Bartosz milewski category theory for programmers pdf

Bartosz milewski category theory for programmers pdf.

Recently I worked with excellent Theory a free book category Bartosz MilewskiÃ ¢ s for Programbers.Ã ¢ The book is available online here and here. I had an incredible moment to read the book and learn to know theory category so I thought IA D publishes solutions to the problems of the online book to make it easier for other people to have such experience. You can find my solutions: Section 1 Def solution Identity (X): Return X P1.2 Implement the composition function in your favorite language. Requires two functions such as topics and returns a function that is their composition. Solution Def Composition (F1, F2): Return Lambda X: F2 (F1 (X)) P1.3 Write a program looking for tests that your composition aspects identity function. Assert Solution Composition (Lambda X: X + 4, Identity) (5) == 9 Assert Composition (Identity, Lambda X: X + 4) (5) == 9 P1.4 Solution The World Wide Web is really A category, if we consider objects from web pages and exists a ¢ arrowÃ ¢ between AE B if there is a way to get to B from clicking on the link P1.5 is a category, with people like Objects and friendships like Morphisms? NO SOLUTION, Because only because A -> B and B -> C does not implicate to -> C P1.6 when is a graph oriented a category? Whenever a solution each node has a border that points to it and for every two nodes a, b such that there is a path from A to B, there is also a connection board to a B. Section 2 Solution Def Memoize (F): call = {} def memoized (x): if x not in calls: calls [x] = f (x) return calls [x] Return memoized test p2.2 to memoize a function from the standard library that you can It normally uses to produce random numbers. It works? Solution This will not operate P2.3 Most random number generators can be initialized with a seed. Implement a function that takes a seed, call the random number generator with that seed, and returns the result. Memoize this function. It works? DEF solution seed_to_random (seed): np.random.seed (seed) np.random.random Return () = memoized_random memoize (seed_to_random) assert np.isclose (memoized_random (0), memoized_random (0)) assert memoized_random (0)! = memoized_random (1) p2.3 Which of these C ++ functions pure? Try to memoize them and observe what happens when it's called them several times: memoized and not. A: factorial function from the example in the text. Factor solution is a pure function B: STD :: Getchar () Getchar solution is not a pure function, given that it is based on Stdin C: "Hello" BOOL F () {STD :: COUT BOOL OPPOSTED TRUEORFALSE = NON TRUEORFALSE ALWAYSTRUE :: BOOL -> BOOL ALWAYSTRUE _ = TRUE ALWAYSFALSE :: BOOL -> BOOL ALWAYSFALSE _ = FALSE SECTION 3 A With a node and free of edge solution Add an Identity arrow. A chart with a node and one (direct) edge (Tip: This edge can be composed with a sÃ ©) Solution Add infinite arrows to represent each number of direct edge applications. A chart with two knots and a single arrow between their solution arrows add identity. A graph with a single knot and 26 arrows marked with the letters of the alphabet: A, B, C Ã ¢ | z. Solution Add an identity arrow, and then add endless arrows, one for each A-Z combination of any length. A set of sets with the inclusion report: A is included B If each element of A is also an element of B. Solution It is a partial order. For any (A, b) is at most one a -> B and if a -> B and b -> then a and b have the same elements and are the same together. Since there could be some (a, b) in which an intersect b is This is not a total of C ++ type order with the following relationship of sottotaling: T1 T2 is a subtype if a pointer on T1 can be passed to a function that expects a pointer to T2 without activating a compilation error . Solution This is a partial order. For any (T1, T2) there is at most a T1 -> T2, and if T1 -> T2 and T2 -> T1, T1 and T2 are the same type. There are types that are not connected by a subdued affair, so this is not a total order. * Solution and Closed output and is Boolean * Identity: the identity is True * Associative: easy to show by enumeration or * closed: the output of either is boolean * Identity: the identity * association is false: easy to show P3.4 Enumeration represents the BOOL Monoid with the operator and the operator as category lists morphisms and their rules of composition. Solution single item in this category is the type BOOL. The morphisms are real and (identity) and false. The composition of these two and is false. P3.5 represents added Module 3 as a category moniglata. Solution single item in this category is the type [int = 0]. The morphisms are * A: 3N ADD (Identity) * B: ADD 1 + 3N * C: ADD 2 + 3N The morphisms in this category are closed in association because © b. B is both c and b. C, c. B is a section 4 SOLUTION OPTIONAL CLASS: def __init __ (self, value): Value = Self._Value DEF IS_VALD (SELF): return self._value is not none of final (car): Assert self.is_valid () Return Self. _Value DEF Compose (F1, F2): DEF Compound (X): F1OUT = F1 (x) Returns F2 (ftoout.get ()) if ftoout.is_valid () Else optional (None) Return Def Def Identity (X): return Optional (x) P4.2 Implement embellished Safe_reciprocal function that returns a valid reciprocal of its argument, if it is non-zero. Def Safe_root Solution (x): optional return (np.sqrt (x)) if x> = 0 stretch Optional (none) def safe_reciprocal (x): optional return (1 / float (x)) if x! = 0 otherwise optional (None) Assert not safe_root (-1) .is_valid () assert np.isclose (Safe_root (4) .get (), 2.0) Assert not safe_reciprocal (0) .is_valid () Assert NP.ISCLUSIONI (SAFE_RECIPROCAL (4). Get (), 0.25) P4.3 Compose Safe_root and Safe_reciprocal to implement Safe_root_reciprocal that calculates SQRT (1 / X) when possible. Solution Safe_root_reciprocal = Compose (Safe_reciprocal, Safe_root) Assert not safe_root_reciprocal (0) .is_valid () Assert not safe_root_reciprocal (-5) .is_valid () Assert np.is_Valid (Safe_root_reciprocal (0.25). Execute (), 2) Section 5 Solution consider Two objects terminals a, B. There is exactly one morphism M1 from a -> B and B's since © terminal and exactly one morphism m2 from B -> a since a terminal. So M1. M2 is a morphism from B -> B and m2. M1 is a morphism from A -> B is © A. Since the terminal, there is exactly one morphism from B -> B, then M1. m2 is the identity. Therefore M1, M2 form an isomorphism between A and B. © Since there are no other morphisms between A and B, M1, M2 is a unique isomorphism. P5.2 What is an product of two objects in a Poset? Tip: Use the universal construction. Solution The product of two objects A, B in a Poset C is the object that is less than both A and B (existing: P: C -> A and Q: C -> b) and for any other object D which is also less than a and B, there is D -> C. This object is not always exists. Sufficiency Let's say that we have an A, B, C. Now consider some object that P2 D: D -> A, Q2: D -> B. Then we have some M: D -> C so P1. M: D -> A and Q1. M: D -> B. Now since there is at most one morphism between any pair of objects in a poset, it is true that P2 = P1. m and q2 = Q1. m, then m facrizis p and q. Necessity if there were some objects D such that D -> A, D -> B but not d -> c, so there is morphism m such that P1 = P2. M Since M must be a morphism from D -> C. Therefore c is not the product of a and b. P5.3 What cos'Ã¨ a two coproduct in a poset? Solution We have just reversing the arrows in P5.2. The coproduct of two objects A, B in a poset is the object C which is greater than both A and B (ie. (Ie. P: A -> C and Q: B -> c) and for any other object D that is also greater than A and B, there is C -> D. This object is not always exists. P5.4 Implement the equivalent of Haskell both as a generic type in