

Puzzle

by Davor Runje

1 Puzzle

Student picks two numbers between 2 and 100, and gives their sum to professor Sum and their product to professor Product.

Professor Product takes a look at the product and says: I don't know what the numbers are.

Professor Sum replies: I knew you wouldn't know.

Professor Product replies: but now I do.

Professor Sum replies: and now I do too.

What are the numbers?

2 Syntax and semantics

The syntax of our language, as well as its semantics, can be viewed as a subset of multimodal hybrid language with the downbinder operator.

Declaration of functionality shared by all formulas. `TrueAtNode()` represents the usual satisfaction relation over a Kripke model, while `TrueAt()` is a set of all states where formula holds.

```
abstract class Formula
  abstract TrueAtNode(node as Node, model as Kripke) as Boolean
  virtual TrueAt(model as Kripke) as Set of Node
    return {node | node in model.W where TrueAtNode(node, model)}
  operator-() as Formula
    return new Negation(me)
  operator*(rhs as Formula) as Formula
    return new Conjunction(me, rhs)
```

Propositional variables, they are not really needed for the puzzle.

```
class PropVar extends Formula
  override TrueAtNode(node as Node, model as Kripke) as Boolean
    node in model.V(me)
  override TrueAt(model as Kripke) as Set of Node
    return model.V(me)
```

Definition of negation and conjunction. Conjunction is not needed for the puzzle also.

```

class Negation extends Formula
  subject as Formula
  override TrueAtNode(node as Node, model as Kripke) as Boolean
    return not subject.TrueAtNode(node, model)
  override TrueAt(model as Kripke) as Set of Node
    return model.W - subject.TrueAt(model)

class Conjunction extends Formula
  lhs as Formula
  rhs as Formula
  override TrueAtNode(node as Node, model as Kripke) as Boolean
    return lhs.TrueAtNode(node, model) and rhs.TrueAtNode(node, model)
  override TrueAt(model as Kripke) as Set of Node
    return lhs.TrueAt(model) * rhs.TrueAt(model)

```

Professors are represented with modalities. Modal formulas are the usual diamond and box formulas. Box operator models knowledge in the following way: if the formula f holds in all state accessible from a state n with relation $R(\pi)$, then π knows f in n . Diamond formula is not needed for the puzzle, and it is also definable with Box formula and negation.

```

class Modality
  Box(f as Formula) as Formula
    return new BoxFormula(me, f)
  Dia(f as Formula) as Formula
    return new DiaFormula(me, f)
  UniqueEdge() as Formula
    return new UniqueEdgeFormula(me)

class DiaFormula extends Formula
  pi as Modality
  subject as Formula
  override TrueAtNode(node as Node, model as Kripke) as Boolean
    return exists node' in model.R(pi)(node) where subject.TrueAtNode(node',
model)
  override TrueAt(model as Kripke) as Set of Node
    s = subject.TrueAt(model)
    return {node | node in model.W where s * model.R(pi)(node) <> {}}

class BoxFormula extends Formula
  pi as Modality
  subject as Formula
  override TrueAtNode(node as Node, model as Kripke) as Boolean
    return forall node' in model.R(pi)(node) holds subject.TrueAtNode(node',
model)
  override TrueAt(model as Kripke) as Set of Node
    s = subject.TrueAt(model)
    return {node | node in model.W where model.R(pi)(node) subseteq s}

```

And now something completely different. Professor knows pair of numbers iff he consider only one state possible in a state he is in. This is semantically equivalent to having only one edge in a state, connecting the state with itself. A hybrid modal formula

expressing this property is $\neg x. \text{pi.Box } x$ and $\text{pi.Dia } x$, where \neg stands for the downbinder operator.

We choose not to introduce all of the machinery needed for handling formulas with the downbinder operator. Instead, we introduce a new modal operator pi.UniqueEdge which is true in a state n iff there is only one edge from n with relation $R(\text{pi})$ in a model. This is equivalent to formula $\neg x. \text{pi.Box } x$ and $\text{pi.Dia } x$ if the modality pi represents a reflexive relation, which is the case in our puzzle.

```
class UniqueEdgeFormula extends Formula
  pi as Modality
  override TrueAtNode(node as Node, model as Kripke) as Boolean
    return Size(model.R(pi)(node)) = 1
```

Kripke structure is a triple (W, R, V) , where W is a set of possible states, $R(\text{pi})$ is a binary relation over W and V is valuation, that is mapping of propositional variables to set of nodes where the variable holds.

```
type Node = (Integer, Integer)
class Kripke
  W as Set of Node
  R as Map of Modality to (Map of Node to (Set of Node))
  V as Map of PropVar to (Set of Node)
```

3 Solution

The initial Kripke structure has nodes representing all possible ordered pairs of number from 2 to 100. Initially, Professor Sum considers all state with equal sum of numbers possible. Similarly, professor Product considers all states with equal product possible. We don't use valuation at all.

```
abstract class Professor extends Modality
  Knows(f as Formula) as Formula
    return Box(f)
  KnowsNumbers() as Formula
    return UniqueEdge()
  abstract Distinguish(a as Node, b as Node) as Boolean

class ProfessorSum extends Professor
  override Distinguish(a as Node, b as Node) as Boolean
    (x, y) = a
    (x', y') = b
    return x + y <> x' + y'

class ProfessorProduct extends Professor
  override Distinguish(a as Node, b as Node) as Boolean
    (x, y) = a
    (x', y') = b
    return x * y <> x' * y'
```

We define how the Kripke structure changes in a case of a public announcement, represented by appropriate formula.

```

class PuzzleModel extends Kripke
  Announce(f as Formula) as PuzzleModel
    w' = f.TrueAt(me)
    R' = { pi -> {n -> R(pi)(n) * w' | n in R(pi).Indices() * w' } | pi in R}
    V' = { p -> v(p) * w' | p in V }
    return new PuzzleModel(w', R', V')
shared CreatePuzzleModel(w as Set of Node, profs as Set of Professor) as
PuzzleModel
  R = {p -> { n -> {n' | n' in w where not p.Distinguish(n, n')}} | n in w} | p
in profs }
  return new PuzzleModel(w, R, {->})

```

Public announcements, represented by formulas, change the Kripke structure. After all announcements have been processed, we print out all possible states (pairs of numbers) in the resulting Kripke structure.

```

Main()
  n = 100
  Sum = new ProfessorSum()
  Product = new ProfessorProduct()
  W = {(x, y) | x in {2..n}, y in {2..n} where x <= y}
  K0 = CreatePuzzleModel(W, {Sum, Product})
  // Sum announce: i knew that you didn't know numbers (state)
  K2 = K0.Announce(Sum.Knows(-Product.KnowsNumbers()))
  // P announce: now I know numbers
  K3 = K2.Announce(Product.KnowsNumbers())
  // S announce: now I know numbers also
  K4 = K3.Announce(Sum.KnowsNumbers())
  writeLine("Possible pairs of numbers: " + K4.W)

```