AKKA <~~> LTS [?)

Ruslan Shevchenko <ruslan@Shevchenko.Kiev.UA>

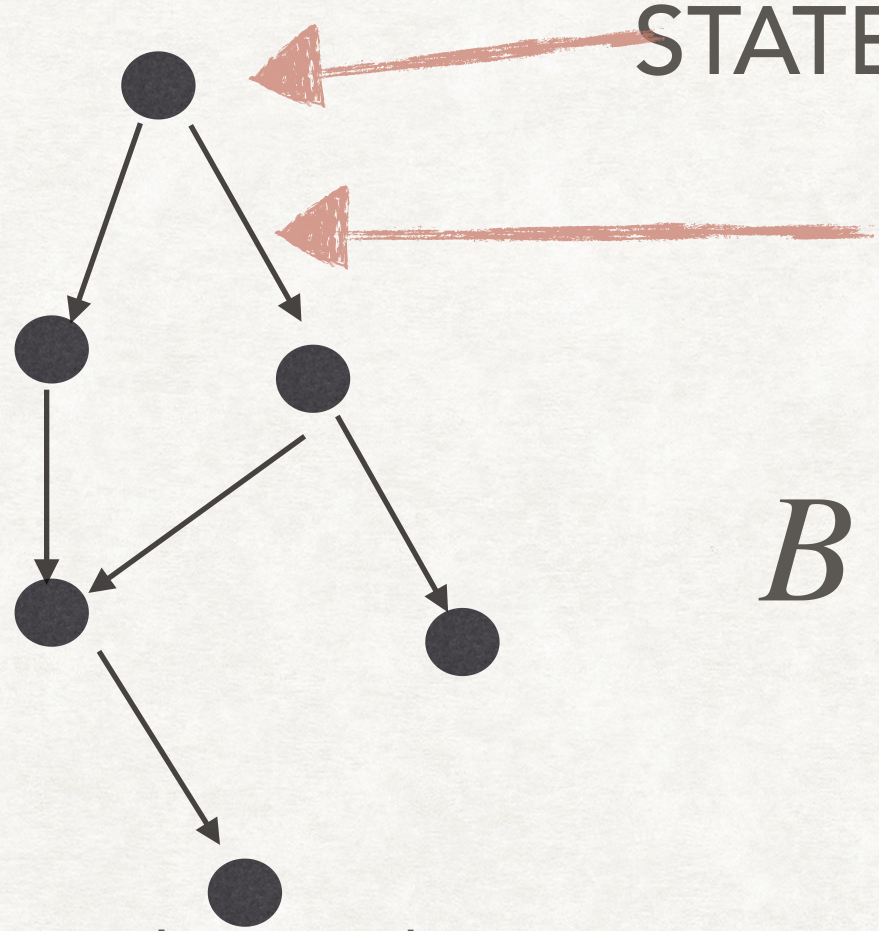
http://garuda.ai

BEHAVIOR DESCRIPTION

// BASIC QUESTIONS //

- What is a behavior ?
- How we can specify behavior ? (math / akka)
- Qustions with answer:
  - Can we bring to programming/verification well-known techniques from mathematics ?
- Questions without answer:
  - Limit of applicability. When this can be useful ?

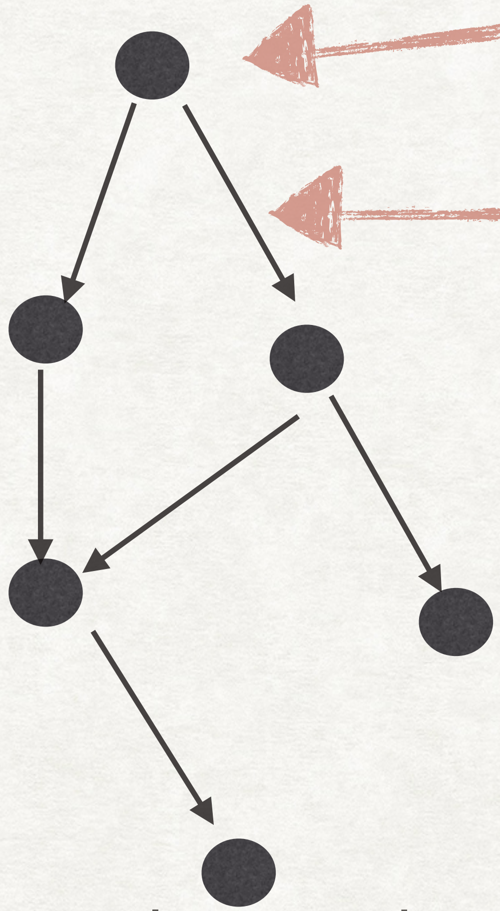
// WHAT IS A BEHAVIOR(?)//



$$B = E \times S \to S$$

BEHAVIOR DESCRIPTION

// WHAT IS A BEHAVIOR(?)//



Delta = End

STATE $s \in S$

ACTION (EVENTS) $a \in A$

$$B_{LTS} = A \times S \to S$$

$$\to_a \subseteq S \times A \times S$$

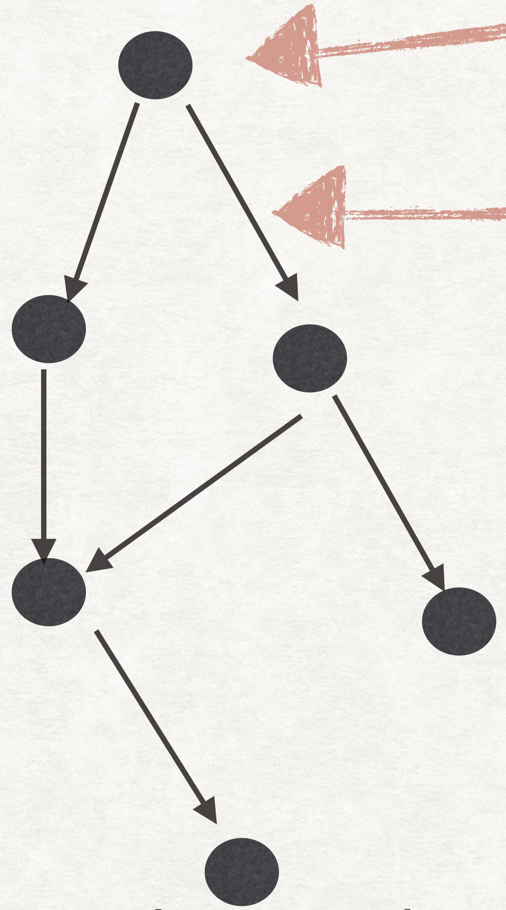
LTS = Labelled Transition System

BEHAVIOR DESCRIPTION

// WHAT IS A BEHAVIOR(?: AKKA)//

STATE

ACTION (EVENTS)