your favorite language (other than Haskell). EITHERABTRACT solution class (object): Passage of class Righteither (eitherabstract): DEF __init __ (self, right): self.right = right class LeftEther (eitherabstract): Def __init __ (self, left): self.left = Left # Do something similar this in Python is a recipe for disaster :) Def either_factory (left_type, right_type): def generate (left = none, right = none): assert ((left is none) ^ (a right is none)), if left is not none: assert isinstance (left, left_type) out = leftither (= left left) if it is not none: assert isinstance (right, right_type) out = righteither (Right = Right) Return out of return generate my_left = either_factory (int, str) (left = 5) assert my_left.left == 5 my_right = either_factory (int, str) (right = "5") assert (my_right.right = = "5") Searches: either_factory (int, str) () print ("failed") with exception assertionerror: pass Visit: either_factory (int, str) (left = 5, right = "5") Print ("failed") with exception of AsserretError: Passage P5.5 View that or is a Ã ¢ Int co-co-cover equipped with two injections: INT I (INT N) {RETURN N; } INT J (BOOL B) {RETURN B? 0: 1; } Tip: Define an INT M (or CONST & E) function; What Factorizes I and J. Solution # consider the following injections from INT and BOOL IN INT DEF INT_TO_INT (INT_PARAM): ASSERT ISINSTANCE (INT_PARAM, INT) Returns INT_PARAM DEF BOOL_TO_INT (BOOL_PARAM): ASSERT ISINSTANCE (BOOL_PARAM, BOOL) Return 1 if Bool_Param otherwise 0 # we can define the Next Morphism from Both in Int Def eithera_to_int (either_param): isinstance (either_param, eitherabstract) SE EITHER_PARAM.KIND == "LEFT": OUT = INT_TO_INT (ETHER_PARAM.LEFT) ELIF EITHER_PARAM.KIND == "RIGHT": OUT = BOOL_TO_INT (either_param.right) return out "" "then bool_to_int (x) = either_to_int (either_factory (int, bool) (left = x)) int_to_int (y) = either_to_int (either_factory (int, bool) (right = y)) so EITHER_TO_INT FACTORIZES BOOL_TO_INT AND INT_TO_INT "" "ASSERT EITER_TO_INT (EITHER_FACTORY (INT, BOOL) (LEFT = 5)) == INT_TO_INT (5) ASSERT EITHER_TO_INT (EITHER_FACTORY (INT, BOOL) (RIGHT = TRUE)) == BOOL_TO_INT (TRUE) ASSERT EITHER_TO_INT (EITHER_FACTORY (INT, BOOL) (= des Between false)) == bool_to_int (false) p5.6 Continuing the previous ROblem: How to claim that Int with the two injections I and J can't be a trib to both? Solution Say There is some impossible_m function such that either_factory (int, bool) (left = x) = impossible_m (int_to_int (x)) either_factory (int, bool) (right = y) = impossible_m (bool_to_int (y)) for all int x and y bool. Then it must be the case that: impossible_m (1) = leftither (left = 1) impossible_m (1) = righteither (right = 1) which is not possible, since the output of a function for a given entry topic must be Unique. P5.7 Still continuing: what about these injections? Int (int n) {if (n SupereEther; boolbool x = boolbooltuple (x, x) then we can define more morphisms from Supereither in or this or the first or second element. Section 6 Solution We can define the following two functions, which serve as MAYBETOEHEASHER reversers :: Perhaps a -> O () a MaybetoEether inputmaybe = Inputmaybe case of only to -> to the right one nothing -> Left () Eitomaybe :: o () A -> Perhaps an eithromaybe Inpute Other NÃ © Ther = Inputative case on the right A -> Only left () -> Nothing P6.2 Here "Sa Sum Type defined in Haskell: Form Data = Float | Float Redd Float When we want Define a function as the area that acts on a form, we do the model correspondence on the two manufacturers: Area :: Form -> Floating area (rim R) = PI * R * R Area (RECT DH) = D * H Tool form in C ++ or Java as an interface and creates two classes: circle and rect. Area implement as a virtual function. Solution # User again Python, only for the funny class Abstractshape (Object): Def Area (SELF): Assert NotimpleDereerRor () DEC CIRC (SELF): ASSERT NOTIMPLEATERRORROR () Class Circle (Abstracts HAPE): DEF __IT __ (SÃ ©, radius): self.radius = RADIUS DEF area (car): return self.radius ** 2 * np.pi def circle (car): return 2 * self.radius * np. PI Class RECT (ABSTRACTSHAPE): DEF __Init __ (SÃ ©, height, width): self.height = height self.width = width area DEF (SELF): RETURN SELF.HEIGHT * SELF.WIDTH DECG (SELF): RETURN 2 * self-height + 2 * self-width redd = rect (3, 5) assert redd.circ () == 16 assert redd.Area () == 15 p6.3 Continue with the previous example: we can easily add A new function function that calculates the circumference of a form. We can do it without touching the definition of the form: circ :: shape -> circum float (rim r) = 2.0 * pi * r circia (ret dh) = 2.0 * (d + h) Add circia all your c ++ or java implementation . What parts of the original code did you have to touch? Solution see above. We had to add it to each class, including Abstractshape. Haskell solution We must update the shape definition and add another line to circal and area implementations. For Python we needed to write a new class with a new initializer, inheriting from Return Haskell: data form = floating circle | Float float recto | Floating Square Area :: Form -> Floating Area (Circle R) = PI * R * R Area (Rectal DH) = D * H Area (SQUARE H) = H * H CIRCING :: Form -> Float CIRC (CIRCLE R) = 2.0 * PI * R CIRC (RET DH) = 2.0 * (D + H) CIRC (SQUARE H) = 4.0 * H Python: Class Square (RT): Def __init __ (SELF, length): self.height = length yourself .width = Square length = Square (5) Assert Square.circ () == 20 Assert Square. Area () == 25 Solution A + Equivalent to AA and 2 * A Equivalent to (BOOL, A). We can define the following invertible functions between them. APLUSATOTWOTIESA :: O AA -> (BOOL, A) APLUSATOTWOTISIS ECHEO = CASE BONDED OF LEFT A -> (TRUE, A) RIGHT A -> (False, a) Twotesatoaplusa :: (BOOL, A) -> AA twessisataplusa Twotisima = Case twotimesa di (true, a) -> left a (false, a) -> right a section 7: function solution no, this mapping of morphisms does not retain identity. For some only A, let's see that: (FMAP ID) only A = Nothing ID Just A = Just A P7.2 Tests the laws of the functionist for the reader's player. Tip: It's really simple. Solution We must use the equivalent reasoning to demonstrate that FMAP maintains the identity and preserves the ID ID ID composition (A-> b) = (.) ID (A-> b) = ID (A-> b) FMAP composition ((C-> d). (B-> c)) (A-> b) = (C-> D). (B-> c). (A-> b) = (C-> D). FMAP ((B-> C) (A-> b)) = FMAP (C-> D) (FMAP (B-> C) (A-> b)) P7.3 Implement the player's function in the second preferred language (The first is Haskell, of course). Solution DEF READER_FUNTR_FMAP (F, R_TO_A): Return Lambda R: F (R_TO_A (R)) DEF (R): RETURN 0 DEF R_TO_1 (R): RETURN 1 R_TO_5 = READER_FUNTOR_FMAP (Lambda X: X + 5, R_TO_0) Assert ("R") == 0 ASSERT R_TO_1 ("R") == 1 ASSERT R_TO_5 ("R") == 5 P7.4 Demonstrate the laws of the leads for the list player. Suppose the laws are true for the part of the list of the list you are applying it (in other words, use induction). FMAP FMAP IDENTITY BASE SOLUTION SOLUTION Nil = Nil = FMAP Commposition ID (F. G) Nil = F (Nil) = FMAP) ENT) ENTER ID (CONTRUE ID (CONTROUR (FMAP)) = Contro (ID) (ID) = AD (XT) ID (F. G) Composition (F. G) (F. V. G) x) (F. G) (f. G)) t) // definition of fmap ((f. g) ((fmap f. FMAP f) // = FMAP F (ADDRO (FMAP) (FMAP) // FMAP definition = Fmap f (fmap (counter (fmap) // definition (ft) section 8 (f fmap fmap g.): FUNCTORALITY SENDING TO MEAN AFTERNET, HOME THE FMAP INTO i read Ã¨o: FMAP F fari (a), (c), (c) identbian italie c = peers the f * g composition f * g look like F * g (a) C = fins fins fmap f * f * f fmap fmp = app pairs ab = AB PairbimaP :: (a -> C) -> d) -> Para Ab -> Cats ParairBapapapap -> Cedb Franquable ParairBarap (g Bis) (hb) Pai RFIRST :: (a -> C) -> CB SAIRF AP) = Corsain (ga) B PaairSCON :: B PAIRSCON ::> couple Pairsecond pairing pairsecond f (couple ab) = pair of (fb) the test that these definitions be compatible with the default implementations each time you can be applied. PAIRBIMAP (PAIRBIMAP GAN) (PARR) (HB) // definition of Pairbimap Pairbimap (HB) // 1 Pairfirst (pairfirst (pairfirst (pairfirst (pairfirst (pairfirst / Definition of meek pairnecond cuchi tale tale Paairbimap Gh = pairfirst pairsecond h proof of pairfirst pairfirst g (pairs (ga) b // 1 pairfirst (g) of pairbimap (g id) s // definition of pairbimap that means that Pairfirst G = Pairbimap Pairsecond F (PARSCOND pair) // PainSeCond look ((FB) // Definition of Pairbimap (ID) Pairbimap Pair // 1 Pairbimap Il mean than pairsecond = pairbimap date myidentity a = myidstity dates. (Myidentity a) (myidentity a) (myidentity a) (myidentity): (myidentity a) (Myconst ()) = no desugateDtomaybean (myidentity (myidentitygared.:> Maybe. (Myconst () Maybetodeded () Maybetodugred () May a) = right (myidentity a) we showing you are the inverso of each other uses using equation exqution modernized desguedtomaybe (left ())). maybeToDesugared not Hing = maybeToDesugared Left (MyConst ()) desugaredToMaybe maybeToDesugared Nothing = desugaredToMaybe (Left (MyConst ())) = nothing maybeToDesugared desugaredToMaybe (right (MyIdentity a)) = maybeToDesugared re = Right (MyIdentity a) desugaredToMaybe maybeToDesugared only = desugaredToMaybe Right (myidentity a) = Only a p8.3 Let it will arrange another data structure. Io we chament a prelist perch-© engaged s a foreounding of a list. Esso substitutes the recall with a pair of Type B: Data Pelist to B = nil | Counter a b. If you are recovering our previous definition of a listing Applicant Recursively prelisting to you (Wea ll see how it is made of fixed points). Showing you preelist Â,one an instance of bifunctor. Consent Solution Formace Formare FMAPLOW PHY (A -> (PLECIST CD) -> (PLECIST CD) FM Nil = FG AB = AB = AB = (F BIS) (GB) Diesno Tenamo B (WLOG). Then the fmap for an ã fmap f ot a fmap fcru c = counter (fa) c of (f'to) c f'.expse (f'ho) b this (fa) and functor (f'er) ft. f C = ADDROO. g = nil nil fmap f. g counter a group c = count f. ga c = f fmap count g = fmap f (g fmap count a c) P8.4 Destiny Data defaults in A e B: K2 Cabs = K2 C data = Fst a data snd ab = snd b solution k2: without loss of generality, if we consider constant b, then it becomes k2 const, which is a fst fundor: if we keep a constant, then it becomes fst const, which is a one If we keep a constant, then FST becomes identity, which is a SND worktore: if we keep a constant, then SND becomes identity, which is a funtor if we keep a constant, then it becomes SND CONST, which is a P8.5 subtore Define a bifunctor in a different language Haskell. Implement bimap for a generic couple in that language. Bifunctor solution class (object): DEF apply_bimap (SELF, F, G): FALSE ASSERTION @ClassMethod Def first (CLS, F): Return Lambda Couple: pair.apply_bimap (F, Lambda X: X) @ClassMethod Def second (CLS , g): return lambda couple: pair.apply_bimap (lambda x: x, g) @classmethod def bimap (cls, f, g): lambda return pair: pair.apply_bimap (f, g) equal class (bifunctor): def __init __ (Self, Aval, BVAL): self.aval = aval self.bval = bval def apply_bimap (self, f, g): return coupling (f (self.aval), g (self.bval)) p = pair (5, "4") first_mapped = bifunctor.first (lambda x: x + 1) (p) assertion first_mapped.aval == 6 assertion first_mapped.bval == "4" second_mapped = bifunctor.second (lambda x: x + "1") (P) Second_mapped.aval assertion == 5 Second_mapped.bval assertion == "41" Bimapped = Bifunctor.bimap (Lambda X: X + 1, Lambda S: S + "1") (P) Bimapped Assertion . AVAL solution STD :: MAP should be considered a PROFUNTOR A KEY AND T. You can define as a PROFUNTOR as follows: Get a :: -> maybe b Example Profontor to arrive where Dimap FG Get = LMA P f. RMAP G LMAP F GET = X -> GET (FX) RMAP G GET = X -> FMAP G (x Obtain) SECTION 9: Types of function (no challenges) Solution Nattrans :: Maybe a -> [a] Nattrans (Enough X) = [x] Nattrans Nothing = [] The condition of naturality is GF A | I ± a = i Â ± b in | F F, which translates into FMAP_LIST f. NATTRANS = NATTRANS. FMAP_MAYBE F NO CASE: FMAP_LIST f. NATTRANS NOTHING = F FMAP_LIST [] = [] = NATTRANS NOTHING = NATTRANS. FMAP_MAYBE F NOTHING (JUST X) Case: FMAP_LIST F. NATTRANS (JUST X) = FMAP_LIST F [X] = [F (X)] = NATTRANS (JUST (F x)) = NATTRANS. FMAP_MAYBE F (X ONLY) P10.2 Define at least two different natural transformations between reader () and the funtor list. How many different lists of () are there? NATTRANSRL1 :: ((() -> a) -> [a] nattransrl1 _ = [] nattransrl2 :: (() -> a) -> [a] nattransrl2 g = [g ()] nattransrl3 :: (() -> a) -> [a] nattransrl3 g = fmap g [(), ()] since there are an infinite number of lists of [(), ...] there are an infinite number of these natural transformations. P10.3 Continue the previous exercise, with BOOL reader and maybe. Solution There are three natural transformations from Reader Bool -> Perhaps Nattransrb1 :: (Bool -> a) -> maybe a nattransb1 _ = Nothing nattransrb2 :: (Bool -> a) -> maybe a nattransrb2 g = just (g true) NATTRANSRB3 :: (BOOL -> A) -> Perhaps a nattransb3 g = just (f fake) p10.4 Displays that the horizontal composition satisfies natural transformation the naturality condition (Tip: use the components). ItÃ ¢ s a good exercise in the hunting diagram. Solution that we have the F, G and natural transformations: I ± a :: F A -> F'a Ãžâ²a :: G A -> G'a must show that (G 'F'.) F. (IÂ² a | â ±) a) = (iâ² Ã ¢ | i Â ±) b. (. G f) f (iâ² Ã ¢ | Ãž Â Â ±) b. (G. f) = (Ãžâ²f'b. Gi Â ± b). G. F = f // Definition of the horizontal Ãžâ² "composition. G. F 'f. Ãž Â ± a = // g Ãž Â ± b :: g (f b) -> g (f'b) (g 'f') f. (IÂ² a | i Â Â ±) a = // Ãžâ²f'b :: g (f'a) -> g '(f'a) p10.5 Write a short essay on how you can enjoy writing the diagrams Needs evident to demonstrate the interchange law. Solution if it's the case that: f -Ãžâ² '-> f' f '-i â â ±' -> f '' g -Ãžâ²-> g 'g' -i Â Â ± -> g '' Therefore for the definition of the horizontal ita s simple to see that: fg - (iâ² 'a | iÂ²) -> f'g' - (i â ± 'a | i Â Â Â) -> f''g' 'fg - (IÂ² 'I Â ±') A | (IÂ² I Â ±) -> F'G '' Furthermore, per horizontal: FG - (iâ² 'A | IÂ²) -> F'G' F'G '. - (I Â ± 'A | I Â Â ±) -> F''g' 'FG - (IÂ²' A | IÂ²). (I Â ± 'A | I Â Â ±) -> F''g' '(IÂ²' A | IÂ²). (Â ± 'at | i Â Â ±) and (iâ²'. I Â ± ') A | (. IÂ² I Â ±) The equivalent effect on the FG P10.6 creates a couple of test cases for the opposite condition of naturalization of transformations between different op. HereÃÂ ¢ s a choice: op :: op bool int op = op (x -> X> 0) and F :: :: -> Int fx = reading newtype on ra = op (s (s) = a g f.) = a (s) = a abra - b - to a f) FX - Test 1 OP1 :: SO Bool int (\ x -> (X>)) F1:> Int F1 = 0 Els 0 OPOBOOGAR3 :: on Bool A -> On char an opketboard (SO ATBOLIC) = OP (\ X -> If atbool_) boolh_f folchar_f folchar_F folchar_op1 = control1 = control F1 (OPOBOOCOOPCAR OP1) TEST1A = (UNWRAP_OP1B_F_FROP1) - Test1B .: Test 1P2 = OP (\ x - displayed x) F2 :: Int -> Double F2 x = Sqrt (SQRGENCE -Regral (SAPTROGNEOPTA) (SAPTROMING) (SAPTROGER int string_contra _f op2 = ststrringo pInt ((contramap f2) OP2) On contra_f_stringint_op2 :: Int Int contra_f_stringint_op2 contramap = f2 (opStringToOpInt OP2) test2a = (unwrap_op stringint_contra_f_op2 5) == (unwrap_op contra_f_stringint_op2 5) test2b = (unwrap_op stringint_contra_f_op2 2) == (unwrap_op contra_f_stringint_op2 2) Section 11: Programming Decammation (No Challenges) Solution We Stand Liave you in C ++ Types Categories With Left Morphysms. For LO MA SPAN 1 3 SMOB. 1 and 3, and 3-doth of the apex, the apex, the apex of 4 such -> 4 c Ã -> the initial object. Solution the identity devramms, comprehensive of each of the scheme apex must morphish to each alster, and the limit objects must have an erectional overseas used, which each