平成 22 年度 国立情報学研究所　市民講座　第 3 回
「プログラミングの科学　－積み木のようにソフトウエアを作るには？－」
講師：　胡　振江（国立情報学研究所　アーキテクチャ科学研究系教授）

◆　講　義　◆

・スライド 1「プログラミングの科学：積み木のようにソフトウエアを作るには？」

胡／国立情報学研究所の胡 振江（こ　しんこう）と申します。よろしくお願いいたします。
こんなにたくさんの人の前で発表するのは……学会はそんなに大きくないし、そんなにありません。結構、緊張しています。今朝、エレベーターに乗ったら、今日は何を話すのかと聞かれました。「プログラミングの科学」。具体的に何を話すのかと聞かれて、プログラミングの運算をやろうと思いまして、えぇっそれでいいのと言われました。
確かに難しい話ですが、この話は、92 年に日本に留学してからずっと研究してきました。
研究者の中では分かる話ですが、世の中では、ほとんど何も知られてない。
逆にこの機会を使って、こういう話を皆さんにお伝えできればと思って、この話題を選びました。分からないところがたくさんあると思いますけれども、よろしくお願いします。

「プログラミングの科学」、このタイトルを見ると、不思議に思う方もおられると思います。
プログラミングは、普通は技法で、科学とはあまり関係ないんじゃないか、と。
書けば書くほど、いいプログラムを作れるんじゃないかと思うかも方がいらっしゃるんじゃないかと。

もう 1 つ、不思議じゃないかと思うのは「積み木のように」ソフトを作るということ。積み木というのは面白いですね。たくさん部品があって、その部品を組み合わせることによって自分が作りたいものを作る、構成していく、そんなに楽しいものができるのかという方がいるかもしれません。
この話では、この 2 点に絞ります。まず、プログラミングの科学としてのあり方について説明したい。そして、このプログラミングの応用として、楽しく、積み木のような感じでプログラムを作れるということをご説明できればと思います。

・スライド 2「プログラミングとは？」

まずは、プログラミングとは一体どういうものか、というところから説明したいと思います。
皆さんは、たぶん知っていると思いますが、プログラミングは、問題があってその問題を解くプログラムをつくる。それが、プログラミングなのです。
もう少し詳しく言います。まず、解きたい問題の仕様がある。どんな問題か。
そして、自分が好きな言語、Java とか、C などの言語を使ってプログラムを作る。
作ったものを計算機の上で実行する。
そのプログラムを「作る」と「問題の仕様」間のプロセスをプログラミングと言います。
この言葉はよく知っていると思います。たくさんプログラミングをやっている人がいます。
世の中には、たくさんのプログラマーがいます。

・スライド 3「世界のプログラマー数」

この図は、世界にどのくらいプログラマーがいるかを示しています。InfoWorld によると、2010 年は、

すでに 1950 万人に達していると。結構たくさんいます。
その中をちょっと見てみます。
いちばん多いのはどこかというと、37%が実にアジアなんです。
その下にヨーロッパが大体 30%、アメリカは 23%。アジアが一番多いんです。
皆さん、非常に多いのは今はインドだろうとか、中国だろうと想像されていたと思いますが。
こういったプログラマーはプログラムを作っていく。
この講義では、皆さんに 1 つの問題を考えながら聞いていただきたい。
問題があった時にあなたはどうやってプログラムを作るのか。想像ではどう作るかを考えながら聞いていただきたいと思います。

・スライド 4「プログラミングの例題」
まず、プログラムの問題を見ていきたいと思います。1 番目の問題。数列の最大値の問題です。数列があって、その中での最大値を求めるというような問題。これは非常に単純な問題。解けないとプログラマーとは言えない問題です。この例では、数列があって、この中の最大値は 97。これは考えなくてもいいです。
次の問題を、ぜひ考えていただきたい。
同じ「最大」ですが、今度は、最大部分列和の問題。一種のプログラミングの問題です。
これはどういう問題かといいますと、対象は数列で同じです。その中で和が最大となるような連続する部分列の和を求める。先ほどの問題と、ちょっと違いますね。
問題をもう少し見てみますと、入力は、スライドのようにリストがあって、数列がある。
中にはたくさん数字があります。この数字には正のものもあるし、負のものもあります。
その中で、連続している部分列の和はたくさんあります。1 つだけでもいいし、連続しているものもある。このリストには、その和が計算できるものがあります。その和の中でいちばん大きい値を求めたい。
結果は、187 です、なぜか。この値はこの部分列の和です。他の部分列を探しても、その和は超えない。
これが 1 番大きい。

