Modeling and verifying Scala actors

Damien Zufferey

École Polytechnique Fédérale de Lausanne

Laboratory: Models and Theory of Computation
Supervisor: Tom Henzinger
Assistant: Thomas Wies

January 12, 2009

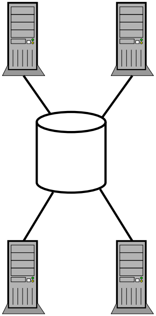
1. Introduction
   - Paradigms for concurrency
   - Actors in SCALA
2. The Actor Model
3. π-calculus and $A\pi$-calculus
4. Petri nets

Outline

1. Introduction
   - Paradigms for concurrency
   - Actors in Scala
2. The Actor Model
3. π-calculus and $A\pi$-calculus
4. Petri nets

Shared memory

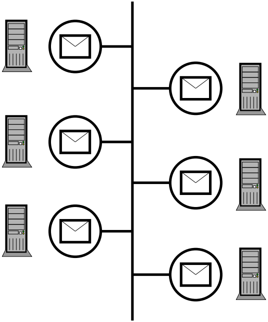
Communication using a memory that every process can access (read and write).



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

Message passing

Processes exchange messages.



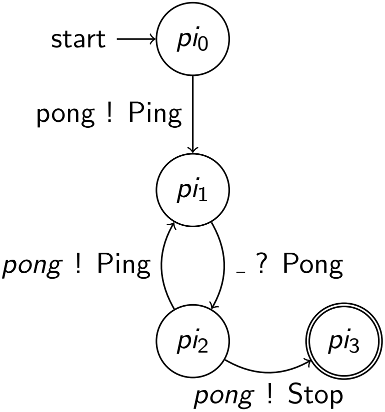
+ Scales well
– Slower
$\sim$ Hard to program (easier than shared memory ?)

