【修正の際の注意事項】
・修正ご希望箇所がございましたら、直接作業いただければと存じます。履歴が残るかたちで編集できるようになっております。

・どうしても聞き取れなかった箇所は【??? 0:30:24】のようにタイムスタンプを押してありますので、お手数ですがご確認・捕捉をお願いいたします。

・大変お手数ですが、かならずこの画面上での修正をお願い致します。その他の形式での修正は、修正反映漏れの原因となります。ご了承ください。

・公開の際は、タイトル、小見出しの付与、スライドの挿入をさせていただきます。

・各記事のタイトルや記事中の小見出しに関しましてはログミー編集部に編集権があるため、記事内容ご確認後、記事公開時に付与される予定です。（基本的に事前確認はございません）

【スピーカー】
aiya000 氏

<h2>Semigroupとは？　Monoid？　環？</h2>

aiya000氏（以下、aiya000）：あいやと申します。今日は「Semigroupとは？　Monoid？　環？」というテーマで代数についての発表をします。よろしくお願いします。

（会場拍手）

推しVimはNeovimです。活動はTwitterやGitHubなどをやっています。このスライドは「代数に興味があるけど、ちょっと敷居高いなぁ」という人に向けて作っています。

4

内容です。まず前半はマグマ・半群・モノイド・群というものを紹介して、後半は擬環・環・体という代数を紹介します。これらをHaskellのロジックと型で記述していきます。応用例等は重視しないですが、軽く紹介したりします。今回は用語の厳密さよりも、初心者フレンドリーな表現を心がけています。

最後に、今回いくつか型クラスや関数が登場するんですけど、形が若干標準ライブラリと違ったり、そもそも標準ライブラリに無いということもあるので、ご了承ください。あとコードの大部分はコンパイルをちゃんと通しています。ご安心ください。

<h2>マグマ</h2>

本編を始めさせていただきます。まずは「代数って何？」というところから、「マグマ」という一番制約の緩い代数を説明していきます。

8

マグマというのは、足し算あるいはかけ算という二項演算を扱える代数構造です。

9

マグマはこんな型クラスで表現できます。

10

これが何かというのは置いておいて、インスタンスはInteger、あとリストと、BoolのAnd（&&）、Floatの+、あとUnit「()」などが、マグマの具体例としてあります。

11-12

これらの関数、「<>」や「++」と「+」って、「〜に閉じた二項演算」「〜の上の二項演算」と呼ばれるんですけど。

13

これは「aの値だけを取って、aの値を返す」というものです。

14

ここで誤解を恐れずにいうと、演算というのはHaskellだと関数のことなので、この場合下の「+」も「id」もIntegerに閉じています。

15

二項演算とは、2つの引数で値を返す関数のことですね。

16

まとめると、「+」はInteger上の二項演算です。このRationalのidは二項演算ではなくて、cons演算も閉じた演算ではないです。

マグマのまとめです。

17

マグマはこのようなかたちをしていて、「a」に閉じているので、何度も値を足し合わせることができるんですね。10+20+30+40みたいに。同じく、リストもマグマになります。

<h2>ある型の複数のマグマインスタンスについて</h2>

ここで型に対して複数のインスタンスが定まることがあります。どういう意味かというと、このIntegerの場合、プラス（+）とかける（＊）がどっちもIntegerに閉じた二項演算なので、どっちにするか困っちゃいます。

20

Boolについても同様で、Or（||）とAnd（&&）は、両方Boolの上の二項演算になっていますね。

21

この解決策として、Haskellらしいというか……、newtypeを作って、newtypeを通してインスタンスを定義してあげます。

22

このように、マグマのインスタンスを、「足し算のa」「かけ算のa」という注釈として利用してあげます。

23

この上のderiving instance、どう見てもderivingですね。これStand-alone derivingという、あとGeneralizedNewtypeDerivingってGHC拡張を使っています。

Boolに対しても同じで、Andのインスタンスとorのインスタンスをこのように分けてあげます。

24

もう1個あって、Xorも実はマグマとして定義できます。

25

なので、こんな感じで定義できます。Unitについては、インスタンスが1つだけ定まるので、ここでnewtypeは使いません。

<h2>半群</h2>

以上で代数の導入は終わりです。次に、半群というもうちょっと扱いやすい代数を導入します。

26

これは、標準ライブラリに意味の上で同じもの、Data.Semigroupというモジュールで実装されています。

半群は、二項演算に加えて、左右どちらから演算しても変わらないものになっています。

27

この法則を結合法則と呼びます。Haskellで表すとこんな感じです

28

ここで便宜上Eqインスタンスを要求してるんですけど、別にEqインスタンスはSemigroupに要求されるものじゃなくて、確認するための都合というのが注意です。

では、定義はどんなものかというと、このようにマグマを継承してあげます。

29

関数が入ったりということはないんですけど、二項演算が結合法則を満たすというマーキングとして、このようにインスタンスを定義してあげます。

じゃあ何ができるのかというと、結合法則を満たすと、こんな感じの多相的な関数を定義してあげられます。

30

これは「first」という、渡された複数のMaybeの値のうち、最初の有効な値を持ってくる関数とか。あと、渡された値のうち一番大きい要素を取ってくるだとか、そういうものが定義できます。