さて、このプログラムはどう作ればいいか。
皆さんはいろんな言語を持ってると思いますが、正しくていいものを作るには、どうしたらいいか。これはプログラミングの問題ですね。この問題を考えながら、聞いていただきたい。
最終的にはこの問題の解法を皆さんに紹介したいと思います。

・スライド 5「プログラミングは難しい？！」①
プログラミングは、簡単か難しいか、よく聞かれます。
プログラミングは簡単ではない。難しくない。矛盾しているように聞こえますが、まずはプログラムはいろいろな性質を満たさないとプログラムとは言えない。
どういった性質を満たさなければいけないのかと言いますと、まず、作ったプログラムが正しくないと、意味が全くありません。
例えば、先ほど最大値を求めるプログラムを作ってくれといいましたが、仮に最大値ではなく和を計算するプログラムを作ってしまったら、何も意味がない。本当に問題をきちんと解くようなプログラムを作らなくてはいけない。そして解くのではなく、「正しく」解かなくてはならない。正しさが非常に重

要です。

ここには、正しくないプログラムを作った場合の損失の情報を書いてあります。
これは古い情報ですが、2002 年にアメリカでは年間 590 億ドル、これが正しくないプログラムが作った損失です。日本だと社会的損失は約 3 兆円ぐらい。新聞を見ると、銀行のシステム問題などいろいろ損失が出たりとありますね。まず、正しいプログラムを作らないと全く意味がない。では、正しいプログラムを作ればいいかというと、それだけでもない。

・スライド 6「プログラミングは難しい？！」②

2 番目には、計算資源を効率的に利用するプログラムを作らなくてはいけない。
プログラムがあったとして、計算資源は有限です。メモリとか計算時間とか。作ったプログラムは、沢山、時間を使わなくてはいけない、沢山メモリを使わないといけない、あるいは作ったプログラムは無限の時間があれば正しい結果が出せるというものでは、何も意味がない、正しくて効率が良いものを作らないといけないのです。

計算時間と処理データサイズの一つの計算のコンプレスティブのグラフが書いてあります。ここで何を言っているのかといいますと、同じ問題を解くようなプログラムもですね、設計によって、書く方法によって、全然計算の複雑度が違うということを言っているのです。
同じ問題を解くには、入力のサイズに比例するプログラムを使って計算するものと、自乗、コストを長さ n とすると自乗の時間がかかるプログラム、或いは三乗いろいろあります。自乗、三乗、n 乗といくと段々計算時間が、非常にかかる。こういったものをそれぞれ、同じ問題を解くのですが、できるだけ計算時間がかからないものを作りたい。これが 2 番目の要求なのです。つまり、正しく効率のいいものを作るということ。

・スライド 7「プログラミングは難しい？！」③

更に、作ったプログラムを計算機に実行してもらうが、計算機が分かればいいわけではなく、作ったプログラムをいろんな人に見せる。だから、これからは変更しやすく維持しやすい形にしなくてはならない。作ったプログラムは、人が読まないといけない。そのために、重要なのは読みやすいプログラムを書かなきゃいけない。
ここには「プログラムの美学」そのスローガンをここに書いてあるのですが、東京大学の名誉教授の和田先生が提唱されています。美しいプログラムを書きましょうと。人が読んで楽しく読めて、分かりやすいプログラムを作らないといけない。
更に作ったプログラムはいろんなところで再利用したり、変更しやすい形にする必要がある。スライドにはプログラムの部品化と書いてあります。こういう性質を満たすプログラムを作るのは、簡単ではないのです。

・スライド 8「プログラミングは創造性の高い活動」

さて、この講座では、正しくて効率のよい、美しいプログラムをどう作ればよいかをお話します。それは、単に技法ではないということをお伝えしたいんです。
その科学的なあり方について説明したい。難しいですが、実際にはプログラミングは創造性の高い活動

なのです。単に技法だけではありません。人間の智慧が欠かせません。いろんな工夫が必要でなのす。
2 つの誤解が書いてあります。1 つは、計算機を使って自動的にプログラムを作ろう。
これは理想ですが、実際には正しくて効率のいい美しいプログラムを作るのは、少なくとも今はまだ、現実ではないです。計算機が自動的に作るのはなかなか難しい。
さらに、誤解 2。20 年まえぐらいによく言う話ですが、これは計算物理学者が書いたものなのですが、この雑誌に出ています。プログラミングは頭を使う必要がない、退屈な仕事だと。しかし、実際はそうではないと皆さんにこの講座で示したい。

