

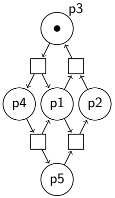
Example



Siphon p2,p3,p4,p5

- has marked trap p2,p3,p4,p5.

Example



No more siphon.
Petri net is deadlock free.

Results

Name	#places	#transitions	Deadlock	Tina[1]	heuristic
Philo 2	129	232	Yes	0.4 s	0.4 s
Philo 3	265	650	Yes	62.4 s	0.4 s
Philo 4	449	1398	Yes	—	0.6 s
Philo 8	1665	9610	Yes	—	28 s
Pi approx 3	130	260	No	0.2 s	5.8 s
Pi approx 4	204	490	No	1 s	137 s
Pi approx 5	294	822	No	11 s	—
Pi approx 6	400	1274	No	121 s	—
Pi approx 8	660	2610	No	—	—
Cell safe	34	38	No	—	0.1 s
Cell safe compact	4	26	No	6.1 s	0.1 s
Cell unsafe	34	29	Yes	0.1 s	0.1 s
Cell unsafe compact	4	17	Yes	0.1 s	0.1 s

[1] [Berthomieu & Vernadat, 2006]

Outline

Static Actor Systems: Too Restrictive

Unfortunately, most real life programs are not static.
They also create actors.

We explore two cases:

- Parametric systems
- Systems with creation of actors

Parametric Systems

A parametric system is a function $\mathcal{F}$ that for any $n \in \mathbb{N}$ maps to a static system $\mathcal{F}(n)$. The type of problem we are interested in is:

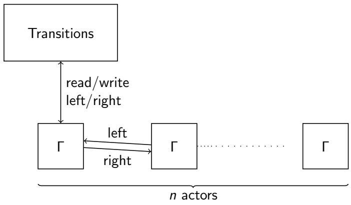
Given $\mathcal{F} : \mathbb{N} \rightarrow$ Static System, P a safety property, prove:

$$\forall n \in \mathbb{N}, \quad \mathcal{F}(n) \text{ verifies } P$$

Unfortunately, deciding such problem is in general not possible [Apt & Kozen, 1986].

$\lim_{n\rightarrow\infty} \mathcal{F}(n)$ is Turing Complete.

It is possible to encode a n-bounded Turing machine within $\mathcal{F}(n)$.



Dynamic Creation of Actors

What we want

Adding equations of the kind:

$$P(\vec{a}_I; \vec{a}_O) = (\nu\vec{n})(Q(\vec{n};\emptyset)|P'(\vec{a}_I; \vec{a}_O, \vec{n}))$$

Consequence: Without any restriction, we can still apply the Turing machine construction for general actors systems.

Restrictions

Restrictions

- Limiting name mobility (1 hop)
- Only a finite subset of all actors are allowed to create actors.

What it does: Created actors are isolated from each other.
(i.e. no recursive infinite structure)

Star Topologies: Definition

$$P(\vec{a}_I;\vec{a}_O) = \sum_{i\in I} a_i(\vec{b}_i).P_i(\vec{a}_I;\vec{a}_O,\vec{b}_i) \quad \forall i\in I, a_i\in\vec{a}_I \quad (1)$$

$$P(\vec{a}_I;\vec{a}_O) = \overline{a}\langle\vec{b}\rangle|P'(\vec{a}_I;\vec{a}_O) \quad \forall b\in\vec{b}, b\notin\vec{a}_O \quad (2)$$

$$P(\vec{a}_I;\vec{a}_O) = A(\vec{a}_I;\vec{a}_O)\oplus B(\vec{a}_I;\vec{a}_O) \quad (3)$$

$$P(\vec{a}_I;\vec{a}_O) = (\nu\vec{n})(Q(\vec{n};\emptyset)|P'(\vec{a}_I;\vec{a}_O,\vec{n})) \quad (4)$$

Static actors can have all types of equations, but they cannot be created.

Dynamic actors are created. However, they cannot have equations of type (4).

Outline
Introduction
Actor Systems
Deadlock Freedom of *Static* Actor Systems
Extensions to *Dynamic* Actor Systems
Parametric Systems
Star Topologies

Star Topologies: Analysis

Definition (Control Flow Reachability)

Given a system of equation in $A\pi$-calculus containing an equation identifier A and an initial configuration P, the control flow reachability problem asks whether it is possible to reach a configuration P' *containing* A:
$P \rightarrow^* P'$ where P' is a process of the shape $\ldots|A(\ldots)|\ldots$.

Outline
Introduction
Actor Systems
Deadlock Freedom of *Static* Actor Systems
Extensions to *Dynamic* Actor Systems
Parametric Systems
Star Topologies

Star Topologies: Issues

2 dimensions of unboundedness:

- unbounded number of actors;
- unbounded number of messages in each actor's mailbox.

Static actor systems have only the message dimension.

