DOWNLOAD PDF THE ALGEBRA OF PROGRAMMING

Chapter 1 : CiteSeerX â€” Citation Query The Algebra of Programming

Arithmetic Algebra. Programming is more than just coding, it is about solving racedaydvl.com How to solve it, PÃ³lya shows us that we can add to our repertoire of problem solving solutions by linking.

Because I always liked algebra and thought it was simple, this made FP appealing to me. Goals The first goal of this lesson is to give some examples of how FP code is like algebra. The mindset of this lesson is that each pure function you write is like an algebraic equation, and then gluing those functions together to create a program is like combining a series of algebraic equations together to solve a math problem. Gluing pure functions together felt like combining a series of algebraic equations together â€” i. I first learned and then taught Java and OOP in the s and early s, and with that background I always looked at problems from the eyes of an OOP developer. That never made me see writing code as being like writing mathematical expressions. I clearly remember my first thought when I saw the connection between FP and algebra: If I can write one pure function, then I can write another, and then another. This informal definition of algebra by Daniel Eklund fits my way of thinking a little better: For purposes of simplicity, let us define algebra to be two things: As emphasized, the key words in that sentence are set and operations. In the case of numeric algebra â€” informally known as high-school algebra â€” the SET is the set of numbers whether they be natural, rational, real, or complex and the OPERATIONS used on these objects can be but definitely not limited to be addition or multiplication. As an example, a set of natural numbers is [0,1, The key thing to realize here is that an algebra lets us talk about the objects and the operations abstractly, and to consider the laws that these operations obey as they operate on the underlying set. To find the operations that work with that set, you have to think about the problem domain. In a pizza store you add toppings to a pizza that a customer wants, and then you can add one or more pizzas to an order for that customer. The types are your set the nouns , and the functions you create define the only possible operations verbs that can manipulate that set. Given that discussion, a Scala trait for a Pizza type might look like this: Pizza def setCrustType t: Pizza def addTopping t: Pizza def removeTopping t: Assuming that all the values are integers, they can be written as these functions in Scala: For example, take a look at these Scala expressions: Using simple substitution, the first two expressions can be combined to yield this: You can write the code in the three lines, or perform the substitutions to end up with just one line. Either approach is valid, and equal. What makes this possible is that other than getEmailFromServer src , which is presumably an impure function, the code: Only uses pure functions no side effects Only uses immutable values When your code is written like that, it really is just a series of algebraic equations. Algebra is predictable A great thing about algebra is that the results of algebraic equations are incredibly predictable. For example, if you have a double function like this: Because of this, you know that these things will always happen: What can possibly go wrong? You can demonstrate this problem for yourself. Remembering that Scala Array elements can be mutated, imagine that you have an Array[String] like this: I knew that a Java String was immutable, but based on my OOP thinking, I thought this was more of a pain than anything that was actually helpful to me. But when you think of your code as algebra, you realize that mutating a variable has nothing to do with algebra. You never mutate x, but instead you use it as a foundation to create a new value. A simple way to demonstrate this is to show what happens when a person changes their name. As I mentioned earlier, I never thought of my OOP code as having the slightest thing to do with algebra. Now I think of it that way all the time, and that thought process is the result of writing pure functions and using only immutable variables. You can tell the PBT tool to throw test values at your function, or 1,, or many more, and because your function is a pure function â€” and therefore has this algebraic property to it â€” the PBT library can run tests for you. Technically you can probably do the same thing with impure functions, but I find that this technique is much easier with pure functions. Later in this book: For example, this code is an ADT: Double extends Shape case class Square length: Double extends Shape case class Rectangle h: Summary In this lesson I tried to show a few ways that functional programming is like algebra. I showed how simple algebraic functions can be written as pure functions in Scala, and I showed how a series of Scala expressions looks just like a series of algebraic functions. I also demonstrated how a series of expressions can

be reduced using simple algebraic substitution. If you write your code using only pure functions and immutable variables, your code will natural migrate towards this algebraic way of thinking:

Chapter 2 : The Algebra of Types : programming

Describing an algebraic approach to programming based on a categorical calculus of relations, Algebra of Programming is suitable for the derivation of individual programs, and for the study of programming principles in general.