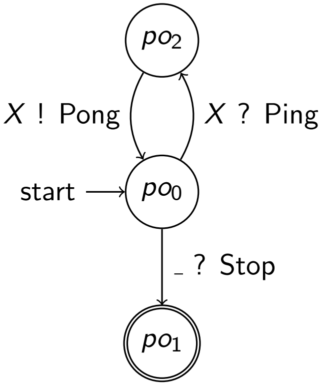
Example (1): scala/docs/examples/actors/pingpong.scala

```
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





Overview of Analysis

SCALA sources

↓ SCALA compiler { namer, typer }

AST

↓ { Unrolling of some `Seq[A]` operations. Flattening of complex expressions. Each method and function transformed into one CFA. }

CFA

↓ { Inlining of function calls. Abstracting parts that are not Actor related. Linking actors and translation into a Petri net. }

Petri net

↓ External tool: tina [Berthomieu & Vernadat, 2006]

Answer

Outline

1. Introduction
   - Paradigms for concurrency
   - Actors in SCALA
2. The Actor Model
3. π-calculus and $A\pi$-calculus
4. Petri nets

Overview

Object oriented programming uses *objects* and their interactions to build software systems.

An object can:

- receive messages
- process data
- send messages

Overview

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

An actor can:

- receive messages
- create new actors
- send messages

The meaning of messages is not the same in both contexts.

Models and decidability

Model	Reachability
Synchronous	decidable
Asynchronous with queues	undecidable [Brand & Zafiropulo, 1983]
Asynchronous with multisets	decidable[1] [Amadio & Meyssonnier, 2002]
Asynchronous with lossy queues	decidable[2] [Abdulla & Jonsson, 1996a]

With recursion, all models are undecidable [Ramalingam, 2000].

[1] fixed number of actors
[2] undecidable with fair channel [Abdulla & Jonsson, 1996b]

Chosen model and Assumptions

Finite state machines with unordered mailbox (asynchronous).

To preserve decidability, we need to add the following assumptions:

- finite data-types,
- no recursion,
- no dynamic actor creation.

Outline

1. Introduction
   - Paradigms for concurrency
   - Actors in Scala
2. The Actor Model
3. π-calculus and $A\pi$-calculus
4. Petri nets

What is it ?

The π-calculus [Milner et al., 1992a, Milner et al., 1992b] is considered to be the λ-calculus of message-passing concurrency. It tries to be minimal, but is still able to model complex features:

- synchronous and asynchronous communication
- changing topologies
- dynamic creation of processes and names

The $A\pi$-calculus [Honda & Tokoro, 1991, Boudol et al., 1992] is a restriction of π-calculus that only allows asynchronous communication.

Grammar

P	$::=$	$x(y).P$	input prefix	(receiving messages)
	$\mid$	$\overline{x}\langle y\rangle.P$	output prefix	(sending messages)
	$\mid$	$P \mid P$	parallel composition	
	$\mid$	$(\nu x)P$	name creation	(names are channels)
	$\mid$	$!P$	replication	
	$\mid$	0	unit process	(finished execution)

The only kind of output prefix allowed in $A\pi$-calculus is '$\overline{x}\langle y\rangle.0$'.

Semantic

The most important reduction rules:

$$\overline{x}\langle z\rangle.P \mid x(y).Q \rightarrow P \mid Q[z/y]$$

Some congruence rules:

- $P \mid Q \equiv Q \mid P$
- $P \mid 0 \equiv P$
- $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$
- $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$
- $(\nu x)0 \equiv 0$
- $!P \equiv P \mid !P$

Example

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0)$$

Example

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0)$$

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0)$$
$$\mid \overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0 \mid p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0$$

Example

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0)$$

$$\begin{aligned}(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0)\\ \mid \overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0 \mid p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0\end{aligned}$$