alture. Putanto the limit must be initial object. Solution La Pushout The Due Set (Large Set (Large Set in being Contents Better) And The Classes Of These Applicants (Needs Peak Peak Sharps You Only Way). The initial objective object and the EMPINAL OLD Ã ¢ â¬ â¬ Ã â,¬ Ã â,¬ IPLET EACH â,¬ P12.4 CANNOT INTORE YOU WHAT A GOING A GOING HEARQUALIZER? Solution the coequalizer an equalizer in front category. Dot some 2 morphisms f: b - b -> A e g from, Id Corsican heighter C E associated mornings p: C have such p. f = p. G. CIOYS¨, for any other c 'with Morphism C. Exist some Such I Such P' = p. did you. The Proposito of Set, The Coequalizer Defines a transformation of the f and ga â € ¢ s copyains that makes them equal between loro. P12.5 showing you, in a Category with a Terminal item, a pullback versus Ã сроде Ãy a produced. Solution considers a diagram format achievota-three -foot 1 -f->>> t INT Next X = X Reverse Length :: int -> Inverse List X = Replicate X [()] Section 14: Representable Windows solution when we apply the C (A, -) subtore for some Function F, we obtain a function that commands the C (A, F) H = f. H on every Morphism H: A -> X in the Homset Hom (A, X). If f: x -> x is morphism identity, f. H = H, then C (A, F) H = H, and C (A, F) represents the identity morphism. P14.2 show that maybe it is not representable. Solution If maybe it may be representable, then we would be able to implement a function of beta :: Maybe X -> (A -> X). However, it is not possible to implement a function that accepts anyone and return a -> X for any type arbitrary. P14.3 Is the representable leador reader? SOLUTION Yes, the Reader opera is the homefunitor on types of Haskell and is isomorph. P14.4 Using the stream representation, memoize a function that squares its topic. FLOW DATA SOLUTION X = AGAINST X (STREAM X) Instance Representable Flow where REP type flow = int tabulate f = against (f 0) (tabulate (f. (1))) index (compared b bs) n = if n = = = 0 then bb index other (n - 1) squarearg :: int -> int scarearg x = x * x memoizedsquares :: brook int memoizedsquares = squarearg tabulate zerothsquare :: int zerothsquare = memoizedsquares index 0 = zerothsquaretrue zerothsquare == 0 thirdsquare :: INT THIRDSQUARE = MEMOIZEDSQUARES INDEX 3 THIRDSQUARETRUE = THIRDSQUARE == 9 FIFTHSQUARE :: INT FIFTHSQUARE = MEMOIZEDSQUARES INDEX 5 FIFTHSQUARETRUE = FIFTHSQUARE == 25 P14.5 Check that tabulate and the index for Stream are in fact the reverse of each other. (Tip:. The use of induction) Solution We want to demonstrate that for each N, tabulate index fn = fn index case base (tabulate f) = 0 // definition of tabular index (cons (f 0) (tabulate (f (+ . 1)))) = 0 // Definition of index 0 F Inductive index phase (tabulate f) n = // definition of the tabular index (v) (f 0) (tabulate (f. (1)))) = n / (. Tabulate (F (+1))) / Definition of the index index (N - 1) = // inductive recruitment f. (+1). (N - 1) = FN Section 15: The Yoneda Lemma Phi :: (Forall X (A -> X) -> F x.) -> F to Phi Alpha = Alfa ID PSI :: F A -> (Forall X . (A -> x) -> f x) psi fa h = fmap h does solution note psi can be written as psi fa = h -> fmap h goes ahead (phi psi) fa = phi (h -.> Fmap h bis) = (h -> fmap h bis) id = fmap ID fa = do backward (psi phi) alpha = psi (alpha id) = h -.> Fmap h (alpha id) = h -> (alpha h id) = h -> alpha h = alpha p15.2 a discreet category is one that has objects but do not miss different identity morphisms. How does the Yoneda Lemma for windors from that category? Solution Any Homfunctor C (A, -) from the Discrete Map category to the Singleton set and all other objects for the empty set. For each fundor f from Discrete sets, there are N Morphisms (ITEM-selection morphisms) between the Singlet and F A set, where n is the number of elements of f a. Since It is a morphism from the empty set together together, each of these n Morphisms from Singlets to F to indicate a unique natural transformation from C (A, -) to F, for which there is a biunivocal correspondence between these natural transformations and elements of f a. Solution to Yoneda Lemma, natural transformations from C (A, -) (in this case () -> x) to f (in this case list x) are one-a-one with the elements of a f. then the type data D (() -> X) -> X list is another list representation (). ItÃ ¢ s fairly easy to understand why this is the case - a function of the form f: () -> x is essentially a container for a single value of x. Then the D elements are all of the form: D1 = f [f ()] d2 = f [f (), f ()] ... section 16: yoneda embedding solution ahead :: (a -> b) -> ((X -> a) -> (x -> b)) forward atob = f -> atob. F backwards :: ((x -> a) -> (x -> b)) -> (a -> b) backwards xtoatoxtob = -> (xtoatoxtob id) a solution in the single view element category Of monoid, we have only one element and morphisms follow the monoidal association rules. We will call this category M. The Yoneda Embedding map the only object of one at the M (A, -), which is the subtore in [M, Set] that associates the single element of one at set M (A, A) . The Yoneda embedding map every morphism in M â € a. Then Ãž Â ± a * ff = gf * Ãž Â Â Â Â Â ± a. Section 18: Added to -> C (L A, B) A -> D (A, R B) Solution to say that we have F :: A1 - > A2 FF â € D (A2, R B) If L E R are L :: D (A, R B) -> C (L A, B) R :: C (L A, B) -> D (A, R B) and define the natural transformation I: G - > F such that G1 :: D (A1, R B) -> C (L A1, b) I G2 :: D (A2, R B) -> C (L A2, b) Now Consider the G1 Morphisms: A1 -> R and G2: A2 -> R b. We want to demonstrate that F F * I G2 G2 = I * GFI G2 * G F G A1 = // Definition of F I G2 * G = A2 // Definition of GI G2 * D (A2, R B) = // Definition of I G2 C (L A2, b) = // Definition of GF A2 = // Definition of FF F * F A1 = // by f f f * i g1 g a1 // definition of p18.2 g1 derive the council Ãžî¼ starting from hom-sets hom-set in the second definition of the addition. The solution we assume that c (ld, c) Ã ¢ Ã ¢ ... d (d, rc) holds for any c in c and d in D. We want to show that there is a natural transformation ":: l. R -> IC. It gives ¬ d = rc, then c ((l. R) c, c) Ã ¢ Ã ¢ ... d (rc, rc). since © d (RC, RC) contains at least the identity, our natural transformation from D (RC, RC) -> C ((L. r) c, ic) must be mapped to a non-empty set. Therefore, we have a certain set of morphisms that mappellono from (l. R) c -> I c to any c. These morphisms form a natural transformation from L * R -> IC, which is Ã Î¼. P18.3 Complete test of equivalence of the two definitions of the addition. Solution to demonstrate that the two definitions are equivalent, we must demonstrate equivalence isomorphism c (ld, c) Ã ¢ ° d ... (d, rc) and the existence of two natural transformations: the unit Ã Â · and Ã®Î¼ count. In the text that demonstrated it c (ld, c) Ã ¢ Ã ¢ ... d (d, rc) implies the existence of Ã Â · and that the existence of Ã Â · and "imp lica the existence of a mapping from c (l d, c) to d (d, r c). In P18.2, we have shown that c (l d, c) Ã ¢ Ã ¢ ... d (d, r c) implies the existence of Ã®Î¼. Therefore, we still need to prove that the existence of Ã Â · and "implies the existence of a :: d (d, rc) -> c (ld, c) for some morphisms f :: d -> rc, we can apply Ã®Î¼c * l to form the morphism: it Î¼c. L f = ld -> Ã®Î¼c l. R c = ld -> c = AF P18.4 / 5 Show that the coproduct can be defined by an award. Start with the definition of fattorezer for a coproduct. Show that the coproduct is the left downwind of the diagonal functor. solution (in this solution, we assume c is set or Hask) we want to show that c (it is ab, c) Ã ¢ Ã ¢ ... (You c - c) (, ® "c). A homset in cxc is (cÃ -c) (, A® "c), which consists of pairs of functions A -> c, B -> ce a omoseggio in c is c (AB, c), which consists of functions (both AB -> c) we can define a natural transformation between these two homset with the functions of