Mixing Machine Learning into Logical Search

ICTERI 2025

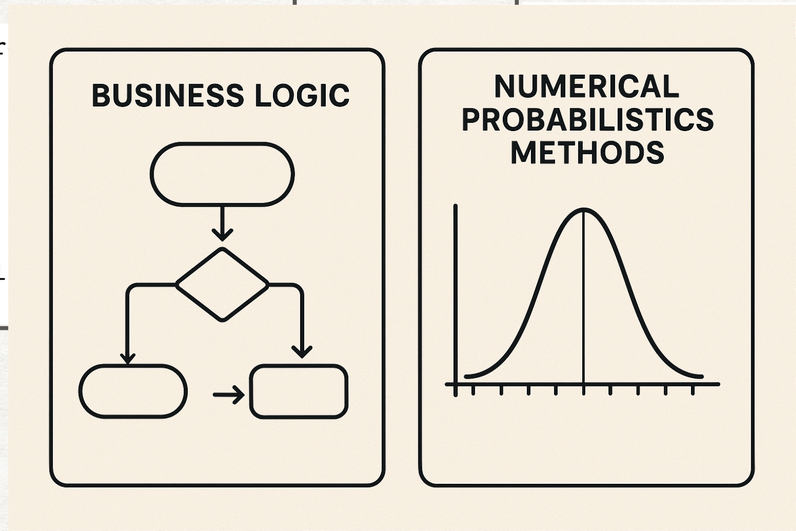
- Ruslan Shevchenko [Institute of Sofware Systems, Ukraine]
- Anatoly Doroshenko [Institute of Sofware Systems, Ukraine]
- Olena Yatsenko. [Institute of Sofware Systems, Ukraine]
- Alexander Newish. [Lantr.io, France]

PROBLEM STATEMENT

BRIDGING MACHINE LEARNING INTO DECLARATIVE PROGRAMMING

Declarative programming

```
def queens[M[_]: CpsLogicMonad](n: Int, pr
          M[Seq[Pos]] = reify[M] {
  if (prefix.length >= n) then prefix
  else
    val nextPos = (1 to n).
        map(Pos(prefix.length + 1, _)).
        filter(isFree(_, prefix))
    reflect(queens(n, prefix :+ reflect(al
}
```



Imperative ML infrastructure

```
ge(num_epochs):
.abels in train_loader:
l pass
= model(images)
:iterion(outputs, labels)

rd and optimize
:.zero_grad()
ward()
:.step()

 [{epoch+1}/{num_epochs}],
                Loss: {loss.item():.4f}')
```

- Hight-level abstractions.
- Immutable data
- Domain-specific DSL

Backend programming, HA Systems

- All transformed to numbers
- Mutable data

Typical ML Stack

PROBLEM STATEMENT

BRIDGING MACHINE LEARNING INTO DECLARATIVE PROGRAMMING

The world if …

Let’s do some steps in this direction.

PROBLEM STATEMENT

BRIDGING MACHINE LEARNING INTO DECLARATIVE PROGRAMMING

The world if …

1. Introduction into
   - Functional way
   - Monadic interfaces
   - Logic monad
2. Construct monads for interfacing with ML
   - Sorted/Orderd Monads
3. Examples in applications.

Let’s do some steps in this direction.

Transform and combinatorial optimisation problem into ML problem

INTRODUCTION INTO THE MONAD PATTERN

COMPLEX DEFINITION FROM CATEGORY THEORY (JUST FOR REFERENCE)

$T : End(C)$ - Endofunctor on C, $(f : X \to Y) \to (f' : F[X] \to F[Y])$; map

$\eta : 1_C \to T$ - unit (pure, return)

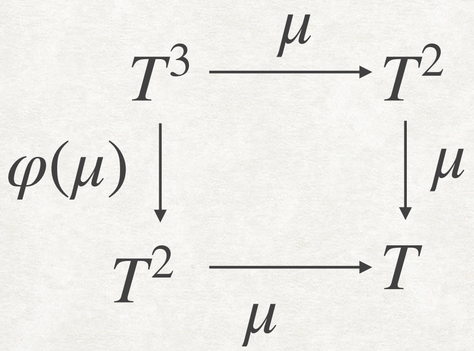
$\mu : T^2 \to T$ - join (flatten, join)

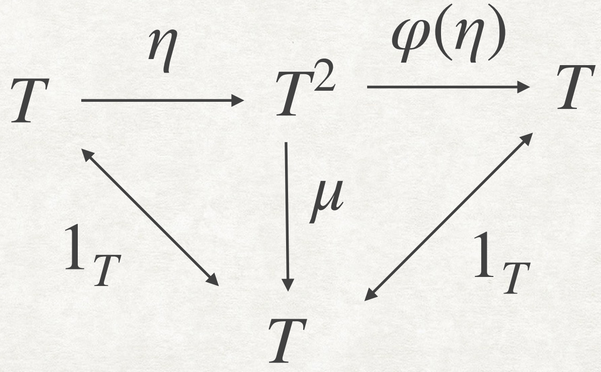
$\varphi : T \to T, \varphi(f) = \mu \circ T(f)$ - flatMap, bind