The laws Complementation 1 and 2, together with the monotone laws, suffice for this purpose and can therefore be taken as one possible complete set of laws or axiomatization of Boolean algebra. Every law of Boolean algebra follows logically from these axioms. Furthermore, Boolean algebras can then be defined as the models of these axioms as treated in the section thereon. To clarify, writing down further laws of Boolean algebra cannot give rise to any new consequences of these axioms, nor can it rule out any model of them. In contrast, in a list of some but not all of the same laws, there could have been Boolean laws that did not follow from those on the list, and moreover there would have been models of the listed laws that were not Boolean algebras. This axiomatization is by no means the only one, or even necessarily the most natural given that we did not pay attention to whether some of the axioms followed from others but simply chose to stop when we noticed we had enough laws, treated further in the section on axiomatizations. Or the intermediate notion of axiom can be sidestepped altogether by defining a Boolean law directly as any tautology, understood as an equation that holds for all values of its variables over 0 and 1. All these definitions of Boolean algebra can be shown to be equivalent. Duality principle[edit] Principle: There is nothing magical about the choice of symbols for the values of Boolean algebra. But suppose we rename 0 and 1 to 1 and 0 respectively. Then it would still be Boolean algebra, and moreover operating on the same values. But if in addition to interchanging the names of the values we also interchange the names of the two binary operations, now there is no trace of what we have done. The end product is completely indistinguishable from what we started with. When values and operations can be paired up in a way that leaves everything important unchanged when all pairs are switched simultaneously, we call the members of each pair dual to each other. The Duality Principle, also called De Morgan duality , asserts that Boolean algebra is unchanged when all dual pairs are interchanged. One change we did not need to make as part of this interchange was to complement. We say that complement is a self-dual operation. The identity or do-nothing operation x copy the input to the output is also self-dual. There is no self-dual binary operation that depends on both its arguments. A composition of self-dual operations is a self-dual operation. The principle of duality can be explained from a group theory perspective by the fact that there are exactly four functions that are one-to-one mappings automorphisms of the set of Boolean polynomials back to itself: These four functions form a group under function composition , isomorphic to the Klein four-group , acting on the set of Boolean polynomials. Walter Gottschalk remarked that consequently a more appropriate name for the phenomenon would be the principle or square of quaternality. There is one region for each variable, all circular in the examples here. The interior and exterior of region x corresponds respectively to the values 1 true and 0 false for variable x. The shading indicates the value of the operation for each combination of regions, with dark denoting 1 and light 0 some authors use the opposite convention. While we have not shown the Venn diagrams for the constants 0 and 1, they are trivial, being respectively a white box and a dark box, neither one containing a circle. However we could put a circle for x in those boxes, in which case each would denote a function of one argument, x, which returns the same value independently of x, called a constant function. As far as their outputs are concerned, constants and constant functions are indistinguishable; the difference is that a constant takes no arguments, called a zeroary or nullary operation, while a constant function takes one argument, which it ignores, and is a unary operation. Venn diagrams are helpful in visualizing laws. The result is the same as if we shaded that region which is both outside the x circle and outside the y circle, i. Digital logic gates[edit.

Chapter 3 : Functional Programming is Like Algebra | racedaydvl.com

"Algebra Of Programming" has been mentioned on LTU a few times (mostly in it seems). Unfortunately the book is not available on-line, and costs $ on Amazon!