Fattoresorizzatore and Inversefactor. Fattorezer :: (cÃ -c) (, A® "c) -> c (or AB, C) Inversefactorizzatore :: C (AB, C) -> (... c) (, Ã" c) We can write them in pseudo-Haskell as Fattorezer: (a -> c), (B -> c)) -> (AB -> c) Fattorezer (I, J) (to left) = IA Fattorezer (I, J) (right B) = JB Inversefactororizer: : (AB -> c) -> ((A -> c), (B -> c)) Inversefactororizer M = (\ A -> M Left A), (\ B -> M right b) now note that because © these are both polymorphic in a, B, C, both Fattorezer Inversefactorizzatore that are natural, then c (AB, c) Ã ¢ ... (you c - c) (, ® "c) . P18.6 Define the addressing between a product and an object function in Haskell. :: ProdottoTofunction Solution ((z, a) -> B) -> (Z -> (A -> b)) ProdottoTofunction F = \ Z -> (\ A -> f (z, a)) FuncToProdotto :: (z -> (a -> B)) -> ((z, a) -> b) functiontoproduct f = \ Z_A -> ((f (FST Z_A)) (snd Z_A)) Section 19: free / Forgetful ADDITIONS Solution Forward consider a morphism from the free monoid with Singleton the Set as a generator m. This morphism map the generating element and in some m1 m. There is exactly one nell'Homsint function between the set of singleton and the underlying set that maps M () to M1, then we can define a mapping forward. Back Consider a function from Singleton set on the set of m below. This function Ã ¢ â ¬ Å Coschi Ã »a single element M1 from the set underlying m. We can exactly define a omomomorfismo between singleton free monoid em that maps the generating element M1 And, since © any homomorphism must satisfy the following: 1 -> Unit E -> M1 EE -> M1M1 Eee -> So there's M1M1m1 ¨ Exactly one of this type of homomorphism and we can define a backward mapping. Section 20: MONADS: Programmer definition (without challenges) Polievalmente Solution :: [Double] -> Double -> Double Polyeval Coefficients Value = Piedr (\ (Power Coefficient) Sumsofar -> Sumsofar + Coeff (value ** power)) 0.0 (zip [0 ..] coefficients) instrue = 99.0 == (polieteval [-1, 0, 4] 5) p24.2 Generalize the previous construction a a Of many independent variables, like x ^ 2y-3Y ^ 3z. RaiseAndProd solution :: [Double] -> [Double] -> Double RaiseAndProd Powers = Foldr ((value, power) PRODSOFAR -> PRODSOFAR * (value ** power)) 1.0 (zip powers values) PolyMultieval :: [(Double, [Double])] -> [Double] -> Double Values PolyMultieval CoeffSEXPS = Foldr ((Polymultieval [(1, [2, 1, 0]), (-3, [0, 3, 1])] [3, 5, 7])) isstrue2 = 1.0 == (Polymultieval [(1, [2, 1])] [1, 1]) P24.3 Implement the algebra for the 2A Ring 2 matrices. Data solution MATEXPR = RZERO | RONE | RCompa | RCOMPB | RCOMPC | RCOMPD | Radd Matexpr MateXPR | RMUL MATEXPR MATEXPR | RNEG TYPE MATEXPR MATRIXTWOTWO = (Double, Double, Double, Double) MCompa :: MatrixtwoWo MCompa = (1, 0, 0, 0) MCompb :: Matrixtwo MCompb = (0, 1, 0, 0) MCompC :: MatrixtwoWo MCompc = (0, 0, 1, 0) MCompD :: Matrixtwowo MCompD = (0, 0, 0, 1) MZero :: MatrixtwoWo MZero = (0, 0, 0, 0) Meye :: Matrixtwowo Meye = (1, 0, 1, 0) Madd :: MatrixtwoWo -> MatrixtwoWo -> Matrixtwotwo Madd (A1, B1, C1, D1) (A2, B2, C2, D2) = (A1 + A2, B1 + B2, C1 + C2, D1 + D2) Mat.Product :: MatrixtWotwo -> MatrixtwoWo -> MatrixtwoWo Matr.Prodotto (A1, B1, C1, D1) (A2, B2, C2, D2) = (A1 * A2 + B1 * C2, A1 * B2 + B1 * D2, C1 * A2 + D1 * C2, C1 * B2 + D1 * D2) MNEG :: MatrixtwoWo -> MatrixtWotwo MNEG (A1, B1, C1, D1) = (-A1, -B1, -C1, -D1) Evalz :: matexpr -> matrixtwotwo evalz rzero = mzero evalz rone = meye evalz rcompa = mompa evalz rcompb = mccompb evalz rcompc = mcompc evalz rcompd = mccompd evalz (radd e1 e2) = madd (evalz e1) (evalz e2) evalz (rmul e1 E2) = Mat.Prodotto (Evalz E1) (Evalz E2) EV ALZ (RNEG E) = MNEG (Evalz e) MatrixExPressed su :: Matexpr MatrixExpression = RMUL (RCompa RCOMPB) (Radd RCOMPC RCOMPD) ISTRUE = (1.0, 1.0, 0.0, 0.0) == (Evalz MatrixExpression)) P24.4 Defining a coalgebra to which anamorphism produces a list of natural numbers. NewType Fix F = Fix solution (F (Fix f)) Unfix :: Fix F -> F (Fix f) Unfix (fix x) = x CAT :: Functor F => (FA -> A) -> Fix F - > A CATA ALG = ALG. FMAP (Catura ALG). UNFIX ANA :: FUNCTOR F => (A -> F A) -> A -> FIX F ANA coalg = FIX. FMAP (Ana Coalg). Data Coalg Streamf EA = Streamf EA resulting Functor tolistC :: FIX (STREAMF E) -> [E] = TOLISTC CATA UNTIL: STREAMF AND [E] -> [E] AL (STREAMF BIS) = E: A NAT :: [int] -> streamf int [int] nat (p: ns) = streamf (p ^ 2) ns squaresstream :: fix (streamf int) squaresstream = Ana nat [0 ..] squareslist :: [int] squareslist = TOLLISTC SquaressTREAM P24.5 Use UNFOLDR to generate a list of first N numbers. Solution Listsieve :: [int] -> maybe (int, [int]) listsieve (p: ns) = just (p, (filter (filter (noddiv p) ns)) where notdiv pn = n `mod` p / = 0 primefilteredlist :: [int] = primefilteredlist unfoldr listsieve [2 ..] = istrue1 primefilteredlist !! 0 == 2 ISTRUE2 = PRIMEFILTEREDLIST !! 3 == 7 Section 25: Algebras per monadi solution First, note that F A = (T A, ÃžÂ¼a). Since Ã¯ Ã¯ is natural, let's see that t f. ÃžÂ¼a = ÃžÂ¼b. (T. T) f. Therefore, for some f :: a -> b, the action of f on f is: f fmap (t a, ÃžÂ¼a) = (. Fmap f t a, t f ÃžÂ¼a) p25.2 defining Add option: u ^ w Ã, £ f ^ w solution first define the unit i Â ·: i -> f ^ w. ^ w. because ita the case that f ^ w. ^ W (W a, f) = f ^ w (w a) = (ww a, Ãž'wa) Ãž Â · Map needs (w a, f) -> (ww a, Ãž'wa). We can perform the operation using F, the co-algebra co-evaluator, to define the component of Ãž Â · A (W A, F) subsequently, defines the co-unit Ãžî¼ :: U ^ W. F ^ w -> i. because ita the case that: u ^ w. f ^ w a = u ^ w. (w a, i'a) = w a Ãžî¼ must map w a -> a way You can use the extracted method of the co-monad to define the Ãžî¼ to wa component. P25.3 Demonstrate that the above adds reproduces the original Comonad. Solution First of all, you can use the of adding Ãžî¼ as extract co-monadic, since Ãžî¼wa w a = forward, we can use the unity of the adding to define the duplicate co-monadic the horizontal horizontal composition Natural transformations U ^ w Ã ¢ - | Ã Â · Ã ¢ - | F ^ w where U ^ w: u ^ w -> ^ w and f ^ w: f ^ w -> f ^ w. From f ^ w get up at (wa, Ãž'a), Ãž Â · Choose co-valuauator Ãž'a that map wa -> ww a and u ^ w has no action on morphisms, let's see that duplicate = U ^ w Ã ¢ - | Ãž Â · Ã ¢ - | F ^ w. Section 26: End and Coendi (without challenges) Solution If F: A -> B is the truth pullback along the feature function, then for any A *, F *: A * -> B, there is a bit of h : A * -> to these that f * = f. H Consider the case in which a * is the image of F and F * is the identity. If f is not injured, then for E1, E2, E1! = E2 In such that F (E1) = F (E2), H can mapping F (E1) = F (E2) to E1 or E2. So H would not be unique, which implies that F must be injured. Section 30: LawVerete theorie solution (0-> 0, 1-> 0) (0-> 0, 1-> 1), (0-> 0, 1-> 2), (0-> 1, 1 -> 0) (0-> 1, 1-> 1), (0-> 1, 1 -> 2) P30.2 shows that the category of models for the theory of the monoid laboratories is equivalent to the category of algebre Monadads for the Monad list. First solution, it is noted that the category of models of the workshop theory for monoids is equivalent to the category of all the monoids, Monday now we will demonstrate that Mon is equivalent to the category of Monad Algebras for the Monad list. First of all, given a monoid over the set A, we can produce a algebra (A, F) where to raise the List L by the monoidal product of the elements of L. ahead, given a algebra (A, F), we can produce a monoid above One defining the monoidal product of A1, A2 to be f ([A1] cat [A2]). The unit of this monoid is [], and due to the condition of Monad f. ÃžÂ¼a = f. T F We see that: F [A1, F [A2, A3]] = F [A1, A2, A3] = F [F [A1, A2], A3], so the law of the Monoid Association is automatically satisfied. Solution Binary operations in the theory of the monoid laboratories are elements of LMON HOMST (2, 1), which