$\mu \circ \mu = \mu \circ \varphi(\mu)$

$\mu \circ \eta = 1_T$

$\mu \circ \varphi(\eta) = 1_T$





INTRODUCTION INTO MONADIC PATTERN

MORE SIMPLE IN PROGRAMMING: MAP VALUES TO EFFECT+VALUES

```
trait Monad[F[_]]:

  def pure[A](a:A): F[A]

  def flatten[A](t:F[F[A]]): F[A] =
     t.flatMap(identity)

  def flatMap[A,B](fa:F[A])(f: A=>F[B]): F[B]
```

η

μ

φ

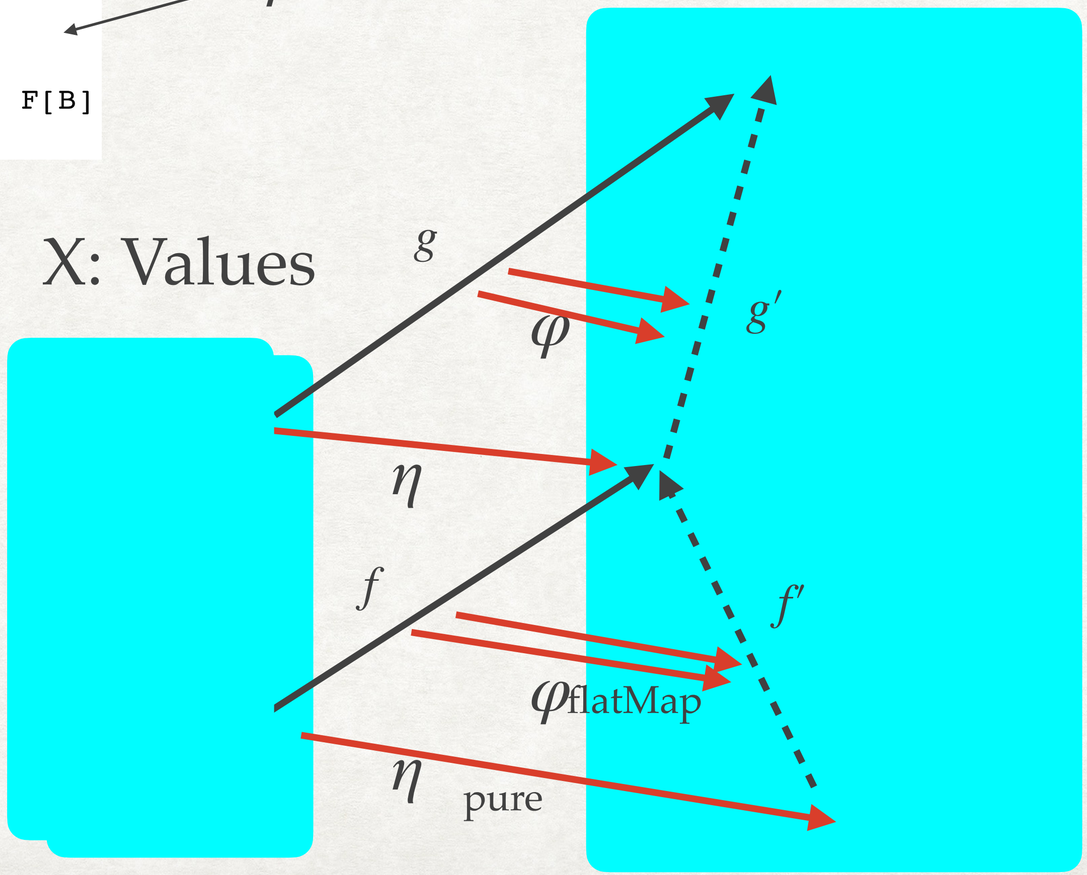
$f : X \to F[X]$

Program

φ flatMap: - compose programs

η pure: - trivial program which return result

F[X]: Values + Effects.