Using linear programming allows researchers to find the best, most economical solution to a problem within all of its limitations, or constraints. Many fields use linear programming techniques to make their processes more efficient. These include food and agriculture, engineering, transportation, manufacturing and energy. It is used to make processes more efficient and cost-effective. Some areas of application for linear programming include food and agriculture, engineering, transportation, manufacturing and energy. Linear Programming Overview Using linear programming requires defining variables, finding constraints and finding the objective function, or what needs to be maximized. In some cases, linear programming is instead used for minimization, or the smallest possible objective function value. Linear programming requires the creation of inequalities and then graphing those to solve problems. While some linear programming can be done manually, quite often the variables and calculations become too complex and require the use of computational software. Food and Agriculture Farmers apply linear programming techniques to their work. By determining what crops they should grow, the quantity of it and how to use it efficiently, farmers can increase their revenue. Sciencing Video Vault In nutrition, linear programming provides a powerful tool to aid in planning for dietary needs. In order to provide healthy, low-cost food baskets for needy families, nutritionists can use linear programming. Constraints may include dietary guidelines, nutrient guidance, cultural acceptability or some combination thereof. Mathematical modeling provides assistance to calculate the foods needed to provide nutrition at low cost, in order to prevent noncommunicable disease. Unprocessed food data and prices are needed for such calculations, all while respecting the cultural aspects of the food types. The objective function is the total cost of the food basket. Linear programming also allows time variations for the frequency of making such food baskets. Applications in Engineering Engineers also use linear programming to help solve design and manufacturing problems. For example, in airfoil meshes, engineers seek aerodynamic shape optimization. This allows for the reduction of the drag coefficient of the airfoil. Constraints may include lift coefficient, relative maximum thickness, nose radius and trailing edge angle. Shape optimization seeks to make a shock-free airfoil with a feasible shape. Linear programming therefore provides engineers with an essential tool in shape optimization. Transportation Optimization Transportation systems rely upon linear programming for cost and time efficiency. Bus and train routes must factor in scheduling, travel time and passengers. Airlines use linear programming to optimize their profits according to different seat prices and customer demand. Airlines also use linear programming for pilot scheduling and routes. Efficient Manufacturing Manufacturing requires transforming raw materials into products that maximize company revenue. Each step of the manufacturing process must work efficiently to reach that goal. For example, raw materials must past through various machines for set amounts of time in an assembly line. To maximize profit, a company can use a linear expression of how much raw material to use. Constraints include the time spent on each machine. Any machines creating bottlenecks must be addressed. The amount of products made may be affected, in order to maximize profit based on the raw materials and the time needed. Energy Industry Modern energy grid systems incorporate not only traditional electrical systems, but also renewables such as wind and solar photovoltaics. In order to optimize the electric load requirements, generators, transmission and distribution lines, and storage must be taken into account. At the same time, costs must remain sustainable for profits. Linear programming provides a method to optimize the electric power system design. It allows for matching the electric load in the shortest total distance between generation of the electricity and its demand over time. Linear programming can be used to optimize load-matching or to optimize cost, providing a valuable tool to the energy industry.

Chapter 4 : Algebra of Programming by Richard S. Bird

This is the th. book in the Prentice Hall International Series in Computer Science. It's main purpose is to show how to calculate programs. Describing an algebraic approach to programming based on a categorical calculus of relations, Algebra of Programming is suitable for the derivation of individual programs, and for the study of programming principles in general.

By dbremner at Sat, Check out their home pages. Rumour goes that de Moor no longer works on this stuff; like all of us, he found it too abstract. If you can do product and sum types in Scheme, e. It is interesting that you called it abstract. I actually started looking into it because it seemed very practical at least from its table of contents. My understanding is that their work lays out the foundation for famous papers on folds the universality of fold, bananas and lenses, etc. They have one function for addition of two numbers, and a whole different implementation for summation of a list or a collection. All this is available in many, many papers, articles, etc. By shahbaz at Sun, The first few chapters cover material about folds and their equational reasoning rules that are well-covered in the literature. The coverage of unfolds in AoP is a lot weaker. The second half is where the book really shines. In the second half, they start talking about optimization problems, and they specify the optimization problems in terms of relations between the input and output. The beauty of their approach is that since they work in the category of relations, the specifications are arrows in the category of sets and relations, and can be systematically transformed into arrows in the category of sets and functions -- which turns the spec into an implementation. By neelk at Sun, What kind of optimization problems do they discuss and solve? From my own experience most mathematicians are unlikely to learn CT just for the matter of shining elegance or abstraction but because they need it as a tool for describing certain classes of algebro-geometric problems adequately, that are hard to solve - no toy problems. The way computer scientists speak about the "elegance of mathematics" and CT in particular resembles me sometimes in apologies of "elegance" by superstring theorists. By Kay Schluehr at Sun, An example of an algorithm in this category is the TeX line-breaking algorithm, whose derivation is one of the examples in the book. Incidentally, I use categorical ideas pretty much every day. By neelk at Mon, Are you familiar with anyone willing to teach CT to someone who is interested in the shining elegance and abstraction? This, of course, is entirely equivalent to if you were to take the set, impose an arbitrary order on its elements, and then fold the operation over the elements of the resulting sequence. You can divide up the set into arbitrary partitions, aggregate those in parallel, and then aggregate the set of partial results. This is a really important property for lots of database applications. By em at Mon, By shahbaz at Mon,

Chapter 5 : Programming is not Algebra â€“ Andy Skelton on WordPress

Algebra of Programming has 11 ratings and 3 reviews. Hugo said: This is a mind-blowing book. Mind you: if you come from a OO background, finishing the fi.