Delta = End

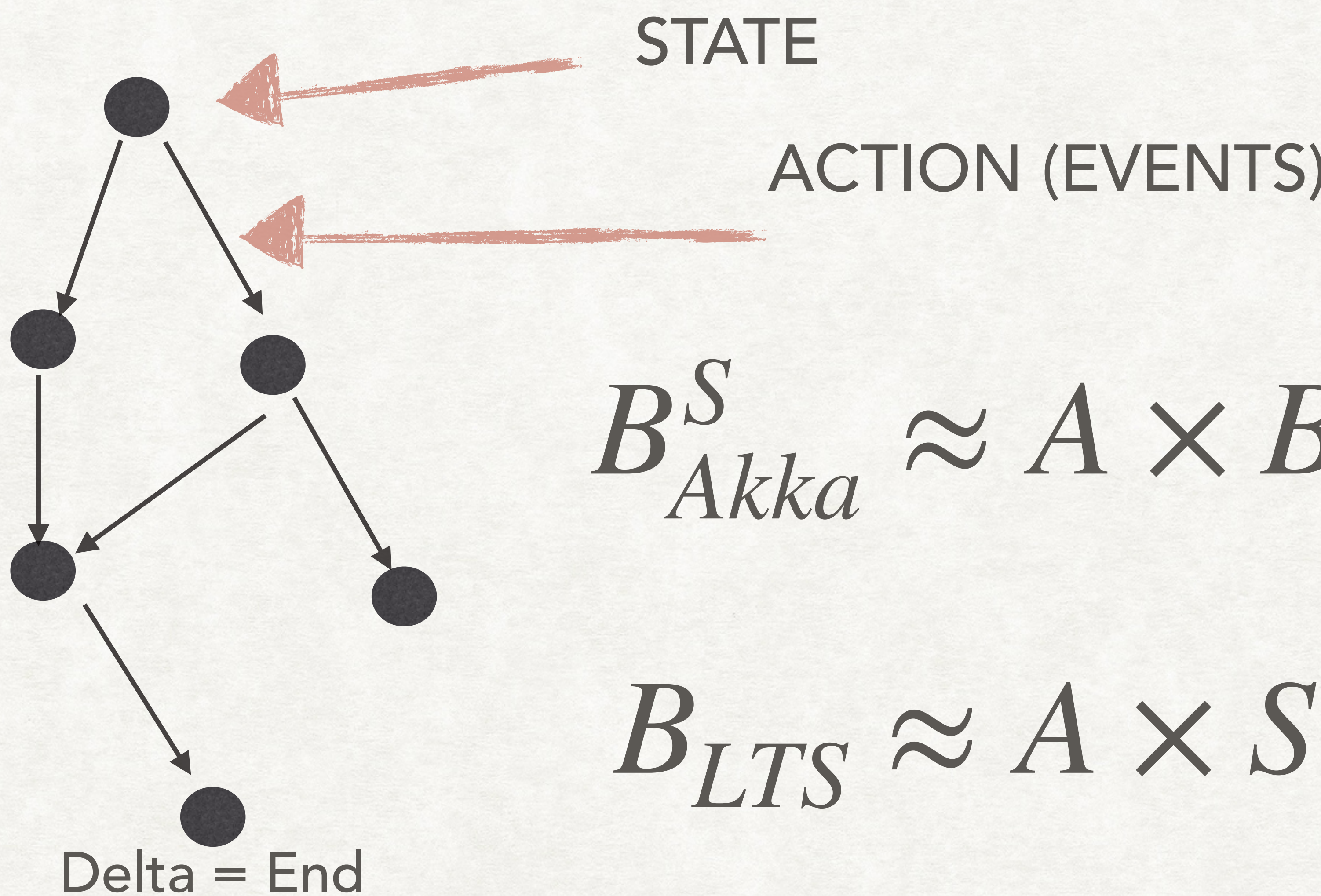
Global Context

$$B^{S}_{Akka} \approx A \times B^{S}_{Akka} \rightarrow B^{S}_{Akka}$$

```
val  behavior(state) = Behaviors.onMessage {
  case  message  =>  behavior(change(state))
}
```

BEHAVIOR DESCRIPTION

// WHAT IS A BEHAVIOR(?: AKKA)//



$$B^S_{Akka} \approx A \times B^S_{Akka} \to B^S_{Akka}$$

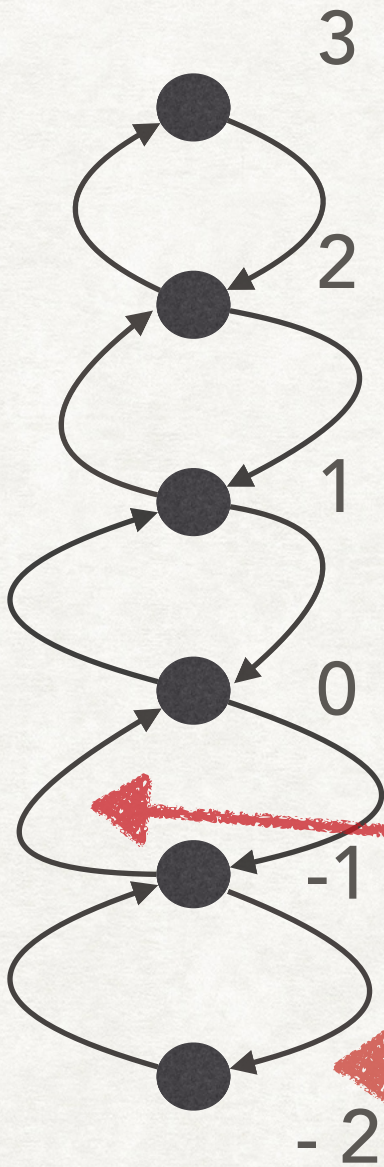
$$B_{LTS} \approx A \times S \to S$$

Classical OOP / FP difference

$$B^S_{Akka} \approx B_{LTS} \times S$$

BEHAVIOUR OPERATIONS

MAKE BIGGER BEHAVIOR BY COMPOSING RECURSIVE EQUATIONS



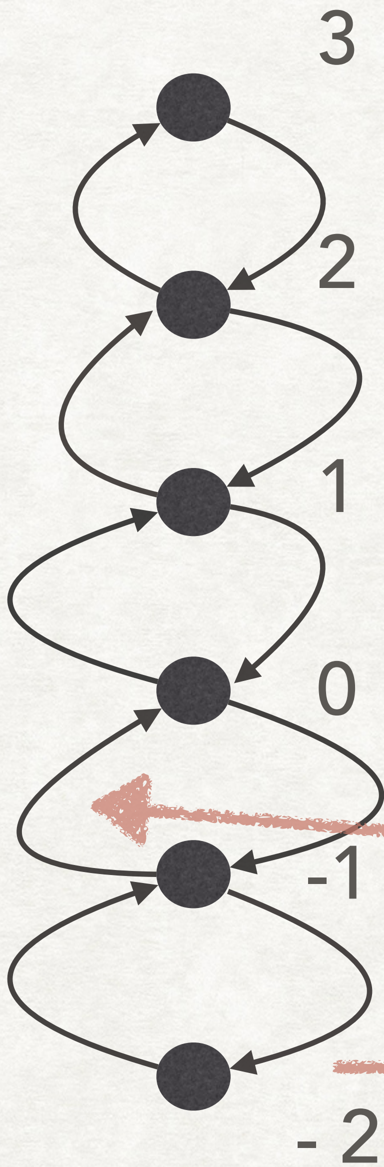
$$B = \begin{array}{l} Left\,.\,(\_ + 1) \times B\ + \\ Right\,.\,(\_ - 1) \times B \end{array}$$