Allows to work with something wich unrepresentable as value (effect)

EXAMPLE OF A SIMPLE MONAD

ABSENCE OF VALUE

```
enum Option[+A]:
  case Some[A](value: A)
  case None
```

A - type parameter

Two possible cases: we have value or not

```
def none[A]: Option[A] = Option.None
```

EXAMPLE OF A SIMPLE MONAD

ABSENCE OF VALUE

```
enum Option[A]:
  case Some[A](value: A)
  case None
```

```
object OptionCpsMonad extends CpsMonad[Option]:

  def pure[A](a: A): Option[A] = Option.Some(a)

  def flatMap(fa:Option[A])(f: A => Option[A]) =
    fa match
      case Option.Some(v) => f(v)
      case Option.None => Option.None
```

EXAMPLE OF A SIMPLE MONAD

ABSENCE OF VALUE

```
enum Option[A]:
  case Some[A](value: A)
  case None
```

```
object OptionCpsMonad extends CpsMonad[Option]:

  def pure[A](a: A): Option[A] = Option.Some(a)

  def flatMap(fa:Option[A])(f: A => Option[A]) =
    fa match
      case Option.Some(v) => f(v)
      case Option.None => Option.None
```

```
def none[A]: Option[A] = Option.None
```

Usage:

```
def safeDivide(x:Int, y:Int): Option[Int] =
  if (y == 0) then none
  else Option.Some(x / y)
```

EXAMPLE OF A SIMPLE MONAD

ABSENCE OF VALUE

```
enum Option[A]:
  case Some[A](value: A)
  case None
```

```
object OptionCpsMonad extends CpsMonad[Option]:

  def pure[A](a: A): Option[A] = Option.Some(a)

  def flatMap(fa:Option[A])(f: A => Option[A]) =
    fa match
      case Option.Some(v) => f(v)
      case Option.None => Option.None
```

```
def none[A]: Option[A] = Option.None
```

Usage:

```
def safeDivide(x:Int, y:Int): Option[Int] =
  if (y == 0) then none
  else Option.Some(x / y)
```

```
def safeDivide(x:Int, y:Int): Option[Int] =
   reify[Option]:
       if (y == 0) then none.reflect
       else x / y
```

EXAMPLE OF A SIMPLE MONAD

ABSENCE OF VALUE

```
enum Option[A]:
  case Some[A](value: A)
  case None
```

```
object OptionCpsMonad extends CpsMonad[Option]:

  def pure[A](a: A): Option[A] = Option.Some(a)

  def flatMap(fa:Option[A])(f: A => Option[A]) =
    fa match
      case Option.Some(v) => f(v)
      case Option.None => Option.None
```

```
def none[A]: Option[A] = Option.None
```

Usage:

```
def safeDivide(x:Int, y:Int): Option[Int] =
  if (y == 0) then none
  else Option.Some(x / y)
```

// monadic style

```
def safeDivide(x:Int, y:Int): Option[Int] =
   reify[Option]:
       if (y == 0) then none.reflect
       else x / y
```

// direct style

// differs only in syntax

OTHER ‘STANDARD’ MONADS

Try — value or exception

```
enum Try[+A]:
  case Success[A](value: A)
  case Fail(ex: Throwable)
```

// Identity — minimal monad which do nothing

```
type Id[A] = A
```

// running computation, which returns the result A

```
trait Future[A]:

  def isFinished: Boolean

  def value: Option[Try[A]]

  ...
```

// lazy computation which yet not started

```
trait Task[A]:

  def unsafeRun(): A

  def cancel: Task[Unit]
```

LOGIC MONAD

WELL-KNOWN (O.KISELEV, 2005)
M[_] REPRESENT BRANCHES OF THE COMPUTATION

```
trait CpsLogicMonad[M[_]] extends CpsTryMonad[M] {

  type Observer[A]


  def empty[A]: M[A] = mzero

  def mplus[A](a: M[A], b: => M[A]): M[A]


  def msplit[A](c: M[A]): M[Option[(Try[A], M[A])]]


  def fsplit[A](c: M[A]): Observer[Option[(Try[A], M[A])]]

  …

}
```

LOGIC MONAD

WELL-KNOWN (O.KISELEV, 2005)
M[_] REPRESENT BRANCHES OF THE COMPUTATION

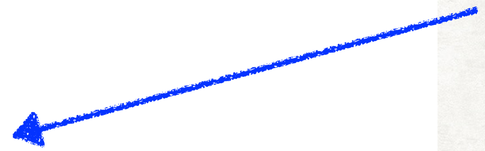
```
trait CpsLogicMonad[M[_]] extends CpsTryMonad[M] {

  type Observer[A]

  def empty[A]: M[A]

  def mplus[A](a: M[A], b: => M[A]): M[A]


  def msplit[A](c: M[A]): M[Option[(Try[A], M[A])]]


  def fsplit[A](c: M[A]): Observer[Option[(Try[A], M[A])]]

  …

}
```

// absence of result. (As Option.None)

LOGIC MONAD

WELL-KNOWN (O.KISELEV, 2005)
M[_] REPRESENT BRANCHES OF THE COMPUTATION

```
trait CpsLogicMonad[M[_]] extends CpsTryMonad[M] {

  type Observer[A]


  def empty[A]: M[A]

  def mplus[A](a: M[A], b: => M[A]): M[A]


  def msplit[A](c: M[A]): M[Option[(Try[A], M[A])]]


  def fsplit[A](c: M[A]): Observer[Option[(Try[A], M[A])]]

  …

}
```



At first, explore ***a*** then ***b***

b is passed by name - ie.