半群ってけっこう扱いやすくて、プログラミングでもこういう関数として応用が利いて、カジュアルに使えます。

31

ここで、マグマだけど半群になれない例は、Double、Floatがありまして。

32

これ「丸め誤差」によって結合法則を満たさない、おもしろい例になっているんですね。なので、現実にある実数は半群になるんですけど、コンピュータの浮動小数点数は残念ながら半群になりません。

33

まとめるとSemigroupは左右どっちから演算しても結果が変わらないよという代数でした。インスタンスはこんなもの。あと、Rationalの足し算・掛け算や、Orなどがあります。

<h2>モノイド</h2>

次はモノイドです。

34

モノイドって、みなさんもしかしたらほかの代数より名前を聞いたことあるかもしれないんですけど、これも標準ライブラリのData.Monoidに定義されています。

35

モノイドは、半群に加えて、単位元というものを備えた代数になります。この等式は単位元の法則です。Haskellだとこんな感じで書けます。

36

単位元emptyは、もう1つの値xに左からかけても右からかけても、他方を変えないという値です。

37

定義はこのように単純で、Semigroupに加えて、emptyって単位元を要請してあげます。そうすると、Integerの足し算については「0」、BoolのAndについては「True」がemptyになります。

38

例を見てみると、Integerの足し算だと3・5・7という値があるんですけど、ここに0をどっちから足し合わせても変わりませんよね、ということになっています。

39

このようにモノイドは、emptyを適用することで自明な初期値を自動で適用できます。

40

どういうことかというと、例えばsum関数を考えてみると、sum関数は初期値は0ですよね。その上からほかの与えられた要素の数を足していって総和を求めるという、そういう性質なんですけれども、まさにモノイドですね。

allも同じで、初期値がTrueです。

まとめると、モノイドは自明な初期値が定まった代数です。

ほかのインスタンス。

43

Integerの掛け算とRationalのかけ算。あとリストです。これはemptyの1と、あと空リストが単位元になります。

あとは、OrとXorもあります。Rationalの足し算もあります。

44

UnitはUnitです。

45

インスタンスになれない型として、NonEmpty Listがありまして、空リストがないので単位元がないんですね。

46

あとは、FirstとLast。これはMaybeのnewtypeなんですけど、最初の有効値を探して持ってくるのがFirstで、そもそも渡されたものが何もないと一番最初というのは考えられないので、モノイドではないです。

47

こんな感じで、emptyは、左右から足しても他方を変えない値になります。

<h2>群</h2>

では次の代数。

48

群という、けっこう制約が強い代数がありまして、群ではあるaの値xに対して、xの逆元というものが1つ定まります。

49

これをHaskellで書くと、このようにinverseという関数として表現できます。

50

「xとxの逆元を足し合わせると単位元になる」という法則ですね。

51

Groupは、モノイドを継承して、ある値に対してその逆元を取ってくる関数を追加したものになります。例えば、Integerの足し算だと、「10」とかそういう値に対して「-10」みたいなマイナスの値を取ってくるものです。Xorは実はidentity関数で、xの逆元がxそのものというかたちになります。

52

Integerの足し算は、こんな感じで0になりますね。

53

Xorについても、TrueとTrueはFalseになるし、FalseとFalseはTrueになります。ということで、法則を満たします。

54

モノイドであって群でない例はけっこうあって、これらは群になれないです。

55

AndだとFalseの逆元がなくて、OrだとTrueの逆元がない。考えてみるとわかるんですけど、片方がFalseだとAnd演算ってTrueにならないですよね。Or演算も同じで、片方がTrueだとどうしてもFalseにならないので、群にはなれないことになります。

56

リストも逆というものがないんですね。まさに群じゃない代表的な型だと思います。

57

Product IntegerとProduct Rationalはなりそうで、10の逆元は「1/10」って気がしちゃうんですけど、「1/10」はIntegerじゃなくてRationalなので、残念ながら逆元がないです。Product Rationalも、「0/x」に対して「x/0」というゼロ除算を発生させてしまうので、残念ながら群にはならないです。