・スライド 9「本講座では」

まとめます。
ここで我々が目指したいのは、正しくて効率がよくて美しいプログラムを作ること。それが目標なのです。そういうものを作るためには、人間の創造的な活動が必要です。これをやるためには、今日の講座で紹介したいのは、プログラミングの科学、もっと具体的に言うと、プログラムの運算という新しいプログラミングのスタイルを皆さんにご紹介したいと思います。おそらく、この言葉は初耳という方が多いかと思います。
Calculational programming
運算と言ったら数式の運算はよく聞いたことがあると思います。プログラムの運算。どんなものか・・・
この枠組みで一言で言うと、正さは、基本の部品であって、これを設計してこの基本部品を使って、それらを組み合わせることによって問題を正しく解く。
そして、それはあくまでも正しいプログラミングなんですが、あとは運算ルールによって効率的なものに変換していく。
そして、最後に美しさ。この講義では、あまり美しさは強調しませんが、基本的に関数を使って、プログラムを簡潔に表現していくというものです。

・スライド 10「鶴亀算問題：どう解決するか？」

プログラムの問題と、普通、我々が解く問題とどこが違うか。息子は高校 2 年生なんですが、結構、子どもに問題を解いてもらいます。
ここに書いている問題は、鶴亀算。これはプログラムの問題と全く関係ないです。
こういった問題を息子が 3 歳の時に与えました。そうしたら、そこで一生懸命解いているんですね。まず、問題を見てみましょう。
この問題は、鶴亀算ですが、鶴と亀が合わせて 6 匹います。足の合計が 20 本でした。
鶴と亀はそれぞれ何匹いますか?という問題です。

・スライド 11「幼稚園の子供の解答」

3 歳の子どもに解いてもらったら、結構、図が面白い。一生懸命にやりました。
3 時間で解答が出てきました。紙の上に、いっぱいの鶴と亀が出てきました。
どうやって計算したか。全部で鶴と亀が 6 匹、全ての組み合わせを考えます。
まずは、全てが鶴の時。鶴を書くの大変ですが、その足を数えてみます。
そうすると、12 本ある。我々が欲しいのは 20 本なので、12 本は正しくない。これではいけません。これを消して、もう少し書きます。1 匹、亀だった時に、14 本になった。

まだまだです。もう 1 つ、亀が 2 匹になった時に、16 本、こうやって 3 時間かけて描いて数えたら 20 本になって、やっと結果が出ました。鶴が 2、亀が 4 です。3 時間で、こんなことが出来ました。非常に嬉しかったです。3 歳の時には、この方法しか思いつかないんですね。全てのものを出来るだけ列挙して確認するというのは、自然なんです。

・スライド 12「小学生の解答」

子どもは成長します。次に、小学校に入りました。そして、同じ問題を解いてもらいました。今度は 10 歳で、塾に行きました。塾の先生は、こういう問題を解くにはルールがあると、単に列挙するのは、あんまり賢くない、今は賢い方法があるよと。60 匹だったら絵を描くのはどのくらいかかるか。ルールがあります。先生から、この方法を教えてもらったんですね。算術のルールを使ってこの問題を解く。6 匹全て鶴だとすると、足の数は、全部合わせて 12 本ですね。では足が 20 本ならば、20 は、6 匹全部鶴だと 12 なので、その差は 8 本、8 本の足が足りないことがわかっている。足りないが亀を 1 匹増やすと、2 本足が増える。この足りない 8 本は、割る 2 として、亀を入れればいいと。そして、亀が 4 匹。
全部で 6 匹なので、鶴は 2 匹。このルールを学生に教えました。小学生になったら、この方法だと先ほどよりは、大分早く解けます。

・スライド 13「中学生の解答」

中学生になって、この問題をもう 1 回やってもらいました。今度は、道具がいっぱいあるんです。先ほどのルールではなく、数学的なルールを持っています。特に方程式が解けるようになったんです。小学生には鶴亀算は難しいですが、中学生になれば、簡単に解けるんです。亀が k（匹）、鶴が t（匹）と仮定します。仮定すると簡単に方程式を作れます。
頭が k+t=6、足が 4k+2t=20。ここまで来たら、OK です。なぜかと言うと、方程式を解く理論を勉強したので、あとはこの問題を何も考えなくても、方程式だけ解けばいい。そうしたら、k=4、t=2。
小学校の算術問題は中学校に入って方程式を使えば、簡単に解けるものがたくさんあります。ここから何かヒントがあるでしょうか。
結局、人が持っている道具によって、この問題の難しさが変わるということです。
この例では、これを伝えたいということなのです。同じ問題でも使う道具によって問題の難易度はだいぶ変わります。方程式を使えば、小学生でもたくさん問題を簡単に解くことができるのです。