$Left\,.\,(\_ + 1)$

$Right\,.\,(\_ - 1)$

BEHAVIOUR OPERATIONS

SEQUENTIAL COMPOSITION



$$B = \begin{array}{l} Left\,.\,(\_ + 1) \times B\ + \\ Right\,.\,(\_ - 1) \times B \end{array}$$

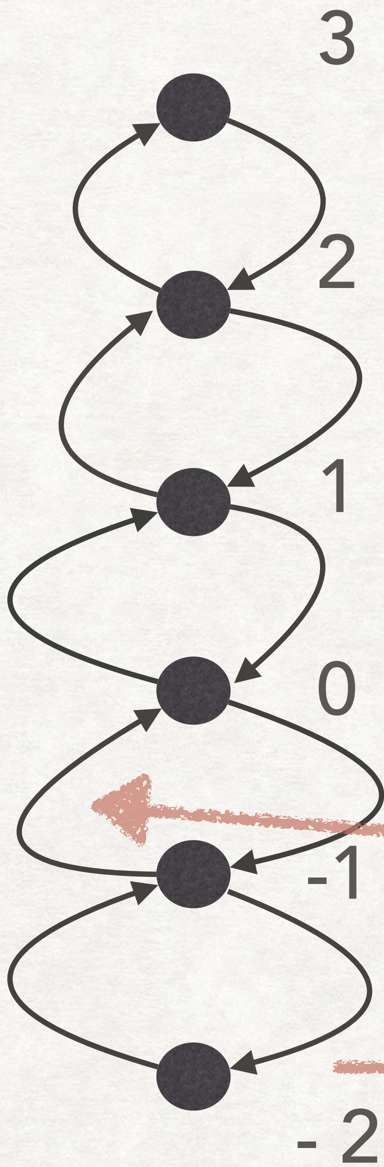
sequential composition (and then)

$Left\,.\,(\_ + 1)$

$Right\,.\,(\_ - 1)$

BEHAVIOUR OPERATIONS

CHOICE COMPOSITION



$$B = \begin{array}{l} Left\,.\,(\_ + 1) \times B \; + \\ Right\,.\,(\_ - 1) \times B \end{array}$$

sequential composition (and then)

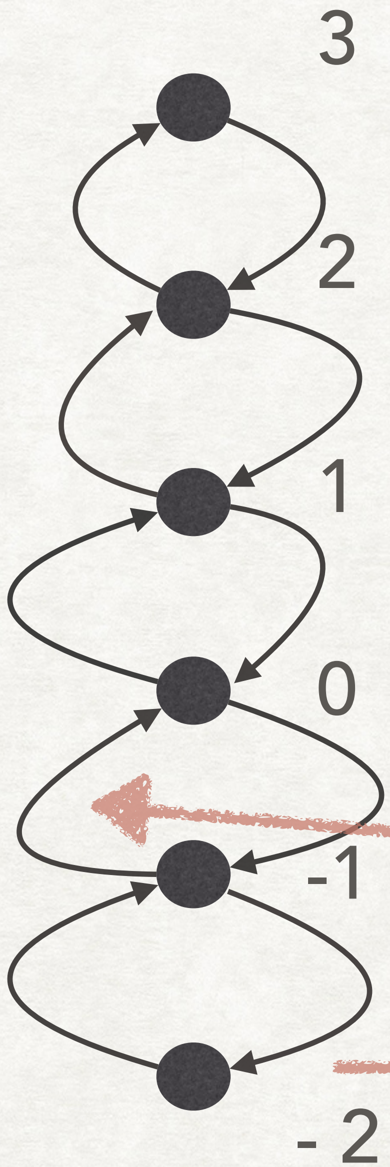
choice composition (or)

$$Left\,.\,(\_ + 1)$$

$$Right\,.\,(\_ - 1)$$

BEHAVIOUR OPERATIONS

RECURSIVE EQUATION



$$B = \begin{array}{l} Left\,.\,(\_ + 1) \times B \; + \\ Right\,.\,(\_ - 1) \times B \end{array}$$

sequential composition (and then)

choice composition (or)

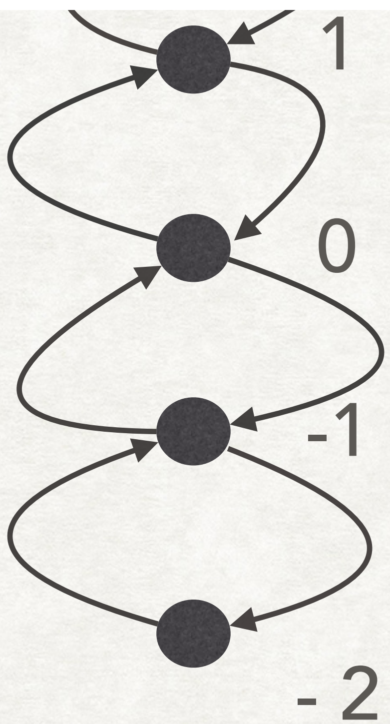
$$Left\,.\,(\_ + 1)$$

$$Right\,.\,(\_ - 1)$$

BEHAVIOUR OPERATIONS

SCALA (?) – SURE !

```scala
lazy val behavior: LTSBehavior[Direction,Int] =
     Once[Direction,Int](Left)( _ + 1) * behavior +
     Once[Direction,Int](Right)(_ - 1) * behavior
```

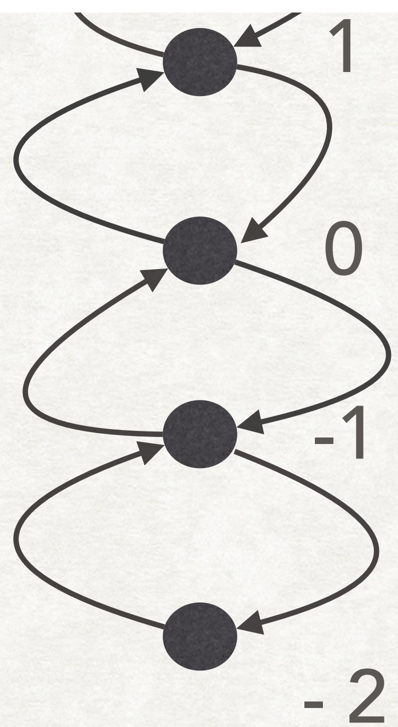



```scala
val actor = ActorSystem(behavior.toPlainBehaviour(0)
                                                  ,"counter")
actor ! Left
```

BEHAVIOUR OPERATIONS

SCALA (?) — SURE !