58

群の応用は数学分野に強いかなと思います。個人的にはプログラミングで直に触れるのはあんまりないんじゃないかなと思います。

59

ほかのインスタンスはこんな感じです。Unitはいつものやつです。

60

まとめると、群aは、aのすべての値に対して、それの逆元が定まるものです。インスタンスは、IntegerとRationalの足し算、BoolのXorと、Unitがあります。

<h2>可換な代数</h2>

というところで、ちょっと寄り道をしてみます。半群とモノイドと、全部のところで横断する概念で、「可換」というものがあります。

62

「可換である代数とはなんぞや？」というと、xとyを足し合わせるのとyとxを足し合わせるのが同じという性質です。

63

この法則を交換法則といって、また可換とか言ったりします。

可換な半群は「可換半群」「アーベル半群」と言ったり、モノイドと群についても同様、「可換」とか「アーベル」とかって言います。

64

Haskellで書くと、交換法則はこのように書けます。

65

Abelianというのが可換半群、あとは可換なほかの代数のマーキングとして、インスタンスを定義します。

66

これの応用例として、ユニフィケーションという分野があるんですけど、それはなんぞやというのはここではあまり説明しなくて。端的にいうと、この1とCharのリストをConsするのは正しいのかそれとも正しくないのかという型推論に使えます。

67

半群であって可換半群でない例というとリストですね。リストは後ろから足し合わせるのと前から足し合わせるのじゃ順序が変わっちゃうので、違います。

68

というところで、Abelianはこのように定義できて、インスタンスはこのように定義できました。リストなどは交換法則を満たしませんでした。

ここで前半戦は終わりです。ここまでのまとめは、マグマ・半群・モノイド・群は、それぞれこのような要素を持っていました。

71

このように、「より強い代数」＝「より弱い代数+何か」というかたちで定義されます。

72

<h2>擬環</h2>

64

というところで、後半編です。

74

まず、擬環という「Rng」という代数があります。これは、可換群と可換半群の両方の性質を持った代数です。

75

分配ができる。分配というと、小学校の頃習ったかもしれないんですけど、下の等式を満たすような性質のことです。

Haskellで表すとこんな感じ。

76

ここで今まで使っていた二項演算を「加法」と言って、先っぽが内側に向いている演算子、加法と向きが逆になったものを「乗法」って呼びます。

77

Rngの定義はこんな感じでできるんですけど、そのうち加法群は、加法の二項演算って、emptyAとinverseAが加法群の部品になっていて、乗法半群は乗法二項演算になります。

78

加法群は群なので、このように加法の単位元、零元って呼んだりもするんですけど、あと各値に対する逆元が定まります。

79

擬環は、要するに「分配をできるよう定まったプロトコル」という捉え方ができます。

80

インスタンスとしては、まずInteger、この分配がみなさんに親しいんじゃないかなと思います。ここで足し算とかけ算、区別がなくなって、両方持っているのでnewtypeが消えました。

81

RationalもIntegerと同じ定義。

82

BoolはXorとAndの合わせになっています。

83

あとUnit。これはいつものですね。

84

というところで、擬環は一番シンプルに分配ができる構造でした。

85

こんな感じでインスタンスがあります。

<h2>環</h2>

次に、1つだけ擬環に概念を加えた「環」という代数があるので、軽く紹介します。

86

加法の可換と、あと乗法のほうがモノイドになっています。

87

乗法のほうに単位元がかかってくるので、RngにemptyM、multiplicative（乗法）のMを足してあげます。

88

これはかけ算についてなので、かけ算の単位元は1ですね。

89

Boolは、Andの単位元なので、Trueを定義してあげます。

90

Unitについてはいつものってことで。

91

こんな感じの、擬環をちょっとだけ拡張してあげたものになります。

92

<h2>写像</h2>

必要な概念なんですけど、今までの流れとは違うものを紹介します。話半分で聞いてもらっても大丈夫です。

みなさん、写像は好きですか？　簡単に写像って何かというと、こんな要素から要素へ割り当てるものですね。

97

ここで写像をちょっと拡張したものとして、半群の準同型写像というものを考えます。

98

半群の準同型写像というのは、まず半群が2つa,bとあって、この二項演算を「!」と「?」で区別します。

99

そのときに、aの値x,yがあったときに、このような等式を満たすようなfです。

101

「なんのこっちゃ？」てことなんですけど、準同型写像を表すHomoという型を定義してあげます。

102

これはaからbへの関数のnewtypeです。そして、ListからIntへの準同型としてlengthを定義できます。それを「ListAToInt」という名前で定義してあります。