プログラミングも同じです。
このようにプログラムの問題があった時に、解くための道具がどこにあるのか。このようなプログラムのための方程式があれば、我々がプログラムを解く時、分かりやすい方程式を立てられる。
別の形の方程式ですが、これを何かシステムに渡して、そこの理論に基づいて、いいプログラムを作ってくれるようなものがあるといいですね。
そういうことを市民講座で説明したいということです。

・スライド 14「鶴亀算からプログラミング問題へ」

先ほどの、Learning problem、もう一度考えてもらいたい。最大部分列和の問題。先ほども説明をして、何回も繰り返して申し訳ないですが、この問題を仮に解くならば、あなたはどうやって解きますか？
もし、今日終わった時、この道具があれば、ここまで書けば、あのツール、理論を使えば解けるな、と

なれば楽になります。そのような楽しく、楽にプログラムを作る方法を少しご説明できればと思います。
我々の目標は正しくて効率のよいものを作ることです。
この問題は、効率を考えなければ、計算資源がいっぱいあって、どんなに遅くてもよければ、それほど難しくない。
とりあえず正しそうなものを作る。効率を考えなければ簡単に解けますね。今、我々がやっている問題は、連続している部分列の和の最大値を求めるということなんですね。
そのままやりましょう。

・スライド 15「ナイブな解法」①
これは入力のリストです。
このリストの中で、まずは連続している部分列の和の最大値、連続している部分列を列挙しましょう。
先ほどの鶴亀と同じようですが。まず全て書きましょう。これは連続している部分リストなんですね。
頭から 31、－41 ももちろんそうなのです。連続しているような部分リストさえあればよいのです。これを全て列挙します。たくさんありますが、何個あるかは、おもしろい数学問題なんですが。始点と終点を決めれば、1 つの連続した部分列は決まるのですが、その組み合わせを考えれば、何個あるかは分かると思います。
長さが n だとすると結構たくさんあります。部分列リストの和の最大値、それぞれの部分列リストに関して、和を計算しましょう。

・スライド 16「ナイブな解法」②
31 の和は 31、31－41 の和は、－10。
計算したら、ここは 187。最後に全て計算しましょう。計算したら後は、このような赤の和の最大値を求めればいい。

・スライド 17「ナイブな解法」③
求めたら、187。このプログラムは、ほとんどの人が書けます。効率は何も考えていませんが。

・スライド 18「ナイブな解法」④
まとめますと、このナイブな解法は、問題そのまま解いているのです。全ての部分列をまず求めて、部分列ごとの和を求める。最後に部分列ごとの和の最大値を求める。このプログラムの問題をちゃんと解いたということです。
この解き方は、全ての候補を生成し、それぞれのテストをして、結果を求めるという、生成、テストという形なんです。ただし、この手法を見ますと非常に時間がかかります。
先ほど言いましたが、部分リストは、長さ 100 だとすると、部分リストはいくつあるか。この講座の後で皆さん考えていただきたいのですが、たくさんあります、数千とあるんです。要するに長さ n だとすると、専門的な言葉で言うと、n 自乗法のものがでてくる。
それぞれの部分リストの和を計算しなければいけない。計算したらまた最大値の計算をしなければいけない。たくさんの計算がその中にあるのです。実際には足し算の数は入力リストの長さの三乗に比例する。非常に時間がかかります。正しそうですが効率が悪い。
我々がやりたいのは、正しくて効率のよいものを作ること。効率を考えなければ、すぐに作ってしまい