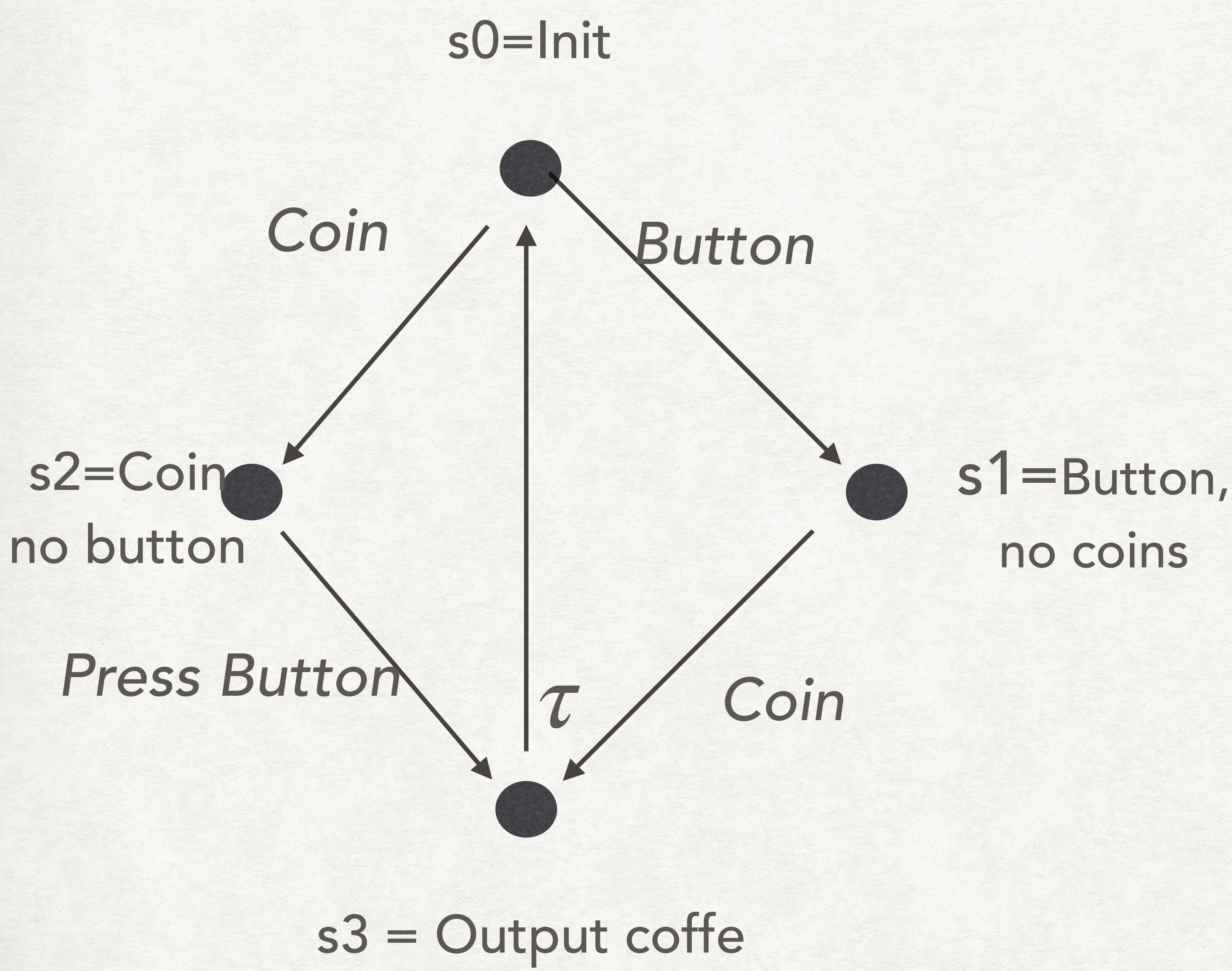
```
lazy val behavior: LTSBehavior[Direction,Int] =
     Once[Direction,Int](Left)( _ + 1) * behavior +
     Once[Direction,Int](Right)(_ - 1) * behavior
```




```
val actor = ActorSystem(behavior.toPlainBehaviour(0)
                                                   ,"counter")
actor ! Left
```