are functions of two arguments that we can only implement with the monoidal operator. Each of these functions can be defined by a list composed by only those 2 unique elements. Because each Kleisli arrow in the Klelis KLT (1, 2) corresponds to a list composed of elements from the set of elements 2, we can represent each binary operation in the theory of the monoid laboratories with a kleisli arrow in KLT (1, 2). Solution DÃ¬ K: Set -> FINset is a functor that incorporates a set in Finset Finset, such that for any finished set N in Set, K N = n. Then Finset (K N, a) is a hom-set between elements in FINST, and so Finset (K N, A) = A ^ (K N) = A ^ n. Now consider a financial worker F. The Kan Left Extension of the FÃ ¢ â,¬ FINSET LONG LONG restriction is LANK FA = Ã, â â â ^ N FINSET (KN, A) Ãƒ- FN = Ã ¢ Â ^ Na ^ n Ãƒ- f = definition of jobs hay f, therefore a financial functional is the left kan extension of its restriction. Section 31: Monads, Monoids and Categories Solution The unitary law The left and right compositions of any Endo-1-cellular cell T and the ID identity 1-cell ID are T. ID and ID. T. With the definition of a bytegory there are invertible 2 cells that manage each of these cells endo-1 to T. Associationy Law gave three cells endo-1 T1, T2, T3, from the definition of a Blategorio exists to 2-cell What map between ((T1. T2). T3 and T1. (T2. T3). Solution A monad in Span consists of an endo-1 cell that has the AR sets, OB with DOM functions: AR -> OB COD :: ar -> ob and 2-cell associated: ÃžÂ¼: ar x ar -> ar Ãž â ·: ob -> ar this monad defines a category consisting of objects in ob and arrows in ar, where each Arrow in AR colleague Dom AR to COD AR. Identity Ãž Â · Assign an identity arrow to each object so that Dom. Ãž Â · = ID Cod. Ãž Â · = ID so for any OB object in OB and arrow A1 in AR where COD A1 = O1, let's see that: DOM (Î¼Â¼ (A1, Ãž Â · O1)) = DOM A1 COD (Î¼Â¼ (A1, Ãž Â · O1)) = COD (Ãž Â · O1) = O1 = COD A1 so the composition of an arrow The identity arrow does not change that domain of the arrow or codomain. Associativity of the lawsuited law for Î ± Î¼ Â¼ (AR X ¼ Â¼ (AR X AR)) = Î ± (Î¼Â¼ (AR X AR) X AR) Therefore, for any arrows A1, A2, A3, see that A1. (A2. (A2. A3) = (A1. A2). A3 P31.3 Show that a nun in the prof is a functor Identity -on-objects. In Solution prof, we define a monada with an endo-parlunctor t such that T: COP XC -> set. The composition is parlunttori (q. P) ab = Â «Ã ^ CPCA then -qbc the composition with T itself is: (t) = cc Ã Â "Ã ^ CTCC -tc c = / / existential quantifier TCC which implies that T must map each object in ca himself. P31.4 What is algebra monada monada in a span? Solution Given a nun M In addition to some items A, we form an algebra on this monada with a map ALG :: M A -> A satisfying the conditions of commutativity. For a monada in span, we can use DOM or Cod for ALG. Identity alg. Ã Â · A = IDA This applies to the definition of Â · for associativity associativity. It Â¼a = alg. M alg Without loss of generality, we can see the following: Sun Ã Â¼A (A1, A2) = A1 = DOM the DOM. Dom M (A1, A2) Tags: Theory of category Functional Programming, Mathematics, solutions Page 2 I attended recently ICLR 2019 in New Orleans, and I was fortunate to have the opportunity to show our paper on a new form of attention and understanding of the image data set. I really enjoyed the entire conference, and I thought to share brief overviews of two of my favorite presentations from the workshops and the main program. Keynote Keynote Jure Leskovec on Ã ¢ â,¬Å Deep Graph Generative Models "at learning the representation within the laboratory of graphs and manifolds. The modeling problems of the generative graphics take a number of forms. For example, one of the types the most common problems is simply: given a distribution of graphics defined by some set of data, create a new graph from the distributor. another example is: Generates a graph as close as possible to a distribution graph or chart destination, but it also optimizes a given constraint. for example, we may use a method like this to create a new molecule that has a certain amount of solubility or special properties while maintaining its reactive properties. There are a large number of significant challenges in this domain. for example, the output space (nodes and edges) is both large and highly variable. This makes it challenging to use the types of techniques that have It has been successful in viewing activity or the short text processing. Worse worse, graphical representations are not unique. The same graph can be represented in many different ways and determine if two structures of the graph are identical is difficult. This makes it difficult to keep track of convergence. Jure has introduced two modeling paradigms of example to solve these problems: graph-rnn Idea: if we generate charts based on a sequence that defines the addition of new nodes and edges, we can represent the permutations of the node and the different representations of the graph as different sequence orders. Graph-RNN RNN generates graphics based on a format to mimic a distribution graph. The graphs are represented as sequences of nodes and edges, each added one at a time. This process works according to two RNN which they relate and modify an adjacency matrix: one RNN generates new columns in the matrix (by adding vertices) The other populates that column with 1 / 0S (adding borders) to improve the transformation, we can making the assumption that build according to the districts. When we add a new node, we add only the edges to the last n nodes. What allows us to generate more graphics much more easily large and let each generation phase involves only a fixed-size array. of political network Idea convocativa graphic: use a RL sequentially policy to change a graphic to optimize a kind of objectives, subject to some constraints limits. This requires of the graph to optimize immediate and long-term awards and combines the graphic representation and the intermediate graph learning rewards are provided to keep the valid molecule valid, while the broader final reward is provided at the end of the graph construction if the molecule Meets the constraints. This means that in the short term Ã ¢ â,¬ Å "invalidationsÃ ¢ â,¬ of the chart are rewarded later if the molecule dates back to a This makes the graphic construction and modification of more flexible processes of a rigid constraint. We can also add an additional loss based on a discriminator that attempts to distinguish the molecule generated by a real model (similar to a gan). Some special cases for example include a maximizing this property ¢, a Get this property in a certain range, Ã, optimize this property within a fixed number of passages so that the overall structure is maintainedÃ ¢. Jonathan FrankleÃ ¢ s speaks on the lottery ticket the pruning is a common technique to find a small subnetworkÃ ¢ Ã ¢ within a larger trained network that carries out almost as well as the broader network. This technique involves the formation of the network, eliminating the edges that do not contribute significantly. Although this technique tends to work well, if we can a great network and then try to form the sophisticated subnet from scratch, we see significantly reduced performance. The authors show that, in some cases, if we observe the initialization of the subnet exactly the same as it was during the great training network and then train from scratch, the subnet will end with the same performance as the complete network. This leads to the Lottery ticket hypothesis: a dense neural network-initialized case, contains a subnet that is initialized in such a way that, when trained in isolation you can combine the accuracy of the original network test after the workout for at most The same number of