ます。この問題でも、効率のよいものを作れるのです。

・スライド 19「最大部分列和問題の賢い解法」

次のスライドはこの問題は賢い解法があります。上は、入力のリストです。
この計算、左側から 2 つの配列について、もう少し補助用の配列を作っておいて、左側から右に計算して最後にここ（一番右端）に来たら、ここ（右端の値）が絶対に我々が欲しい値なのです。どうやって左から右に行ったかと言いますと、2 つの配列の初期値は 0　0 スタートなんです。
表を埋めていきます、どうやって埋めるか。和と 0 の大きいほうの値を求める。
まずは上を埋めます。31 と 0 の和は 31、0 と比べて大きい。31 と－41 の和は、－10。
0 と比べて大きいほうだから 0。値は 0 以下なのでどうやって計算していく・・。
下の数字はどう埋めていくか。これ（0）とこれ（31）の大きいほうを持って行く。
　＊映像参照　31：55
左から右に順に、こうやって表を埋めていけば、最後に出る値は、我々が求めたい値です。
ちょっと不思議だと思いませんか？一見非常に難しい問題ですが、こんなに簡単に解ける。
計算時間は非常に速いですね。先程と比べると。1 回だけです。この計算は、正しいですか？先ほどの例は正しかったのですが、では本当に正しいかどうか。効率はいいですが、正しいことを示さないと分かりません。両立はなかなか難しいですね、正しくて効率のよいものを作るのは両方作らなければ、満たさなければならないのです。

・スライド 20「正しくて効率のよいプログラムを作るには？」

さて、正しくて効率よいプログラムを作るには、両方満たさないといけないのですが、どう作ればいいか？中国の 1 つのことわざがあります。
孟子がこう言っています。「鱼和熊掌不可同时兼得」魚と熊の掌（てのひら）、これは中国の中で価値のある宝物です。この 2 つは不可同時兼得。同時に得ることは出来ないよ、と言っています。同時に良い物を得るのは貪欲すぎてなかなか得られないということです。
ただし、少し赤い文字を入れ替えると、魚と熊の掌を同時じゃなくても違う時に得ることが出来ると。まず、魚を取って、しばらくたったら、もう 1 つ取ればいいと。同時じゃない時には、2 つの物を得ることができるよと。これが 1 つの方法なんです。

この方法に従って、正しくていいものを作るのは 2 つ手法があります。
1 つは、まず効率のよいものを作ってしまう。後で、作ったものの正しさを証明、検証する。
もう 1 つは、まず正しいプログラムを作ってしまう。その時には、効率は考えないんです。
プログラムの意味を変えずに、保存しながら、プログラムを変換していく。効率よいものに変換していく。この 2 つの手法には多くの研究があります。
この中には単に技法ではなく、いろんな科学的アクティビティとして議論しなければいけないのです。どうやって検証すればいいか。そして効率を保ちながらどうやって変換していくかも考えなくてはなりません。

・スライド 21「プログラミングを科学する（1/2)」

歴史から見ると、プログラムを科学するというスローガンを出している人は、30 年前、35 年前。Dijkstra

も重要性をアピールしてきた人です。計算機、コンピュータ科学の中のトップの賞は、チューリング賞で、コンピュータ科学のノーベル賞のようなものです。Dijkstra はその受賞者です。彼が書いた本。『a discipline of programming』
プログラムは経験によるものではないよ。数学的な活動だよと。プログラムを書いたら、その正しさをきちんと証明しなければならない。正しくて効率の良いものを作らなければならない。そのアクティビティは数学的活動だよと主張しています。これはほとんど検証を考えているのです。まずプログラムを作って、そしてプログラムの正しさを証明する。

・スライド 22「プログラミングを科学する (2/2)」
もうひとつの代表的手法としては、構成的アルゴリズム論です。
このようなものは 80 年代、IFIP:(情報処理国際連合)のワーキンググループがたくさんあるんですが、2.1 のワーキンググループが設計したもので、プログラムを構成するための 1 つの枠組みなんです。この枠組みがどういうものかというと、プログラムを運算する。先ほどの数式運算。
プログラムを運算の対象にして、まずプログラムを積み木のように基本ブロック、基本的な部品を組み合わせて構成していく。そして、この組み合わせで作ったものを効率的なものに変換していく。そういうようなものです。

実はこの研究は、先ほど申し上げましたが、私は 92 年に日本に参りまして、東京大学の武市研究室に入り、そこからこの研究を続け、非常に魅力を感じています。
本当に出来るか。出来るためには何をしなくてはいけないか、非常に興味があって、ずっとやって参りまして、あっという間に 18 年経ってしまいました。いろんな面白いものが出ているのですが、なかなかこのような機会が無いのですが、今日は、皆さんの率直なご意見をいただけると非常にありがたいと思います。