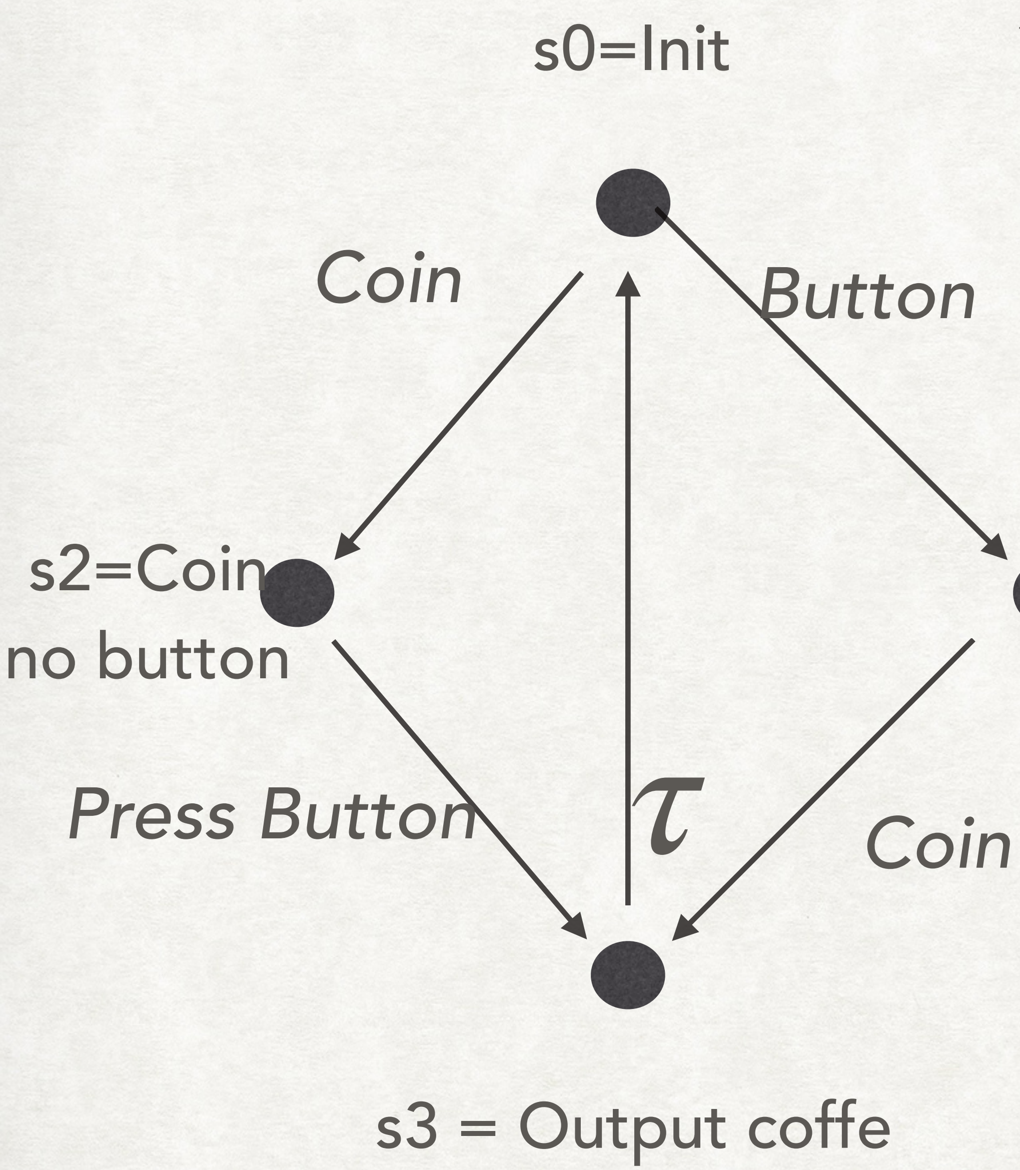
EXAMPLE

COFFE MACHINE - MVP ITERATION



EXAMPLE

COFFE MACHINE - MVP ITERATION



$$B = Button \times Coin \times Output \times B\ +$$

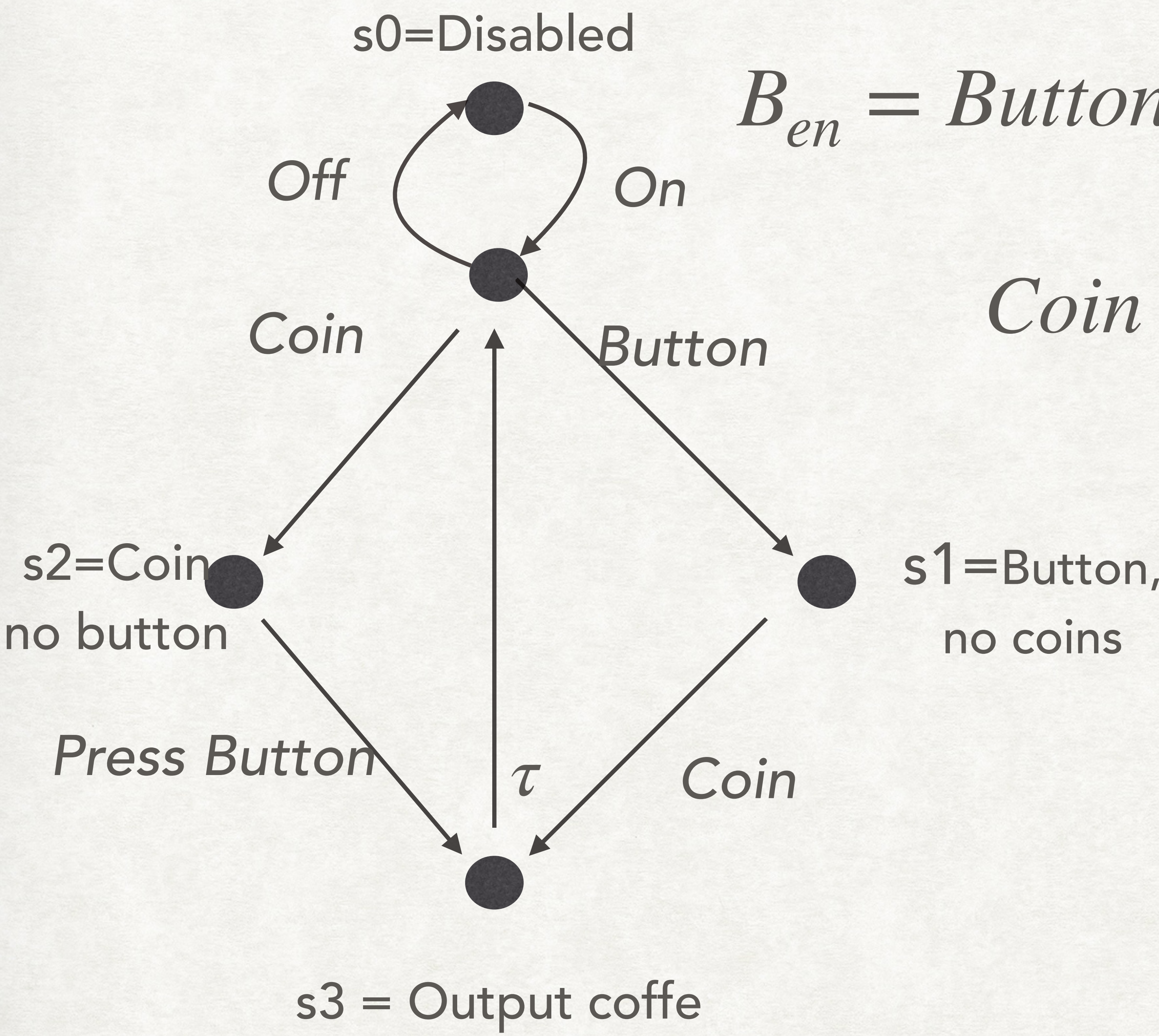
$$Coin \times Button \times Output \times B$$

```
case class State(coin:Boolean,
  button: Boolean,
  makeCoffe: ()=>() )
```

```
lazy val behavior: LTSBehavior[In,State] =
  Once[In,State](Coin)(_.copy(coin=true))*
  Once[In,State](Button)(_.copy(button=true))*
  StateFun[In,State]{ s => s.makeCoffe(); s.copy(coin = false, button = false)}*
    behavior +
  Once[In,State](Coin)(_.copy(coin=true))*
  Once[In,State](Button)(_.copy(button=true))*
  StateFun[In,State]{ s => s.makeCoffe(); s.copy(coin = false, button = false)}*
  behavior
```