It will not be evaluated before call of ***mplus***

LOGIC MONAD

WELL-KNOWN (O.KISELEV, 2005)
M[_] REPRESENT BRANCHES OF THE COMPUTATION

```
trait CpsLogicMonad[M[_]] extends CpsTryMonad[M] {

  type Observer[A]

  def empty[A]: M[A]

  def mplus[A](a: M[A], b: => M[A]): M[A]


  def msplit[A](c: M[A]): M[Option[(Try[A], M[A])]]


  def fsplit[A](c: M[A]): Observer[Option[(Try[A], M[A])]]

  …

}
```

// monad which used for observer result (Future, etc..)
// Identity in the most simple case.

Observer the first result of the logical search

Note, that ***b*** passed in ***mplus***, will start computation when reader will explore second element of the returning pair

LOGIC MONAD

WELL-KNOWN (O.KISELEV, 2005)
M[_] REPRESENT BRANCHES OF THE COMPUTATION

```
trait CpsLogicMonad[M[_]] extends CpsTryMonad[M] {

  type Observer[A]


  def empty[A]: M[A]

  def mplus[A](a: M[A], b: => M[A]): M[A]


  def msplit[A](c: M[A]): M[Option[(Try[A], M[A])]]


  def fsplit[A](c: M[A]): Observer[Option[(Try[A], M[A])]]

  …

}
```

Split the first result and the rest of the logical search as a result of a logical search

Note, that ***b*** passed in ***mplus***, will start computation when reader will explore second element of the returning pair

LOGICAL MONAD

USAGE

```
def guard[M[_]](p: => Boolean)(using mc: CpsLogicMonadContext[M]): M[Unit] =
  reflect {
    if (p) mc.monad.pure(()) else mc.monad.empty
  }
```

```
extension [M[_], A](ma: M[A])(using m: CpsLogicMonad[M])

 def |+|(mb: => M[A]): M[A] =
    m.mplus(ma, mb)
```

```
def eratosphen(c:Int, knownPrimes: TreeSet[Int]): LogicStream[Int] = reify[LogicStream]{
      guard(
        knownPrimes.takeWhile(x => x*x <= c).forall(x => c % x != 0)
      )
      c
} |+| eratosphen(c+1, knownPrimes + c)
```

BACK TO MACHINE LEARNING

WHAT-S CHANGED IN THE PROGRAMMING WORLD

- Everything scored in numbers.
- Decision are based on scoring.

//So, let’s integrate scoring into the logic search

BACK TO MACHINE LEARNING

ORDER THE RESULT OF THE LOGICAL SEARCH.

```scala
trait CpsOrderedLogicMonad[F[_], R:Ordering] extends CpsLogicMonad[F] {

  type Context <: CpsOrderedLogicMonadContext[F, R]

  /**
   * Set ordering for the branches of the computation,
   * Typpical usage:
   *  ```scala
   *  val alterna = m.orderBy(score, windowLength)
   *  ```
   */
  def order[A](m: F[A])(score: A=>R, windowLength: Int): F[A]

}
```

- monad keep order function a a part of context.
- results are stored in a priority queue with at least window size, if possible.

BACK TO MACHINE LEARNING

ORDER THE LOGICAL SEARCH ITSELF.

```
trait CpsScoredLogicMonad[F[_], R: LinearlyOrderedGroup] extends CpsLogicMonad[F] {

 /**
   * Create a pure value with a score.
   */
  def scoredPure[A](a: A, score: R): F[A]

  /**
   * Create the next branch of the computation with a score.
   */
  def scoredMplus[A](m: F[A], scoreNext: R, next: => F[A]): F[A]

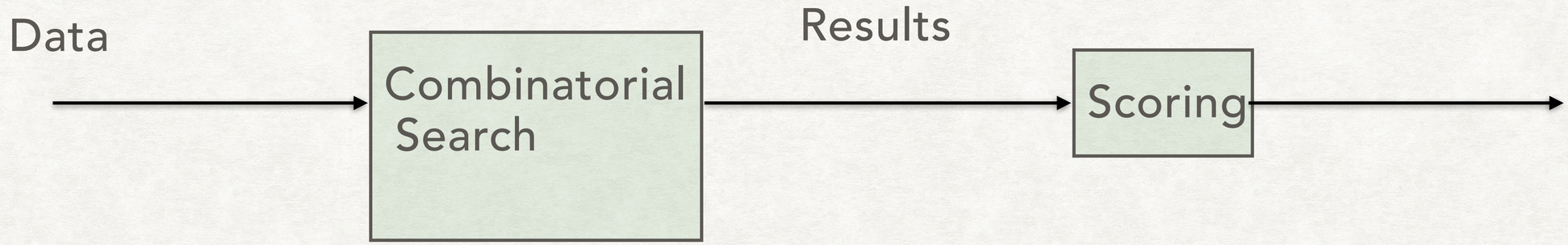
  /**
   * Create the  branch of the computation according to the scopre.
   */
  def multiScore[A](m: Map[R, () => F[A]]): F[A]

}
```