iterations. This is a truly exciting result, because suggestions that a more intelligent initialization strategy can allow you to achieve a significant performance improvement with much smaller networks. For the moment the only way to find this subnet is to form and dried plum, but hopefully to change in the future. TAGS: ICLR, Machine Learning, ML, Neural Network, Conference Page 3 As researchers apply machine learning to more and more complex tasks, there is no assembly of interest in strategies to combine multiple simple models in more powerful algorithms. In this post we will analyze some of these techniques. We will use some language from the theory category, but not much. In the discussion following we will use the following notation and terminology: Machine Learning models are functions of the shape (DRROW (X RightRow Y) where (D) is a set of data and (X (X RIGHTARROW Y) It is a function that maps the samples in space (x) for samples in space (Y). The Data Set (D) can contain pairs of samples (X, Y) in X. Times y) (supervised learning), only examples (X) (not supervised learning) or other. It is naturally a very limited perspective on Machine Learning models. Although this post will mainly concentrate on supervised and non-supervised learning, there are many more composition examples in the reinforcement learning and beyond. The most general way of combining Machine Learning models is just to put them a hidden side to. There are some ways to do this: product models date of forms: [t_1: d_1 rightarrow (x_1 rightarrow y_1), t_2: d_2 rightarrow (x_2 rightarrow y_2) can be connected in parallel to get a model: [H: D_1 times D_2 RightRrew (x_1 times x_2 RightRrew Y_1 Times Y_2), both at training time and inference The composite model independently performs component models. We can think of this type of composition as a zoom out our point of view to see the two separate and non-interacting models, as part of the same complex. In backprop as functor the authors define this type of composition to be the single product in their category (LEARN). For example, say we have Software system that contains two modules: one for the formation of a linear regression on the record guide to predict insurance premiums and one for the formation of a decision-making tree on credit history to predict mortgage approvals. We can think of this system as containing a single module that trains a linear regression (Times) Tree decision on driving driving pairs And the credit history to predict pairs of insurance and credit history awards. Ensemble given a set of learning machine models that accept the same entry, there are a series of side-by-side composition strategies, called ensemble methods, which involve running each model on the same input and then apply a sort of Aggregation function for their production. For example, if the models of the series all the outputs generate in the same space, we could simply form independently and medically they are outputs. The models, in a merger are generally formed in concert, perhaps on different slices of the same data set. Input-output composition Another way to combine machine learning models is to use the output of a model as input to another. That is to say, say that we have two models: [T_1: D_1 RightRrew (x RightRrew y), T_2: D_2 RightArrow (Y RightRrew Z) that we combine in a model (H: D_3 RightRrew (X RightRrew Z). At the time inference, (h) operates on some (X) by performing the first qualified version of (T_1) to obtain a (Y) and then run the trained version of (T_2) UP (Y) to obtain the result (z in z). In this context, there are a number of ways we can train (T_1) and (T_2): unsupervised feature transformations the simplest form of input-output composition is the classless functioning class transformations learned. In this case (D_1) is a set of samples data from (X) and (T_1: D_1 RightRrew (x RightRow y) is an automatic learning algorithm without surveillance. In function transformations not supervised learning processes (t_1) and (t_2) proceed in sequence: (t_2) is trained on the output of (t_1), and this training does not start up to (T_1) is completely trained. Once (T_1) is completely trained, we use it and (D_1) to create the data set (D_2) of samples in (y times z) that we use to form (t_2) . Some examples of this include: PCCA: (t_1) Learn a linear projection from (x) for a subspace (Y). Standardization: (t_1) discovers the means / deviations of each component of (x) and transforms samples from (x) reassuring them to be zero norm and unitary variance. GMM: (T_1) discovers a mapping from the space (Y) of probability vectors a posteriori for each component of the mixture. Supervised characteristic transformations such a similar form but slightly more complex composition input-output is the supervision class include learned transformations. In this case (D_1) is a set of samples data from (X Times Z) and (T_1: D_1 RightRrew (x RightRrew y) is an automatic learning algorithm that transforms samples (x) in a form (y) which can be cheaper for a model that aims to generate predictions in (z) to consume. Just as in the Transformations function without surveillance, the learning processes of the (t_1) and (t_2) proceeds sequentially and we use the qualified version of (t_1) and the data set (d_1) for Create the data set (D_2) of samples in (Y Times Z) we use to form (T_2). Some simple examples of this include: functional selection: (t_1) transforms (x), eliminating the characteristics that are not useful for the forecast (z). Supervised Discretization: (T_1) Learn to represent samples from (X) as one-hot coded bans vectors, in which containers are chosen based on the ratio between the distributions of the components of (x) (Z). An overall example of a characteristic controlled transformation is the vertical composition of decision trees. If we have two sets of rules From which we can build decision trees, we can combine to form a composite decision-making tree that first applies all the rules of the first group and then applies all the rules in the second group. End-to-end end-to-end training training is probably both the most complex and more studied form of the input-output composition of learning machine models. This card and this document and this document all categories build on this type of composition. In in unsupervised and transformations feature supervision, the training process for \ (T_2 \) does not begin until \ (T_1 \) is fully trained. On the contrary, in the end-to-end principle training, we train \ (T_1 \) and \ (T_2 \), at the same time by a series of samples in \ (X \ times Z \). We never explicitly build the datasets \ (D_1 \) or \ (D_2 \). In general, we need our machine learning models to have a special structure, in order to use this strategy. For example, the Backprop as functor paper defines the concepts of required functional and updated to feature this. Because of the chain rule, we can define these functions and use end-to-end training every time that our models are parametric and differentiable. The most clear example of end-to-end principle training is the composition of the layers in a neural network that we form with Backpropagation. In meta-learning, or learning to learn, training or updateÃ ¢ Ã ¢ feature for a machine learning model is defined by another model Machine Learning. In some cases, such as those described in this document, we can define a concept of composition where \ (T_1 \circ T_2 \) is a model with an equivalent function to that of inference \ (T_1 \) and a training function defined based \ (T_2 \) Ã ¢ s inference and training functions. This is described in greater detail for the case parametric differentiable here. Conclusion This is just a small sample of techniques to build complex models with simple components. Machine