If we can remove one dimension, we can use algorithms similar to the static case.

Outline
Introduction
Actor Systems
Deadlock Freedom of *Static* Actor Systems
Extensions to *Dynamic* Actor Systems
Parametric Systems
Star Topologies

Star Topologies: Dual Abstraction

Remove the messages related source of unboundedness with counter abstractions.

drop-counter : 0 to bound then drop
∞-counter : 0 to bound then ∞

drop-abstraction counts the messages of dynamic actors with *drop*-counters.

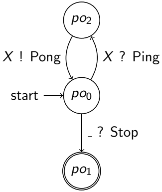
∞-abstraction counts the messages of dynamic actors with ∞-counters.

Star Topologies: Equivalence Classes of Dynamic Actors

Transforming a dynamic actor d with an unbounded number of messages into a finite object.

1. From dynamic actors to equivalence classes:
   - Fetch all messages containing a channel owned by d.
   - Substitute owned channels by `Self`. It corresponds to alpha-conversion in the π-calculus congruence relation.
2. Apply the counter abstraction on each message kind to abstract infinitely many configuration into a finite number of possibilities.

Star Topologies: Example of Equivalence Classes

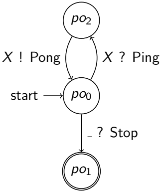


One static ping actor
Many dynamic pong$_i$ actors

- At location po_2
- Parameter X = ping
- Messages:

From	Type	To	#
ping	Ping	pong$_n$	1
pong$_n$	Pong	ping	5

Star Topologies: Example of Equivalence Classes

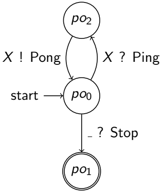


One static ping actor
Many dynamic pong$_i$ actors

- At location po_2
- Parameter X = ping
- Messages:

From	Type	To	#
ping	Ping	Self	1
Self	Pong	ping	5

Star Topologies: Example of Equivalence Classes



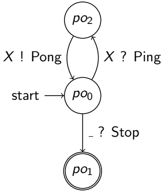
One static ping actor
Many dynamic pong$_i$ actors

- At location po_2
- Parameter X = ping
- Messages:

From	Type	To	#
ping	Ping	Self	1
Self	Pong	ping	5

Bound of abstraction is 3.

Star Topologies: Example of Equivalence Classes



One static ping actor
Many dynamic pong$_i$ actors

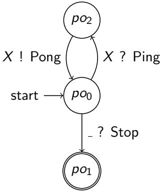
drop-abstraction

- At location po_2
- Parameter X = ping
- Messages:

From	Type	To	#
ping	Ping	Self	1
Self	Pong	ping	3

Bound of abstraction is 3.

Star Topologies: Example of Equivalence Classes



One static ping actor
Many dynamic pong$_i$ actors

∞-abstraction

- At location po_2
- Parameter X = ping
- Messages:

From	Type	To	#
ping	Ping	Self	1
Self	Pong	ping	∞

Bound of abstraction is 3.

Well Structured Transition System (WSTS)

WSTS are a generalisation of Petri nets that keep the monotonicity properties of Petri nets [Abdulla et al., 1996, Finkel & Schnoebelen, 2001].

The transition relation $\rightarrow$, *wqo* $\leq$:

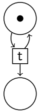
$$A \leq B \wedge A \rightarrow A' \Rightarrow \exists B'. B \rightarrow B' \wedge A' \leq B'$$

The property p also needs to satisfy some monotonicity condition:

$$A \text{ satisfies } p \wedge A \leq B \Rightarrow B \text{ satisfies } p$$

Star Topologies: Coverability Tree

Coverability Trees are used to analyse Petri nets [Karp & Miller, 1969, Finkel, 1991]. The idea was generalized to WSTS [Finkel, 1987].



(1, 0)

Star Topologies: Coverability Tree

Coverability Trees are used to analyse Petri nets [Karp & Miller, 1969, Finkel, 1991]. The idea was generalized to WSTS [Finkel, 1987].

Star Topologies: Coverability Tree

Coverability Trees are used to analyse Petri nets [Karp & Miller, 1969, Finkel, 1991]. The idea was generalized to WSTS [Finkel, 1987].

Star Topologies: Coverability Tree

Coverability Trees are used to analyse Petri nets [Karp & Miller, 1969, Finkel, 1991]. The idea was generalized to WSTS [Finkel, 1987].

Star Topologies: $\leq$

We need a $\leq$ relation that is a well-quasi-ordering,
such that the control flow reachability is monotonic w.r.t. $\leq$
and the transition relation is also monotonic w.r.t. $\leq$.

$\leq$ is oriented around two axes:

- A configuration with more dynamic actor can do more.
- A configuration with more messages can do more.