- monad keep each branch with it’s scoring
- branches for further computation (fsplit, msplit) retrieved according to the scoring)

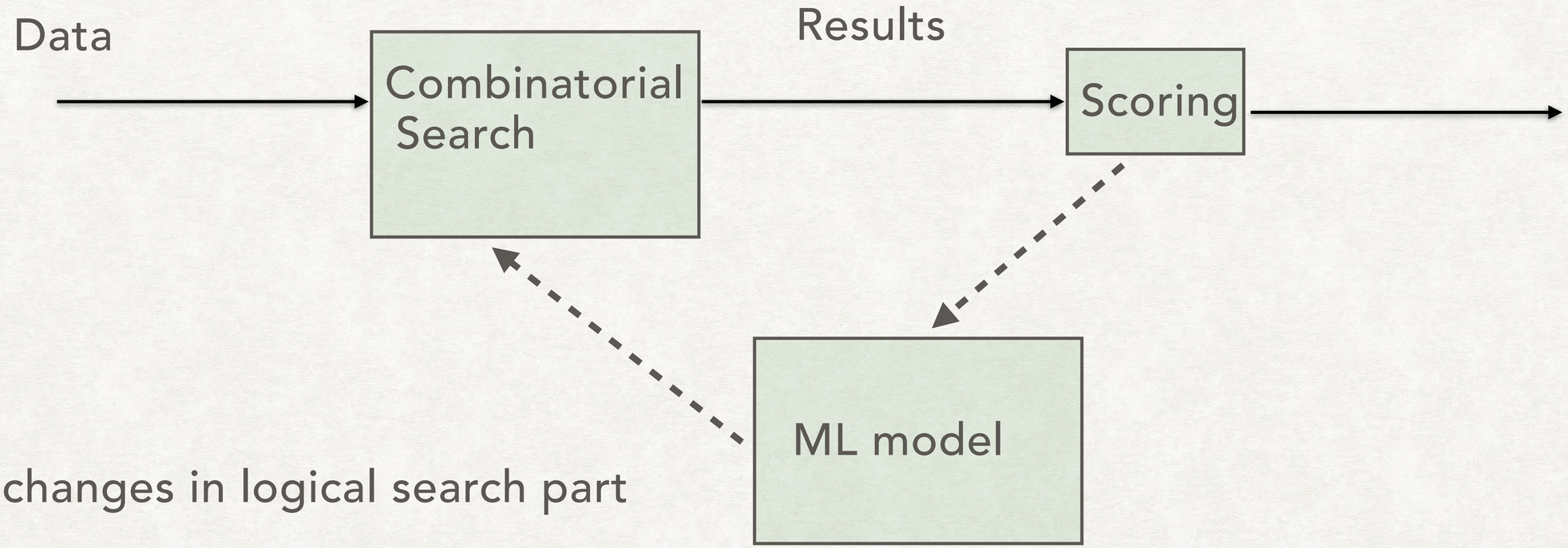
MERGING LOGICAL SEARCH AND MACHINE LEARNING