EXAMPLE

COFFE MACHINE - ADD ON/OFF



$$B_{en} = Button \times Coin \times Output \times B_{en} +$$

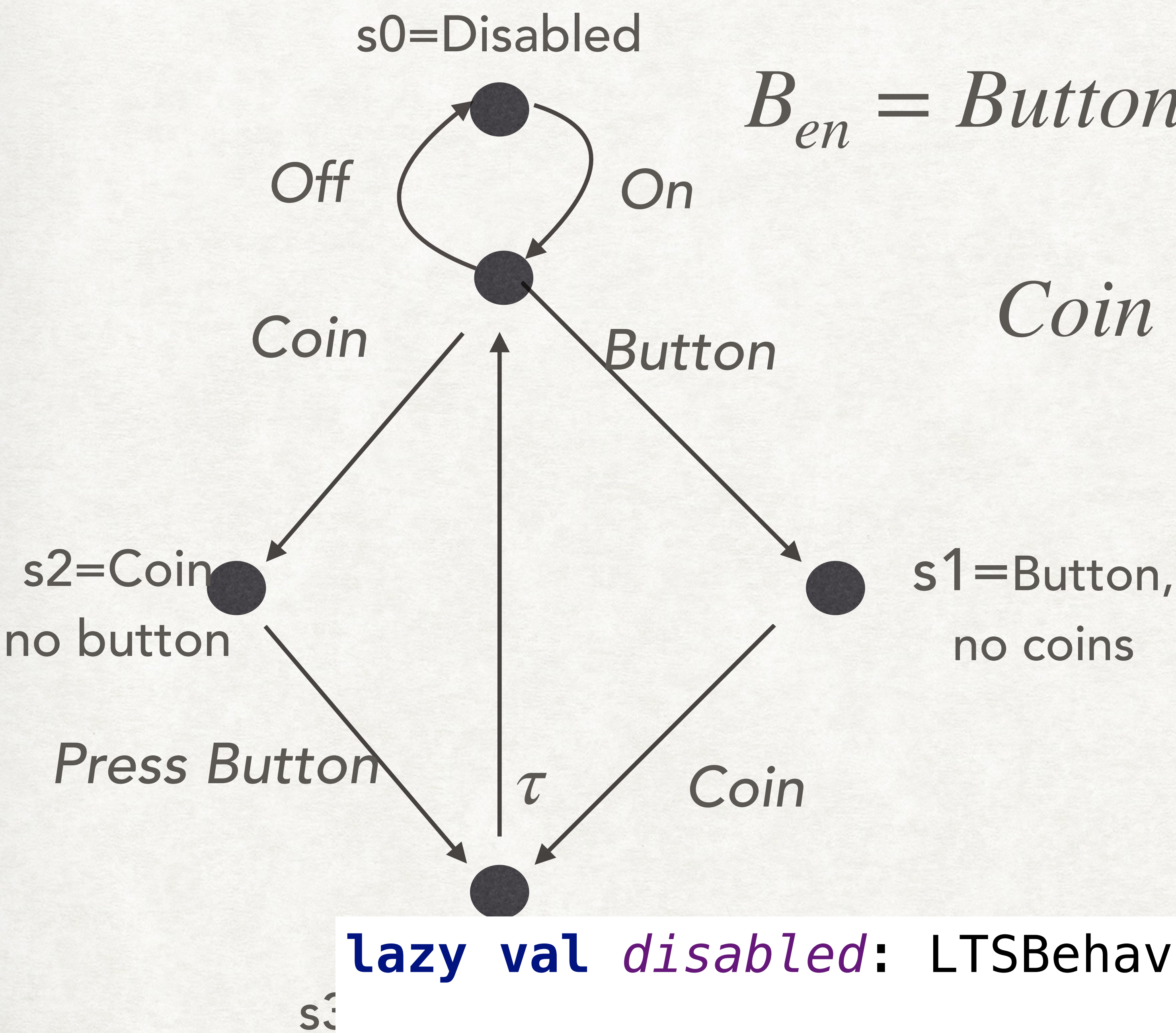
$$Coin \times Button \times Output \times B_{en} +$$

$$Off \times B_{dis}$$

$$B_{dis} = On \times B_{en}$$

EXAMPLE

COFFE MACHINE - ADD ON/OFF



$$B_{en} = Button \times Coin \times Output \times B_{en} +$$

$$Coin \times Button \times Output \times B_{en} +$$

$$Off \times B_{dis}$$

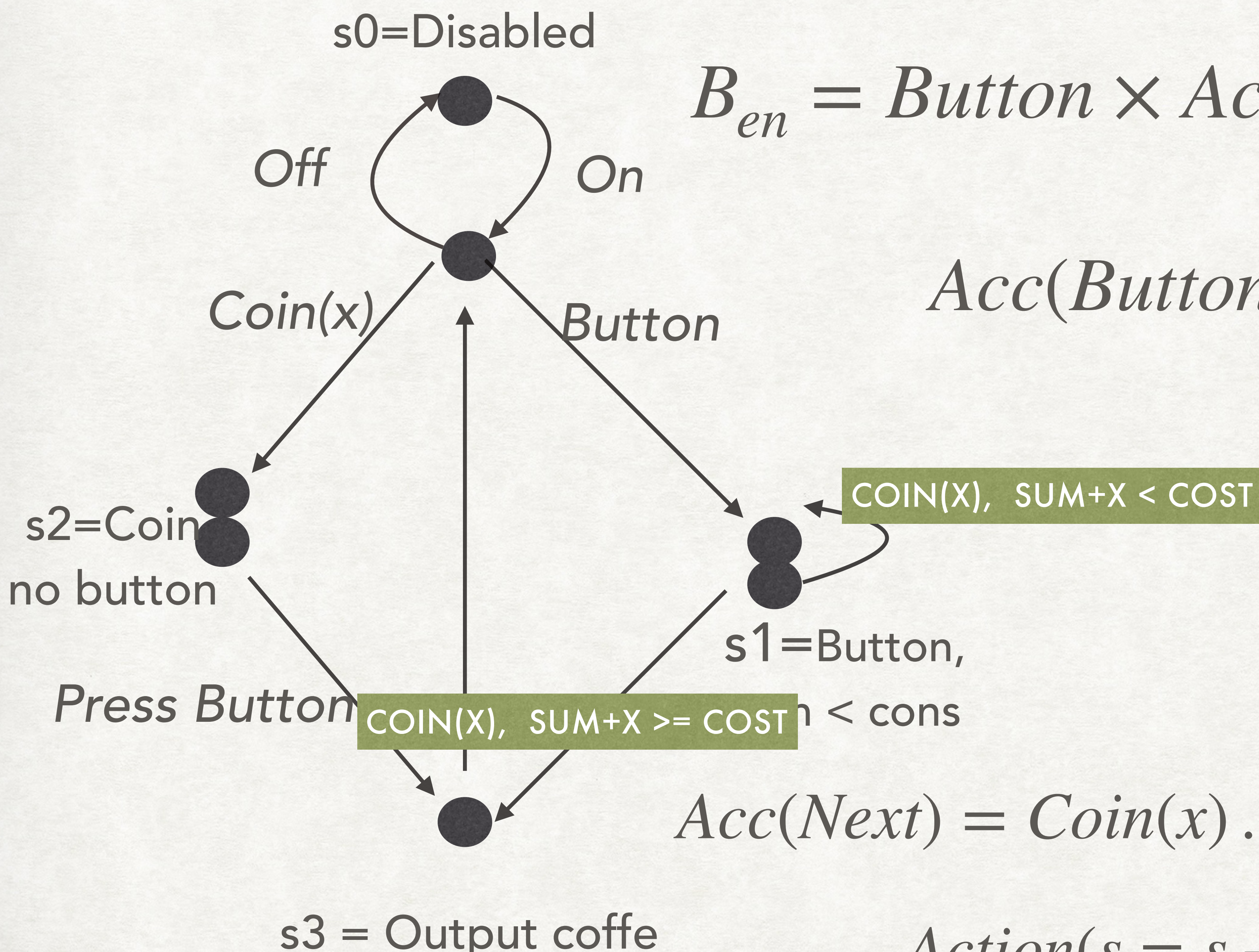
$$B_{dis} = On \times B_{en}$$

```
lazy val disabled: LTSBehavior[In,State] = On * enabled

lazy val enabled: LTSBehavior[In,State] = ….
                                          + Off * disabled
```