$$\begin{aligned}(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0)\\ \mid p_{ing}(p_{ong}).0 \mid \overline{p_{ing}}\langle p_{ong}\rangle.0\end{aligned}$$

Example

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0)$$

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0) \\ \mid \overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0 \mid p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0$$

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0) \\ \mid p_{ing}(p_{ong}).0 \mid \overline{p_{ing}}\langle p_{ong}\rangle.0$$

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0) \\ \mid 0 \mid 0$$

Example

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0)$$

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0) \\ \mid \overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0 \mid p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0$$

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0) \\ \mid p_{ing}(p_{ong}).0 \mid \overline{p_{ing}}\langle p_{ong}\rangle.0$$

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0) \\ \mid 0 \mid 0$$

$$(\nu p_{ing})(\nu p_{ong})!(\overline{p_{ong}}\langle p_{ing}\rangle.p_{ing}(p_{ong}).0) \mid !(p_{ong}(p_{ing}).\overline{p_{ing}}\langle p_{ong}\rangle.0)$$

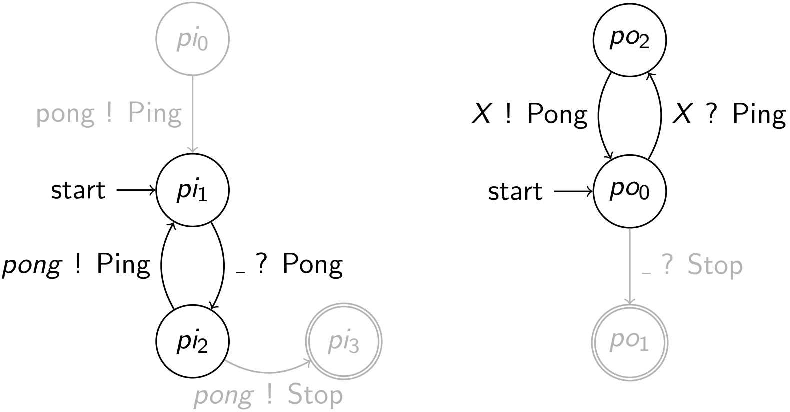
Outline

Overview

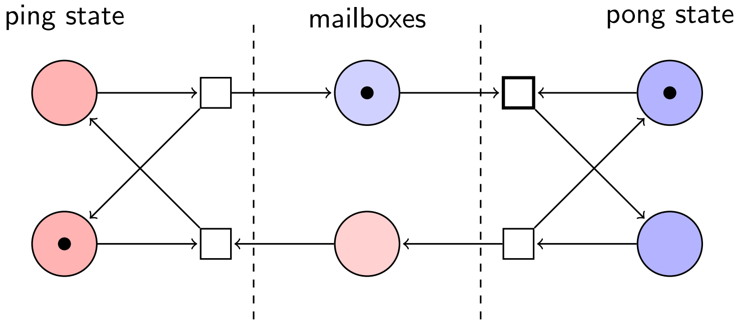
Petri nets are modeling language for discrete distributed systems.

A Petri net is a directed bipartite graph where the nodes are divided in *places* and *transitions*.
Places may contain some tokens, that are consumed and created by transitions.

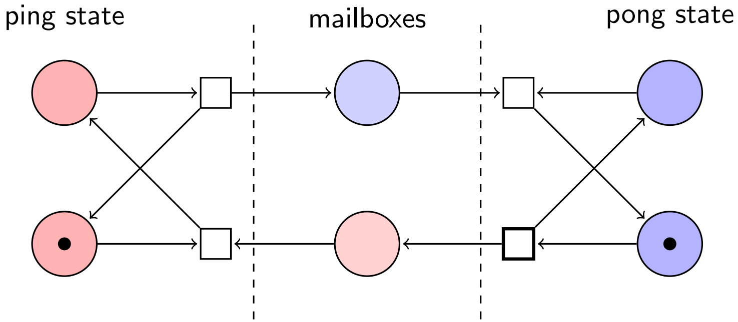
Example


pi_0
pong ! Ping
start
pi_1
pong ! Ping
_ ? Pong
pi_2
pi_3
pong ! Stop
po_2
X ! Pong
X ? Ping
start
po_0
_ ? Stop
po_1

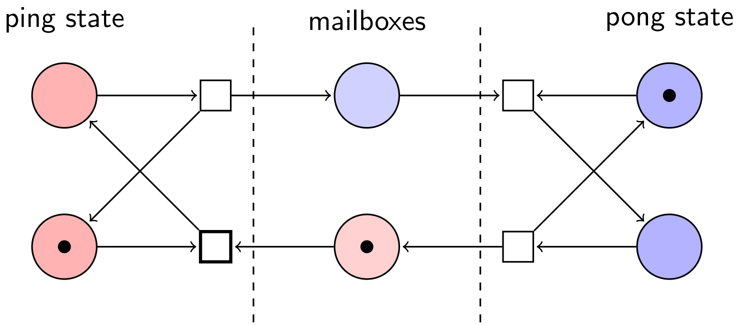
Example



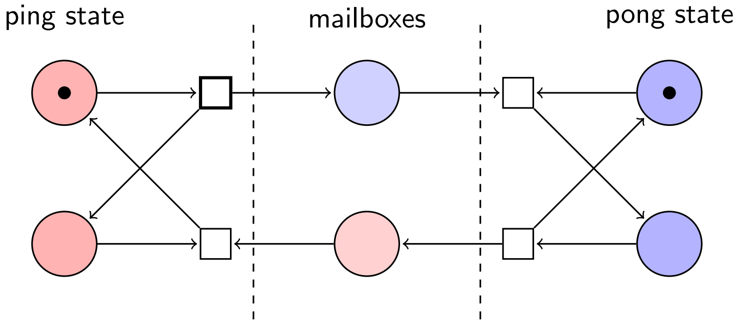
Example



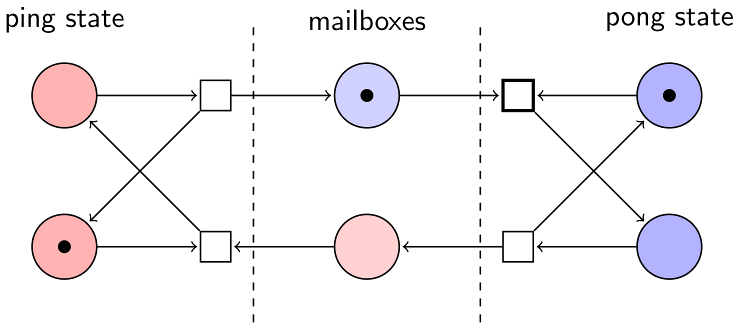
Example



Example



Example



Decidability

Problems like covering, liveness of transitions, reachability for Petri net are decidable, but very expensive (EXPSPACE-hard [Cardoza et al., 1976]).

A survey for Petri nets decidability and complexity for different problems can be found at [Esparza & Nielsen, 1994].

Summary

- Safety properties are decidable for a subset of the Actor model.
- The theoretical justification lies in the $A\pi$-calculus.
- Petri nets are used for back-end computations.
- The Implementation is done up to the translation into Petri nets.

Overview of Analysis

SCALA sources

↓ SCALA compiler { namer, typer }

AST

↓ { Unrolling of some `Seq[A]` operations. Flattening of complex expressions. Each method and function transformed into one CFA. }

CFA

↓ { Inlining of function calls. Abstracting parts that are not Actor related. Linking actors and translation into a Petri net. }

Petri net

↓ External tool: tina [Berthomieu & Vernadat, 2006]

Answer

Further Work

One limitation is the **features** that are supported.
Adding more features means going toward (and beyond) the frontier of decidability.

Another limitation of this verification method is its **complexity**.
$\Rightarrow$ Compositional verification.

Abdulla, P. A. & Jonsson, B. (1996a).
Inf. Comput. 127, 91–101.

Abdulla, P. A. & Jonsson, B. (1996b).
Inf. Comput. 130, 71–90.

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

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.

Brand, D. & Zafiropulo, P. (1983).
J. ACM 30, 323–342.

Cardoza, E., Lipton, R. J. & Meyer, A. R. (1976).
In STOC pp. 50–54, ACM.

Esparza, J. & Nielsen, M. (1994).
Elektronische Informationsverarbeitung und Kybernetik 30, 143–160.

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.