Star Topologies: Under-approximation of *drop*-abstraction

Idea: With the *drop*-counters some messages are removed, the rest is identical. Furthermore, a configuration with less messages is covered by one with more messages.

Lemma

For all realisable paths p_a in the drop-abstraction A, there is a path p_c in the corresponding concrete system C such that the configurations of p_c cover the configurations in p_a.

Star Topologies: Over-approximation of ∞-abstraction

Idea: With the ∞-counters messages can be added, the rest is identical. Furthermore, a configuration were messages are added covers the original one.

Lemma

For all realisable paths p_c in a concrete system C, there is a path p_a in the corresponding ∞-abstraction A such that the configurations of p_a cover the configurations in p_c.

Star Topologies: Analysis Idea

We have:

$$\textit{drop}\text{-abstraction} \leq \text{concrete system} \leq \infty\text{-abstraction}$$

We need a condition for:

$$\infty\text{-abstraction} \leq \textit{drop}\text{-abstraction}$$

To prove that the analysis has enough precision.

Star Topologies: Semi-algorithm

Algorithm 5 Control flow reachability

Require: C a system of actors, q a control flow location to cover
Ensure: returns the answer to is q covered in some execution of C
$n \leftarrow 1$
repeat
  $n \leftarrow n + 1$
  $D \leftarrow$ *drop*-abstraction(C, n)
  $I \leftarrow \infty$-abstraction(C, n)
  $tree_D \leftarrow$ coverabilityTree(D)
  $tree_I \leftarrow$ coverabilityTree(I)
until $tree_D \approx tree_I$
return $q \in tree_D$

Star Topologies: Agreement of Abstractions

$tree_D \approx tree_I$ when:

for all path p_i in the coverability tree of I,
there is a path p_d in the coverability tree of D such that

- both p_i and p_d have length k;
- $\forall i \in [0, k-1]$, ith transition φ_i is the same for p_i and p_d;
- $\forall i \in [0, k-1]$, ith configuration I_i, D_i are similar ($I_i \sim D_i$).

Remark

We use $I_i \sim D_i$ and not $\leq$. $\sim$ is defined to ignore messages of dynamic actors.

Star Topologies: Soundness

Theorem (Soundness)

Given a system C, a bound $n \in \mathbb{N}$, the corresponding drop-abstraction D and ∞-abstraction I, and a control flow location q to cover. If the coverability trees of D and I agree then the answer to whether q is covered can be accurately computed from the tree of D.

Conclusion and Future Work

- Making the agreement condition more flexible (i.e. more robust w.r.t. optimisation in building the tree)
- Proving the completeness of the algorithm
- Reachability problem for star topologies
- Other communication topologies
- Is is possible to find restrictions for parametric systems ?

References I

Abdulla, P. A., Cerans, K., Jonsson, B. & Tsay, Y.-K. (1996).
In LICS pp. 313–321,.

Agha, G. (1986).
ACTORS: A Model of Concurrent Computation in Distributed Systems.
PhD thesis, MIT CSAIL.

Amadio, R. M. (2000).
Theor. Comput. Sci. 240, 147–176.

Amadio, R. M. & Meyssonnier, C. (2002).
Nord. J. Comput. 9, 70–101.

Apt, K. R. & Kozen, D. (1986).
Inf. Process. Lett. 22, 307–309.

Berthomieu, B. & Vernadat, F. (2006).
In QEST pp. 123–124, IEEE Computer Society.

Boudol, G., Laneve, C., Laneve, C. & Meije, P. (1992).
Technical report INRIA Report 1702, INRIA, Sophia Antipolis.

Clinger, W. (1981).
Foundations of Actor Semantics.
PhD thesis, MIT CSAIL.

References II

Commoner, F. (1972).
Technical Report CA-7206-2311 Applied Data Research, Inc. Wakefield, Massachusetts.

Finkel, A. (1987).
In ICALP, (Ottmann, T., ed.), vol. 267, of Lecture Notes in Computer Science pp. 499–508, Springer.

Finkel, A. (1991).
In Applications and Theory of Petri Nets, (Rozenberg, G., ed.), vol. 674, of Lecture Notes in Computer Science pp. 210–243, Springer.

Finkel, A. & Schnoebelen, P. (2001).
Theor. Comput. Sci. 256, 63–92.

Hewitt, C., Bishop, P. & Steiger, R. (1973).
In IJCAI pp. 235–245,.

Honda, K. & Tokoro, M. (1991).
In ECOOP, (America, P., ed.), vol. 512, of Lecture Notes in Computer Science pp. 133–147, Springer.

Karp, R. M. & Miller, R. E. (1969).
J. Comput. Syst. Sci. 3, 147–195.

References III

Milner, R., Parrow, J. & Walker, D. (1992a).
Inf. Comput. 100, 1–40.

Milner, R., Parrow, J. & Walker, D. (1992b).
Inf. Comput. 100, 41–77.