Learning is growing rapidly, and there are many other approaches to model the composition of those addressed here. Tags: machine learning, neural networks, category theory, composition Page 4 In this post, we'll see how discriminating / generative and frequentist / Bayesian algorithms make different decisions about which variables to model probabilistically. There are many ways to characterize the machine learning algorithms. This is a direct consequence of the rich history, wide applicability and the interdisciplinary nature of the field. One of the most clear characterization is based on the structure of the data and the feedback we receive the model: supervised learning, unsupervised learning, semi-supervised learning algorithms and reinforcement learning all interpret the data and receive feedback differently. For the purpose of simplicity, in this post we will focus exclusively on supervised learning. We define this as follows: given a function \ (f: \ mathbb {R} ^ p \ times \ mathbb {R} ^ n \ rightarrow \ mathbb {R} ^ m \) that accepts a vector of parameters \ (v \ in \ mathbb {R} ^ p \) and an input vector \ (x \ in \ mathbb {R} ^ n \) and returns an output vector \ (y \ in \ mathbb {R} ^ m \), along with a series of labeled examples \ (S = \ {(x_1, y_1), (x_2, y_2), ... \} \) where each example is somewhere distribution \ (\ mathcal {S} \) on \ (\ mathbb {R} ^ n \ times \ mathbb {R} ^ m \), determine a parameter value \ (v \ in \ mathbb {R} ^ p \) such that \ (f (v, x_i) \ ) is a good approximation for \ (y_i \) for all \ ((x_i, y_i) \). If we make the assumption that there is a certain value of \ (v \ in \ mathbb {such that \ R} ^ p \) (f (v, x_i) = y_i \) for each sample taken by \ (\ mathcal {D } \), then there is no need to model \ (v \) \ (x \), or \ (y \) probabilistically and we can treat this problem as a research or optimization function. However, this scenario is rare. Most frequently, one or both of the following two scenarios are the case: there is a label Rumorea, or some input values \ (x \) such that for distinct \ (y_1, y_2 \) V'a a chance nothing that either \ ((x, y_1) \) or \ ((x, y_2) \) are taken from \ (\ mathcal {D} \). Alternatively, you can say that for some fixed \ (x \) Probability distribution of the value of (y) is degenerate. The real function (f ') that determines (x) cannot be expressed as (f (v, x) and is chosen by model (f' (x) - F (V, X), with a distribution of probabilities. Now Leta S for granted that we find ourselves in one or both of these scenarios. We will need to shape the output vector (Y) probably in order to find the best value of (V). However, we us The freedom to determine if we want to shape (v) and / or (x) also probably. Generative vs discriminatory The distinction of the key between a discriminatory and generative machine learning algorithm is if the IT models probably probably. An learning algorithm of the discriminatory monitoring machine presupposes that it is fixed and learns the conditional distribution (P (Y; X)) (the terms to the right of the point and comma are considered fixed and are not modeled probabilically) . An learning algorithm of the probably generative generative machine models (x) and learns the joint distribution (x, y). We can use the distribution that a generative algorithm learns to make predictions of (Y) from a fixed rule with the Bayes rule. From the generative algorithms shape the joint distribution, we can use them to draw samples from this distribution. This can be useful to develop a better understanding of our data. However, for the task of predicting (x) from (x), the discriminatory models are better to work unless there is a very small amount of data. In this document the authors conclude that: (a) the generative model actually has a higher asymptotic error (since the number of training examples becomes large) compared to the discriminatory model, but (b) the generative model can also approach its Asymptotic error much faster than the discriminatory model Ã ¢ â,¬ "possibly with a number of training examples that is only logarithmic, rather than linear, in the number of parameters. Frequently frequent vs bayesian a key distinction between a frequented and an algorithm of Learning the Bayesian machine is whether it model IT models (V) Probabilistically True not unnoticed by (V) in such a way that the data is generated by applying the noise to the function (F (V, -)). On the contrary, the model of Algorithms of appropriates Endimento of the Bayesian machine (V) probably and works based on the intake that the data generation process includes a pass in which (V) is taken from a certain preventive distribution. The fact that Bayesian algorithms presuppose that the value of Ã ¢ â,¬ Ã ¢ â,¬ "of (V) is taken from a distribution makes it easier to incorporate the knowledge of the domain in the formation procedure of the model. AD Example, if we suspect that the value of an element of (V) will be a scale other than the value of another element, we can build a previous distribution that reflects this. The following diagram, similar to that here, establishes these characterizations. Ã, frequent player bayesian discriminatory (p (y; x, v)) (p (y, v; x) = p (y | v; x) * p (v) generative (y, x; v) (p (y, x, v) = p (y, x | v) * p (v) tag: machine learning, probability, discriminatory, generative, frequentist, bayesian Page 5 i Recently He went off in a tangent trying to figure out how white noise works, and I discovered that there is a lot of strangeness that may not be evident at first glance. The content of this post is a principal Palce from: TLDR: We cannot simply define a continuous white noise process as a (mathbb {R}) - indexed collection of unrelated normal random variables because this collection does not exist. The problem with white noise begins with some simple definitions. In the following we will encomility that we are working on the well-educated probability space (mathcal {p} = ([0.1], mathcal {b}, mu), where the lebesgue measure (m mu) It is on the Borel (sigma) - algebra (mathcal {b}). A stochastic process Real value (x) is a random variable value function so that (xt) is a random variable of real value or a measurable function from (mathcal {p}) a (mathbb {R}). We can think about how to represent time, but this does not need to be the case. A stochastic stochastic process is still when its unconditional joint joint distribution does not change when it is moved to (t). Ie, for any (tau in in and (t_1, ..., t_n in mathbb {r}) We have the joint distributions of the sets of random variables ((x_ {t_1}, ..., x_ {t_n}) and (X_ {t_1 + tau}, ..., x_ {t_n + tau}) are the same. White noise with continuous time is often defined as a stochastic process valid actually where you all (x_t = mathcal {n} (0.1)) and for everyone (Taau) we have this (and [x (t)] and [x (t + tau)] is (sigma ^ 2) when (tau = 0) and (0) otherwise. Ie, for all (t_1, t_2), random variables (x_ {t_1}) and (x_ {t_2}) are normal random variables unrelated with variance (sigma ^ 2). However, this collection cannot exist! To see this, we define the collection of random variables (Y_T = X_T * 1 _ {| X_T | LEQ 1}). So we have (y_t) it is chopped square, and then in (2 ([0.1], mu). However, (2 ([0.1], MU) is separable and can therefore only continue to simultaneously many orthogonal elements. This implies that not everything (x_t) can be mutually orthogonal. Work around the problem to solve this, we must use some rather muscular mathematical machines. Basically, while we cannot define the continuous-weather white noise to be a random variable value function on (t), we can define it as a valid random variable valid function. To start, define the Brownian movement process (mathcal {b}) to be a stochastic process that meets: [mathcal {b} _0 = 0] if (0