Being able to prove properties of programs using equational reasoning. By Ehud Lamm at Fri, What properties in particular are people interested in? By Carter Cheng at Mon, These three properties, taken together, give a programming language a very declarative feel. Commutativity means there is no implicit control-flow in the syntax i. Associativity makes it easy to rearrange and abstract common subgroups of declarations. Idempotence makes it easy to introduce duplicate expressions to help with refactoring, and later to eliminate duplicate expressions to help with performance. One can control many properties by tying them to algebraic structure. A very useful algebraic property is algebraic closure. Basically, algebraic closure is what makes a programming model compositional. Surprisingly, very few models of programming are truly compositional. Too many fail to achieve algebraic closure for critical functional or non-functional properties. Actors model is not compositional because individual actors have different concurrency properties than a group of actors. Programs using mutexes are not compositional because the composition of programs can deadlock even if both subprograms are individually safe. Remote procedure calls are not compositional because a sequence of calls has very different performance characteristics than creating one huge call thus we end up needing scripts or batching. Concrete class-typed OO is not compositional because it is difficult to present a group of objects as a single object. For easy, robust, scalable composition, a programming model must take into account all the important properties both functional and non-functional and ensure algebraic closure and control over those properties. That includes concurrency and synchronization, latency and efficiency, partial-failure handling and recovery, communication and side-effects. Failure to achieve algebraic closure forces programmers to be much more aware of the implementation details of the elements being composed i. A requirement to grok implementation details of components limits the practical scalability of the model due to limited capacity of programmers. A single language may support or model many programming models. Doing so can be costly - in terms of complexity - if it means violating algebraic closure whether within a model or at the boundaries between them. A duck-typed OO model obeying ocap principles would also be up there E language, Newspeak. Beyond a single programming model, one can also compare and grok models using algebraic concepts. Finding a homomorphism with equivalent behavior proves that one language is at least equal in power to another, and can help you understand a model. Finding two homomorphisms, one in each direction, means the languages have identical power. Finding an isomorphism two homomorphisms whose composition is the identity function indicates a tight relationship between models, such that one can be used easily to verify the other. Finding a homeomorphism between models is essentially a declaration of expressive and compositional equivalence - i. I tend to distinguish programming models based on the notion of homeomorphism. By dmbarbour at Mon, By using transformations that preserve the meaning FSVO, e. In practice, most of the examples in Algebra of Programming seem trivial, in that you could have figured the final program and even the correctness proof for some out without going through Squiggol The formalism introduced in that book is useful because it gives us tools to approach program optimization in a more systematic manner, leaving more human creativity, time and energy for other tasks. By pkhuong at Mon, Lots of optimization problems can be specified in generate-and-test terms ie, find the best solution out of this set of possible solutions , and the equational reasoning used to derive good solutions is quite pretty -- I really liked their derivation of the TeX line-breaking algorithm. By neelk at Tue,

Chapter 6 : Boolean algebra - Wikipedia

The Algebra of Programming group researches mathematically sound yet convenient techniques for manipulating and reasoning with programs, with a particular interest in the functional and relational paradigms and in generic programming. It seeks patterns in specifications, algorithms and programs, and.

DOWNLOAD PDF THE ALGEBRA OF PROGRAMMING

Chapter 7 : Five Areas of Application for Linear Programming Techniques | Sciencing

Reddit gives you the best of the internet in one place. Get a constantly updating feed of breaking news, fun stories, pics, memes, and videos just for you. Passionate about something niche?

Chapter 8 : Algebra of Programming | Lambda the Ultimate

Atze Dijkstra, Jeroen Fokker, S. Doaitse Swierstra, The Structure of the Essential Haskell Compiler, or Coping with Compiler Complexity, Implementation and Application of Functional Languages: 19th International Workshop, IFL , Freiburg, Germany, September ,

Chapter 9 : Algebra Of Programming (Bird, De Moor) | Lambda the Ultimate

/r/programming is a reddit for discussion and news about computer programming. Guidelines. Please keep submissions on topic and of high quality. Just because it has a computer in it doesn't make it programming.