TURN ANY LOGICAL SEARCH INTO SCORED SEARCH



MERGING LOGICAL SEARCH AND MACHINE LEARNING

TURN ANY LOGICAL SEARCH INTO SCORED SEARCH

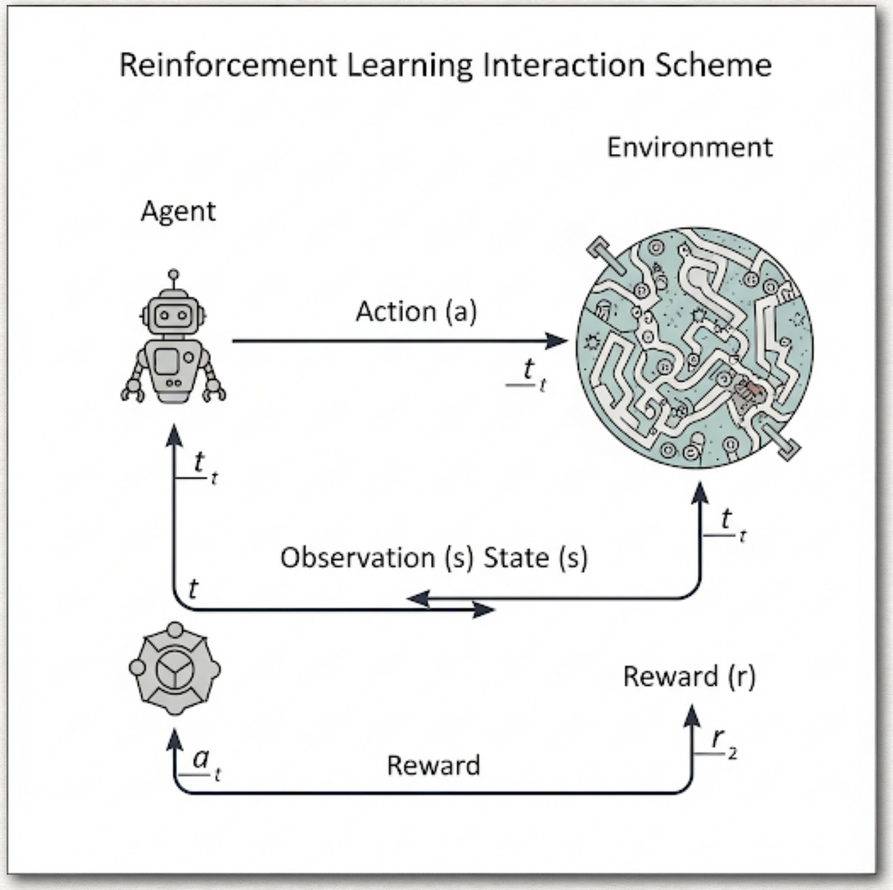


No changes in logical search part

Heuristic in code and data-driven decision can coexist

REINFORCEMENT LEARNING

EXISTS A GENERAL FRAMEWORK FOR THIS.



Agent interact with Environment

Perform actions

Observe state

Environment:

Score agent actions

Change state

ENVIRONMENT

SCALA ENCODING

```
trait RLEnvironment[S, A] {

  def initState: S

  def isFinalState(state: S): Boolean

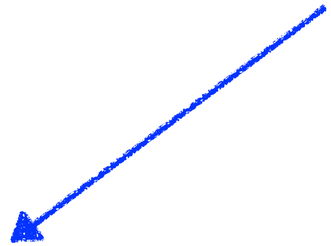
  def applyAction(state: S, action: A): Option[(S, Float)]

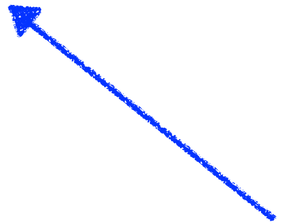
  def possibleActions[F[_]:CpsFloatOrderedLogicMonad](state: S): F[A]

  def isActionPossible(state: S, action: A): Boolean

}
```

Apply action and change state





Generate the set of possible actions

AGENT BEHAVIOUR

SCALA ENCODING

```
trait RLAgentBehavior[F[_] : CpsFloatOrderedLogicMonad, S, A] {

  type AgentState

  def chooseAction(env: RLEnvironment[S, A], envState: S,
                                agentState: AgentState, mode: AgentRunningMode): F[A]

  def performStep(env: RLEnvironment[S,A], envState: S,
                       agentState: AgentState, mode: AgentRunningMode): F[(Option[S], AgentState)]

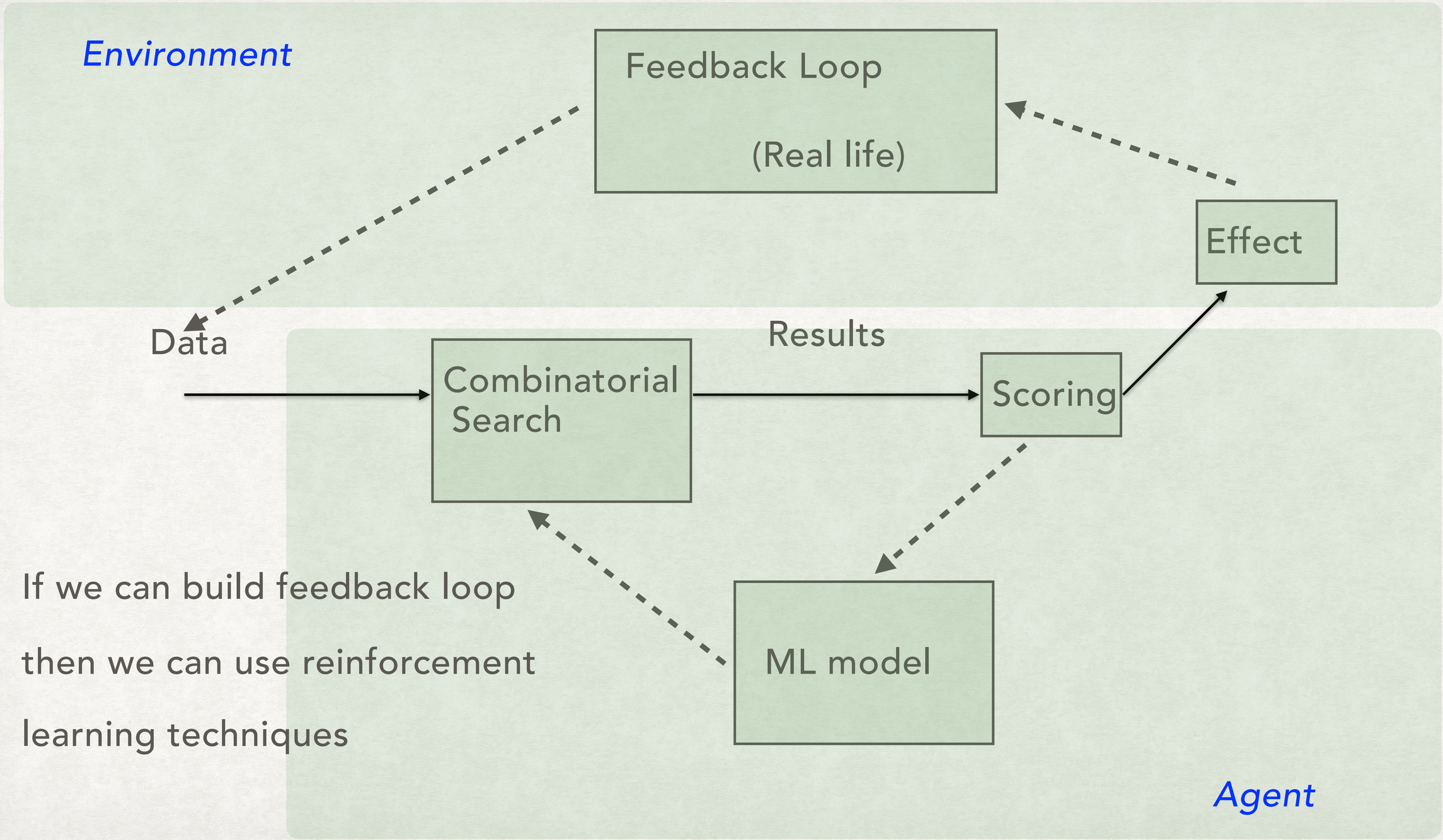
}
```

```
enum AgentRunningMode {
  case Explore, Exploit
}
```