103

このListAToIntは、先ほど書いたこのような法則を満たします。

ところで、これは半群に対するものだったんですけど、さらにモノイドに対する準同型写像や。

104

群に対するものがあります。

105

実は自己準同型写像とその合成はまたモノイドになるんですね。

107

ということはこの「Homo a a」みたいに行き先と行く元が同じ準同型写像は。

108

こんなふうにモノイドインスタンスが定義できます。

109

あとは、このreverse、duplicateという準同型写像があったときに、このような感じに、また別の準同型写像を合成として定義できます。

111

イメージはこんな感じ。

111

reverseはリストからリストへの準同型なんですけど、さらにそのreverseは自己準同型写像のものの値になります。

112

みなさん、すべての道はモノイドに通じます。どうぞ、この言葉をお土産に持っていってください。

（会場笑）

<h2>体</h2>

113

最後に、「体」という代数を説明します。

114

これ、全部のっけみたいな代数なんですけど、加法と乗法が両方、群です。ただし、乗法のほうは0を除いたものです。0を除くというのは、あとで見ていきます。

これ実は四則演算ができる代数なんですね。

116

「なんぞや？」というと、環に対して乗法逆元をまず加えてみます。インスタンスは、基本的なデータ型だとRationalぐらいしかなれないような厳しい制約を体は持つので、Rationalだけ。

117

0 ≠ 1というような、加法単位元と乗法単位元が異なることを要請するものですね。ここでいつものUnitさんはお亡くなりになってしまいました。Unitさん、またね。

（会場笑）

118

乗法逆元は、群でやったところと同じなんですけど、各値に対して逆元が定まります。

119

ただし、0を除くというのをRationalで見てみると、このようなR'という型に対して乗法群が考えられます。これは0、1/0、0がないんですね。というものに対して乗法群を考えて、あと普通に加法群を考えると、体が定義できます。

120

「0を除く」ってやらないと、実は、「0 = 1」とか「0 = a」だとか、全部0になるんですよ。びっくりですね。

（会場笑）

体は四則演算ができるものでした。

121

四則演算ができるっていっても、「引き算、割り算はどこ？」って思うんですけど、このように定義できます。

122

例えば、引き算は、足すの後ろ側にマイナスをつけてあげると、ちゃんと引き算になります。同じように、かけ算に対して後ろの分母と分子を入れ替えてあげると、ちゃんと割り算になってくれます。

最後のまとめです。

123

体は、環に加えて、乗法の逆元を求められる構造です。インスタンスは基本的なデータ型のうちRationalぐらいしかなかったです。

今日の内容はここまでです。おつかれさまでした。最後にまとめ。

127

代数はこのような7つ、あとそれ以上とか、各々のレベルで構造に制約を課してくれるものでした。どんな制約かというと、こんな感じでした。

128

最後に宣伝。<a href="https://aiya000.booth.pm/items/1040121" target="_blank">こんな本を</a>出しています。もし興味あったらよろしくお願いします。

以上で発表を終わります。ご清聴ありがとうございました。

（会場拍手）

<h2>質疑応答</h2>

司会者：質問のある方はいらっしゃいますか？

質問者1：すいません。全部の発表に質問してしまって申しわけないんですけど、ちょっとわからないところがあって。可換というのはユニフィケーションに使えるという話がちょっとわからなかったので、もう少し詳しく教えていただけますか？

aiya000：はい。ユニフィケーションってあるものが別のものと同じかどうかを調べることで、例えばこの式だと、1というNum aの値があって、もう他方はCharのリストがあります。Consは要素とリストをとるので、ここでNum  aと Charが同じかどうかというところを調べられます。

ええと……どうしよう。 Wikipediaさんに聞くとぜんぶ答えてくれちゃうんですけど。

（会場笑）

やばい。あんまり理解していないことがバレてしまった……。

（会場笑）

質問者1：時間がかかりそうであれば、懇親会とかでも大丈夫です。

aiya000：ごめんなさい。お願いします。

質問者1：はい。ありがとうございました。

司会者：ほかに質問のある方はいらっしゃいますか？ ちょっと数学数学した内容でしたが。はい、ありがとうございます。

質問者2：質問というか、ちょっと補足なんですけど。普通は代数的構造を代数とは言わないと思います。

代数的構造というのは、群とかモノイドのように、要素、つまり集合とその相手の演算が与えられる構造であって。一方、代数というのはもっと特殊な場合を指すことが多いですね。

私もあまり代数に詳しくないんですけど、有名なものだと何があるでしょうかね……。そうですね、環上の代数とかが有名ですかね。あと、リー代数とか。だから、できればそこは用語は正しく使っていただければなと思いました。ありがとうございました。

司会者：それでは休憩時間に差し支えているので、このへんにさせていただきたいと思います。あいやさん、ありがとうございました。

（会場拍手）