EXAMPLE

COFFE MACHINE - ADD ACCUMULATING



$$B_{en} = Button \times Acc(Output \times B_{en}) +$$

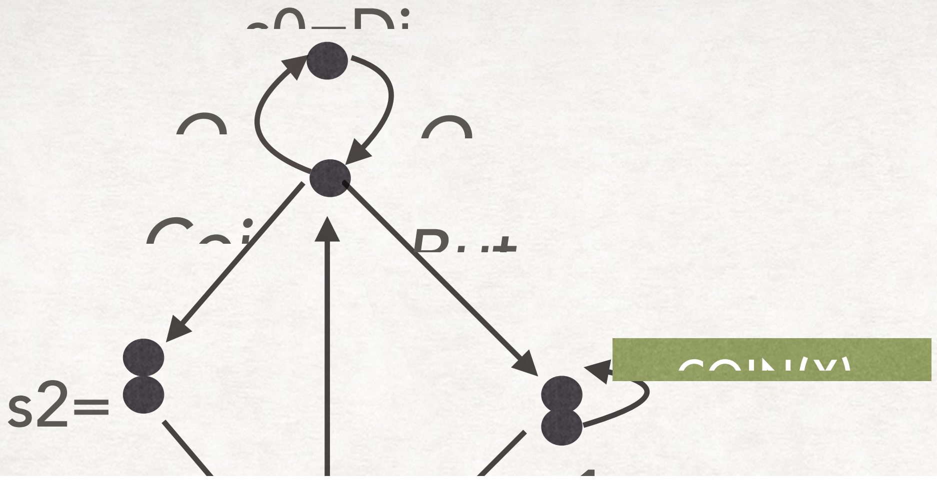
$$Acc(Button \times Output \times B_{en}) +$$

$$Off \times B_{dis}$$

$$B_{dis} = On \times B_{en}$$

$$Acc(Next) = Coin(x)\,.\,s + x < Cost \rightarrow$$

$$Action(s = s + x) \times Acc(Next) \diamond Next$$



EXAMPLE

COFFE MACHINE - SCALA

```scala
lazy val disabled: LTSBehavior[In,State] = On * enabled

lazy val enabled: LTSBehavior[In,State] =
  Acc(
  ConstantInput[In,State](Button)(_.copy(button=true))*
  StateFun[In,State]{ s => s.makeCoffe();
                          s.copy(coins = s.coins - Const, button = false)}*
    enabled
  ) +
  ConstantInput[In,State](Button)(_.copy(button=true))*
  Acc(
    StateFun[In,State]{ s => s.makeCoffe();
                         s.copy(coins = s.coins - Const, button = false)}*
    enabled
  ) +
   Off * disabled


def Acc(next:LTSBehavior[In,State]):LTSBehavior[In,State] =
  Condition[Coin,State]((c,s) => s.coins + c.value < Cost)*Acc(next).upcast[In] +
  Condition[Coin,State]((c,s) => s.coins + c.value >= Cost)*next
```

BEHAVIOUR SPECIFICATIONS

One Actor:

$$a\,.f$$

$$A + B$$

$$A \times B$$

$$\varphi \to A \diamond B$$

Set of Actors:

$$A\,||\,B$$

parallel composition of A,B (start to work in ||)

BEHAVIOUR LOGIC

Henessy-Milner modal logic with recursion =

first-order logic +

$[\mu]P$ P will necessary hold after event

$< \mu > P$ P is possible after event

Words to google: $\mu - calculus$

Behavior Algebra Insertion Modelling

http://garuda.ai

BEHAVIOUR LOGIC

Henessy-Milner modal logic with recursion =

first-order logic + $[\mu]P$ + $< \mu > P$

We can automatically check feasibility of properties

http://garuda.ai (demo)

Non-technical problems:

- verification != business value
- it is possible to kill the project with verification