S - observed env state

AgentState - state of Agent.

- agent state can be a ML Model
- encoding is straightforward



If we can build feedback loop

then we can use reinforcement

learning techniques

EXAMPLE: OPTIMISATION INSIDE SCALUS COMPILER

(WORK IN PROGRESS)

- What is Scalus:
  - Plugin to the Scala compiler for Cardano blockchain smart-contracts.
  - Allow to write both on-chain and off-chain parts in Scala
  - Shares Scala data model (can pass structure from off-chain to onchain)
  - Implementation of Plutus VM for JVM. (Allows to debug)

SCALUS “HELLO WORLD"

// on-chain:

```
@Compile
def validator(datum: Data, redeemer: Data, ctxData: Data): Unit = {
    if sha2_256(datum.toByteString) != redeemer.toByteString then
        throw new RuntimeException("Wrong preimage")
}
```

// off-chain:

```
val tx = new Tx()
     .payToContract(scriptAddress(validator),
                    Amount.ada(1000),
                    toPlutusData(sha2_256 b“1239944”))
     …
```

CARDANO SMART-CONTRACT EVALUATION MODEL

- UPLC (onchain)
  - Untyped lambda-calculus.
  - Cek machinę.
  - Budget (fee for evaluation).
    - Size on term (memory)
    - CPU
    - Size on script.

```
step : State → State
step (s ; ρ ▹ ` x)                = s ◃ lookup ρ x
step (s ; ρ ▹ ƛ t)                = s ◃ V-ƛ ρ t
step (s ; ρ ▹ (t · u))            = (s , -· ρ u) ; ρ ▹ t
step (s ; ρ ▹ force t)            = (s , force-) ; ρ ▹ t
step (s ; ρ ▹ delay t)            = s ◃ V-delay ρ t
step (s ; ρ ▹ con (tmCon t c))    = s ◃ V-con t c
step (s ; ρ ▹ constr i [])        = s ◃ V-constr i ε
step (s ; ρ ▹ constr i (x :: xs)) = (s , constr- i ε ρ xs); ρ ▹ x
step (s ; ρ ▹ case t ts)          = (s , case- ρ ts) ; ρ ▹ t
step (s ; ρ ▹ builtin b)          = s ◃ ival b
step (s ; ρ ▹ error)              = ◆
```

OPTIMISATION - MINIMAL SCRIPT FEE

WHAT WE CAN CHANGE:

Representation of classes

```
enum AuctionMessage:

   case AuctionBid(bidder: ByteString,
                   amount: BigInt,
                   time: PosixTime)

   case AuctionClose(time: PosixTime)
```

1. Data:

C 0 | L[B “bidder” | | 100 | | 1756555593]