このツールは、実は我々の夢なんです。まだ出来ていないんですが、電卓のような感じです。プログラムの電卓、プログラムの Calculator を作ろうとしています。
どういうものかといいますと、普通の電卓は数字を入れたり、数式を中に入れたり、かけ算とか割り算、或いは因数分解とか、クリックしたら結果が出てくるんですね。
同じようにプログラムは、この形の方程式のような感じです。この形でプログラムを書いたら、ここに沢山ボタンがあります。「fuse」、「tuple」とかいろいろあって、これはひとつひとつの二次方程式を解くための装置みたいなものです。これをクリックすると、効率のいいものにもっていける。
もちろん、こういうのができるのは、理論があってこれは作れるよというのを示せないといけない、我々の夢です。今、5〜6 個のボタンができていますがまだ開発中です。まだ完全ではないのですが、でも幾つかは既に出来ています。
今日はこれについてもう少し詳しく説明したいと思います。

・スライド 23「運算によるプログラミング」
具体的に言うと、運算によるプログラミング、積み木のようにプログラムを作る手法を紹介したい。我々は正しくて効率のいいものを作りたい。まず正しいものを作ろう。まず正しいものを作ります。この正しいものを作ったら、全部終わるのではなく、この後、この形に書いたものは効率のいいものにもって

いけるよと保証できなければ意味がない。
結局、正しいだけで効率がよくないプログラムになってしまう。
こう書いたらその後は、システムが自動的に或いは半自動的に効率のいいほうに持って行ける。このような正しいプログラムをまず作ります。
「基本計算パターンと、その組み合わせ」ここで非常に違うのは、どんなものが基本計算パターンか。積み木をやる時やゲームをやる時には、基本ブロックがいろいろありますが、その基本ブロックの設計は非常に難しいですね。それを使って、ちゃんと組み合わせることによっていろいろ作れないといけない。このプログラムを作るために、この基本計算パターンはどういうものか、そして、計算パターンを組み合わせてどうやって問題を解くのかがポイントなんです。
それが出来たら運算理論。先ほどのボタンをクリックすると効率のいいものに持って行けるような枠組みです。
これから１つ１つ詳しく紹介したいと思います。では、基本計算パターンはどういうものか。

・スライド 24「積み木のように…」
基本計算は、プログラムは実際のデータを処理するものです。
何かデータをもらって計算を計算してデータを作る、その意味では一種の関数のようなものです。データはたくさんあります。数列、数字の列、ツリーとかグラフなどいろんなデータがあります。
それらを操作するようなこういう計算パターンをどう定義することが出来るのです。計算パターンはどうやって定義するかは、次のスライドで説明します。
プログラムは計算パターンを組み合わせることによって作っていく。
まず作ったものは効率が悪いかもしれませんが、いろいろな運算ルールの変換によって効率のいいものに自動化、或いは半自動化、あるいはもう少し頭を使って変換していくことで、効率のいいものを作る。というプロセスです。

・スライド 25「チャレンジ」
計算基本パターンはどういうものか。実は、この計算パターンのデザインが一番難しいです。どういうものを基本ブロックとして定義すればいいかが難しいのです。
基本スケルトンと書いてありますが、要するに計算パターンですが、たくさんのスケルトン、たくさんの計算パターンを用意したら、ユーザプログラムを書くときにどれを選べばいいのかわからなくなります。
ただし、とても少ない計算パターン、或いはスケルトンを使うと、複雑な問題にちゃんと対処できないので、バランスをとらなくてはなりません。
そして、基本ブロックがあって、それを組み合わせて問題を解くのですが、問題がきた時に、どうやってこれを組み合わせることによって問題を解くかは、非常に重要なんです。これは系統的な手法を開発しなくてはならない。
こんな形があれば組み合わせて良いプログラムが簡単に作れるよ、と。
そして、作ったプログラムは始めは効率は考えていませんが、最終的には、正しくて効率のいいものを作りたい。効率的なものを作るためには、いろんな変換が必要です。
先ほどの鶴亀算を方程式で解くような理論が必要です。それと同じように、変換の理論が必要です。融合変換や組化変換などが…そういったものは、これから後、もう少し説明したいと思います。