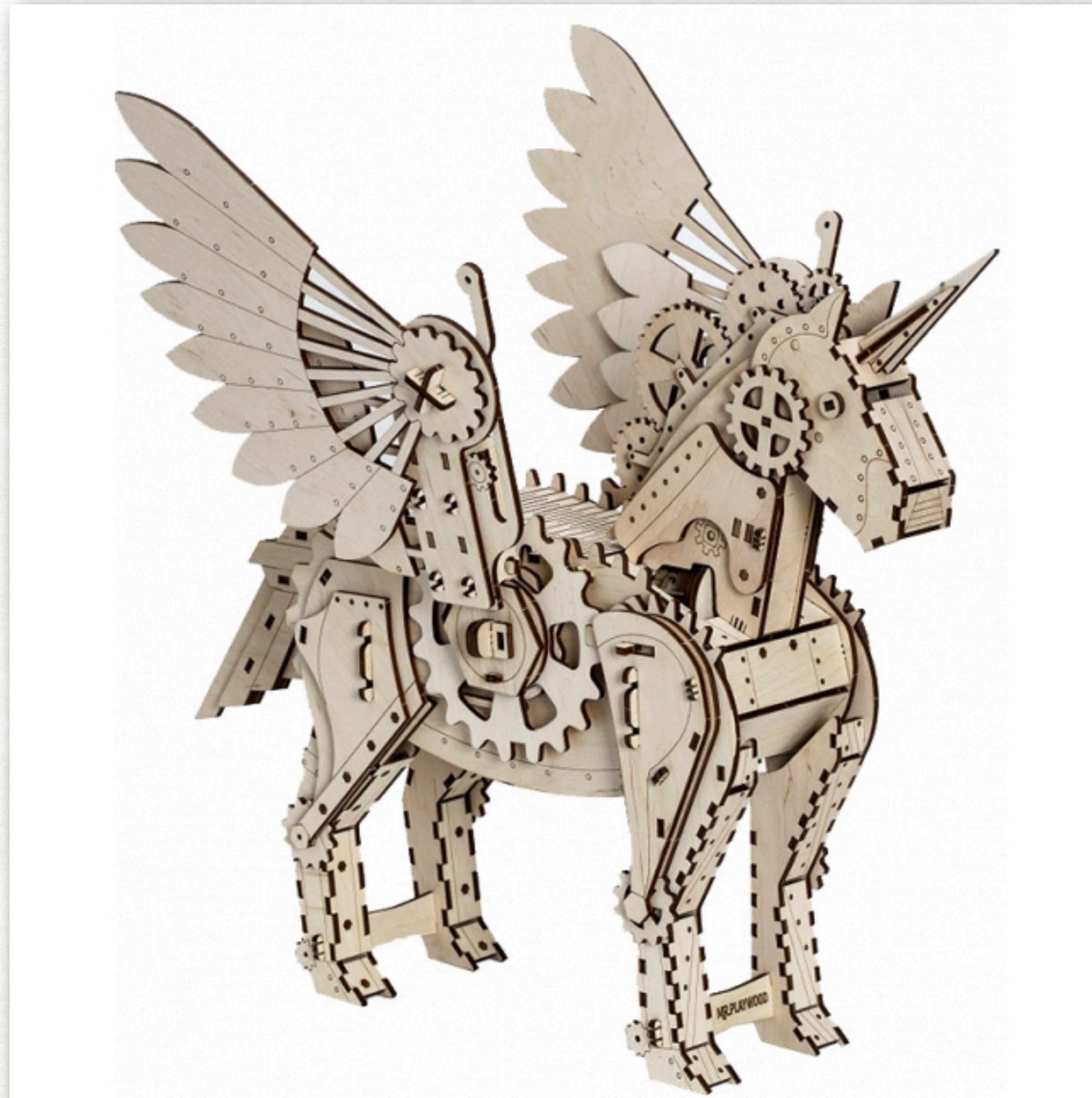
BEHAVIOUR LOGIC

Non-technical problems:

verification !=
business value

carelessly verification
can kill the project

ping me, if anybody
still interested ;)



BEHAVIOR ALGEBRA: CONCLUSION

Consider using Behavior Algebra interpreter

onTop/instead
- Actors,
- State-Machines
- Streams

when state graph is not trivial

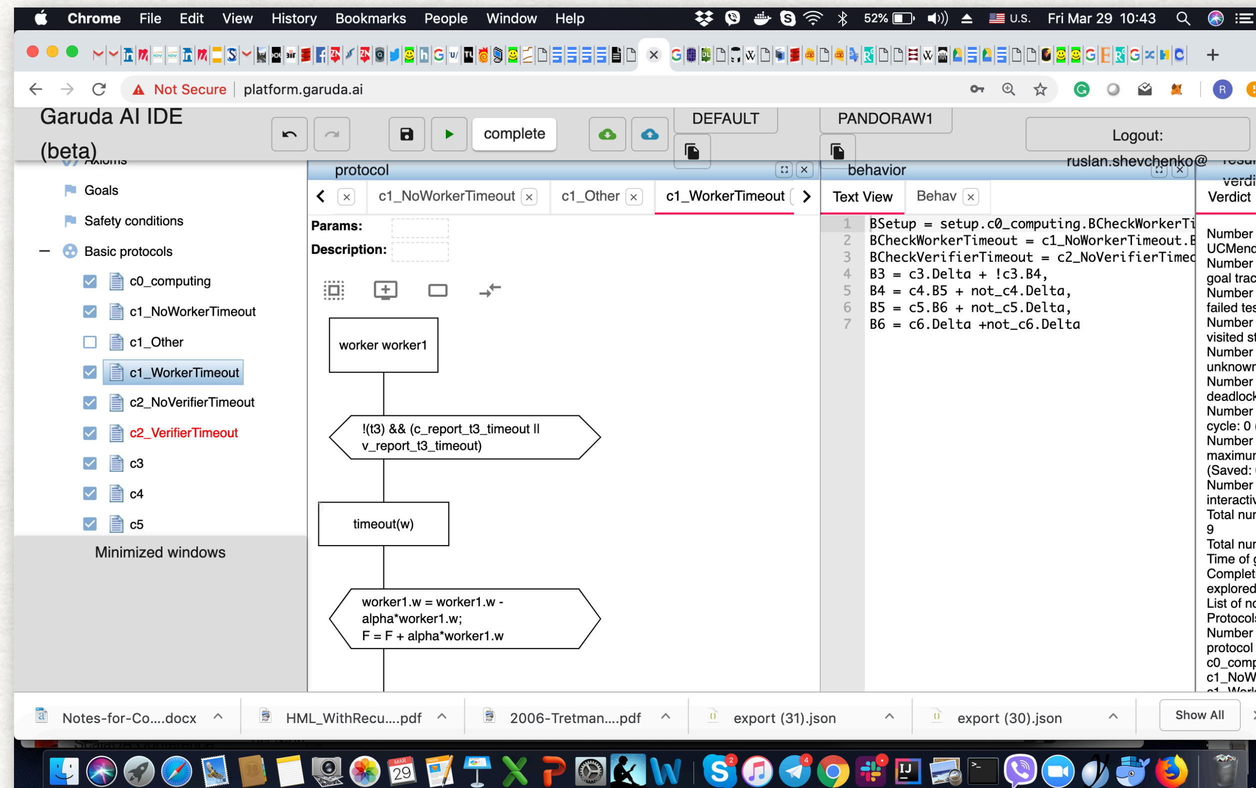
-

It is possible to analyze properties of you system
completeness, liveness, safety ..
in automated way

// this will be mainstream during next 10-100 years

Questions (?)

http://garuda.ai



twitter: @rssh1

e-mail: ruslan@shevchenko.kiev.ua

DONEC QUIS NUNC

EXAMPLE

COFFE MACHINE