2. Uplc term:

$$case(0, ('bidder', 100, 1756555593))$$

3. Uplc term over data:

$$case(0, (D('bidder'), D(i100), D(i1756555593))$$

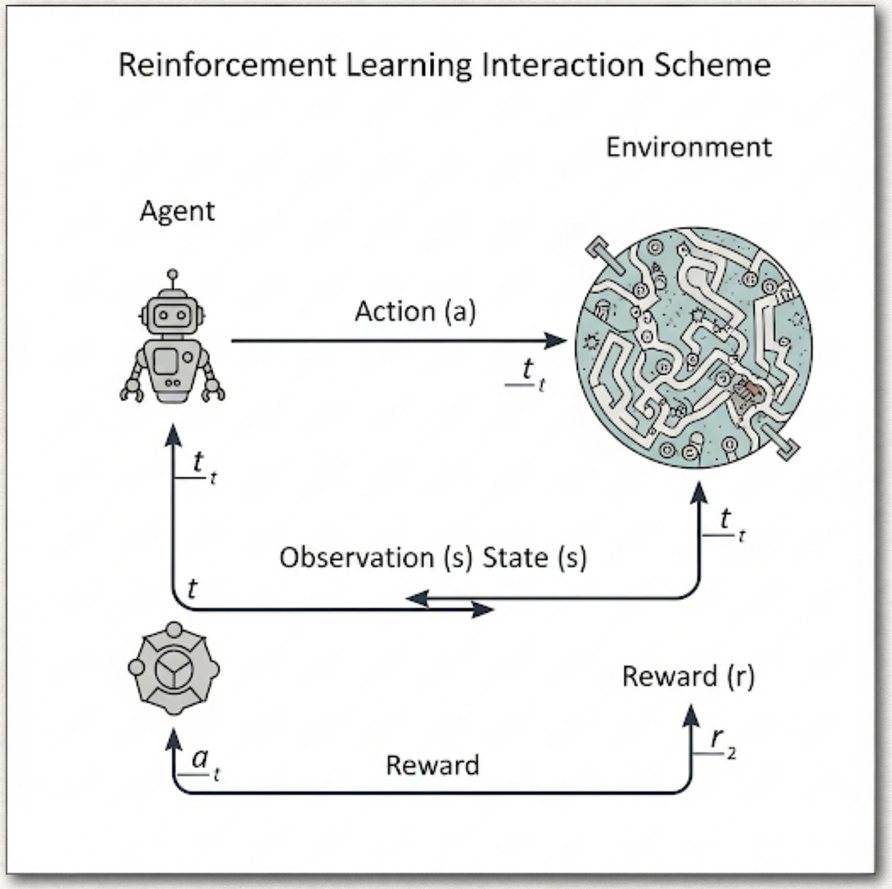
4. Uplc in Scott Encoding:

$$\lambda x_1, x_2, x_2 \,.\, \lambda f_1, f_2 \,.\, f_1(x_1, x_2, x_3)$$

$$'bidder'\ 100\ 1756555593$$

OPTIMISATION IN SCALUS COMPILER

CHOOSE OPTIMAL REPRESENTATION OF VALUE



What is our environment ?

Set of test-cases

Score ?

Cost of test execution

Action ?

change representations for values

Observable State ?

type dependended operations

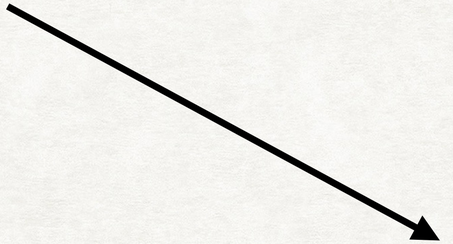
flatten control graph as vector

SCALUS: CHANGES IN THE COMPILER

JUST RUN ALL INSIDE THE SCORED LOGIC MONAD

```
def lower(sir: SIR)(using LoweringContext): LoweredValue = ..

def chooseRepresentation(tp: SIRType,
                         inputs : Seq[LoweredValue ])
                           (using LoweringContext): Representation = …
```



```
def lower(sir: SIR)(using LoweringContext): ScoredLogicStream[LoweredValue] = reify ..

def chooseRepresentation(tp: SIRType,
                          inputs : Seq[LoweredValue ])
                         (using LoweringContext): ScoredLogicStream[Representation[] = reify …
```

```
chooseRepresentation(tp, inputs)
```

```
chooseRepresentation(tp, inputs).reflect
```

FINAL

MIXING LOGICAL SEARCH WITH MACHINE LEARNING

- Technique for injecting ML into the logical search algorithms
- Combinatorial optimisation can be represented as a reinforcement learning.
- ScoredLogic monad on top of well-known Logic monad.

Open source: https://github.com/dotty-cps-async/rl-monad

Work in progress.

Q/A ?

MIXING LOGICAL SEARCH WITH MACHINE LEARNING

- Technique for injecting ML into the logical search algorithms
- Combinatorial optimisation can be represented as a reinforcement learning.
- ScoredLogic monad on top of well-known Logic monad.

Open source: https://github.com/dotty-cps-async/rl-monad

Work in progress.