・スライド 26「リスト上の基本計算パターン」

どんなパターンがあるかについてみてみたいと思います。

今日はとりあえず、数列の問題を考えます。数列、リストですね。要するに列。

順序のある要素の並びをリストと定義します。ここは 3 つのリストを書きました。

一番上は空リスト。何も要素がない。2 番目は、4 つの要素からなるリスト。1,2,3,4

最後には、リストの中の要素は、またリスト。このリストを使えば、いろんなデータを表示、表現することができます。

こういったリストを操作する、計算するような計算パターンは、どういうものになるのか。ということですが、実は計算パターンは、この 4 つだけなんです。

1 つは map。

和訳は分からないので、英語をそのままにしました。

「こう訳したほうがいい」というのがあれば、ぜひいただきたいですが。

2 番目は、fold。

下が平坦になっていて、上に出ているようなものです。

scan、そして最後に zip。この 4 つです。

これらは、どういうものか。非常に単純な計算パターンです。ビルディングブロックなんです。これを使えば、いろんなプログラムを組み合わせて書けることを示したいと思います。

・スライド 27「リスト上の基本計算パターン：map」

まず　map。

リストを操作するのですが、ここは、リストをもらって、リストを作るというプログラムです。基本計算パターンで、これは一番単純な計算です。何をやっているかと言いますと、リストがあって、結果もリストなんです。

全ての要素に関して、同じ処理を行うというような計算パターンです。これは非常に単純です。普通、プログラム言語で書くと、だいたいこんな雰囲気です。リストがあってリストの要素が 0 番から n−1 番までですね。この結果、y[i]は f(x[i])です。

xi をもらって f を処理して、その結果は yi、 i 番のところに置くよと。

これは非常に単純な計算パターンで、これは構造は変わらないのです。計算の時、時々構造が変わるのです。

注意していただきたいのは、ここは計算パターンだということです。

パターンということは、f は違う関数を選ぶことによって、違うプログラムになるんです。

例えば全ての要素、リストがあってリストの全ての要素を全て倍にするのだとすると、この f は倍にする関数。例えばリストをもらって、リストの中の全ての要素を逆数にすると、この f は、x をもらって、1/x を返すような関数、あるいはプログラムになります。

その意味ではこれは、計算パターンなのです。同じ構造、リストからリストを作る。

・スライド 28「リスト上の基本計算パターン：fold」

次に、リストをもらって、実際には、1 つ値を返す、これは fold。fold はたたみ込みでしょうかね。どう翻訳するのか分からないですが、要するに集約する。

上のリストはｘ0 から、 ｘ1、 ｘn-1 までだったら fold を初期値と 1 つの二項演算子をもらって、すみません「二項演算子」という複雑な言葉が出ているのですが、とりあえず上からこのようにｅから x0、x1、 xn-1 とたたみ込んで、最終結果を返すような計算、パターンです。これが fold。
例えば、和を計算すると、初期値はｅは 0 でスタート、o+は足し算だとすると、リストの和を計算することになります。こういうような計算パターン。ただし、こういうものは、ビルディングブロックをちゃんと勉強しておく、覚えておく必要があるのです。

普通、プログラムを書くとこのように簡単なループになるんです。
この下は、要素の和を求める例です。

・スライド 29「リスト上の基本計算パターン：scan」
今は最後に一つ結果を出すのですが、
この途中の結果を全部、記憶するものなら、次のパターン、scan。
この scan は、先ほどの fold と非常に近いのですが、fold は、先ほどの上の全部の和が右下に来るのですが、今は、この下の y1 の値は、上の左端から x1 までをここに置くよといっているのです。要するに部分和を、途中計算した結果を全部記憶しておくというような計算パターンなのです。
プログラム言語で書くと、要するに結果はｙなのですが、y もリストなんです。このｙは、この前の部分、要するに上の左端から３マス目までの和、プラス xi だとすると、上の左端から xi までの和は、下の yi に来るということです。こういう計算パターン。
これは少し複雑ですが、計算パターンは 4 つしかないです、ここまでで４分の３は過ぎました。あと 1 つあれば、プログラムを書くことが出来るようになるのですね。
最後、ここは例えば接頭部分列和を求めるには、接頭からあるところまでの部分列の和を全て求めるのであれば、この Scan。この初期値は 0 であって、真ん中の二項演算子は足し算すれば、全ての接頭の部分列和を求めることが出来ます。まあこれは少しむずかしいですね。ビルディングブロック、積み木をやる時には簡単な三角もありますが、なかにはちょっと複雑なものもありますね。これはちょっと複雑です。複雑なのはこれしか無いのです。

・スライド 30「リスト上の基本計算パターン：zip」
次は zip。
これまでは処理する相手が 1 つのリストでしたが、これは 2 つリストがあった時どうしたら良いのかというものです。
２つのリストがあった時に、この zip は対応する要素を一つのところに持っていく。そうすると２つのリストが１つのリストになる。その後、1 つのリストの上で、いろんな計算をやっていく。そういうような演算なのです。これを使うといろいろなプログラムを書くことができます。

・スライド 31「プログラミング例」①
例えば、数列の要素の自乗の和を求める。数列があって、全ての要素をまず自乗して、この自乗の和を求める。だったら、この２つスケルトンですね。こう組み合わせれば良いと。
こうやって組み合わせるというのは、この結果（下の赤色）はここ（上の青色）の入力のところに持って行く、そういうことです。

まず、map を使って各要素の自乗は簡単に計算できます。先ほど、各要素を倍にする map を使いました。これ簡単に書けます。この結果はまた各要素の和を計算する、先ほどの fold を使ってやりました。この 2 つを合わせると、数列の要素の自乗の和を求めることができるのです。
もう少し何か単純な例が出来ると良いのですが、もう少し複雑に・・・

・スライド 32「プログラミング例」②

今度は、問題はこれです。接頭部分列和の最大値を求める。数列の中で、接頭から沢山連続して部分リストがあるのです。その中の和の最大値を求める。効率は考えず、単なる計算パターンの組み合わせで考えれば簡単です。
接頭の部分列を列挙するのは、scan を使えば列挙できます。（下の黄色）その列挙した結果を map に渡す。（真中の赤色）各列の和を計算する。最後に、要素の最大値を計算する fold。（上の青色）これらを組み合わせれば、全部解くことはできます。
少しインフォーマルな表現ですが、とりあえずこんな感じです。

・スライド 33「プログラミング例」③

さらに、これは皆さんに考えていただきたい問題なのですが、連続の部分列の和の最大値を求める。これは、私達が枠組みで解くとすると、こうなります。
まずは下の緑はまだブラックボックスなのです。
仮に、この部分は、連続の部分列は列挙できる、できたらその後は各部分列の和を計算する、map です。最後に最大値を計算するのは fold。そうすると全部計算できます。
まだ、緑の部分は残っています。ここはもう少し見てみます。

・スライド 34「プログラミング例」④

ここをやるには、同じように scan、map と reduce の組み合わせで表現できます。そうすると、先ほどの問題はこの枠組みで簡単に解ける。
こうやって解いたら後は運算理論によって効率よくもっていけるよというのが理想なのです。
これらのやり方を使えば、いろんな計算、プログラムを記述できます。効率はとりあえず考えないんですけれど、

・スライド 35「その他の例」

科学計算とか、行列計算とか、QR 分解とか、いろんな計算の問題、文書の処理とか、最適化問題とか、画像処理とか、いろんな問題をこの方法で解くことが出来ます。

・スライド 36「基本パターンの表現力」

さて実際には、このようなパターンを使うと、どのくらい表現できるのか。
結論から言うと、リスト上の計算可能な関数、計算可能なプログラムだとすると、このような計算のパターンの組み合わせで必ず表現出来るよ、しかも効率的に実現できる形にもっていくことができる、表現出来るよと。これは分離定理というのですが、まあ分離定理はどんなものかはよいとして、このような結論になると。
実際には、もう少しプラクティカルに応用から見ると、最近は Google を皆さんよく使ってると思いま

すが、そこには沢山の計算があります。
ランキングや、サーチングなどいろいろありますが、その中の計算は、Google で最近論文を発表したのですが、実を言いますとその中で全てこのような計算パターン（スライド 36）で、彼らは、mapReduce と言っています。実は、この概念は最近出ているのですが、
このmapReduceの概念はプログラム運算の世界では20年前に出ているんですが、mapしてGroupbyして、reduce すると、ほとんどの Google の計算を表現することができるよということなのです。結構表現力が高い。
ここまでは、これを使えばいろんなプログラムを書くことはできます。
ただし効率的なものになるかどうかは分からないですが、効率的にもっていくためには、方程式を解くための先ほどのような理論が必要なんです。プログラムの運算のための理論が必要なんです。

・スライド 37「効率化：融合変換」
1 つは融合変換。よくこのようなものがあります。
この結果（赤色）はこの入力として（青色）こうやる、こんなことだったら、自動的に 1 つの fold にもっていくことができると。（緑の矢印下の青色）これは融合変換です。
これによって、不必要な途中のデータの引き渡しをなくすことができます。

・スライド 38「効率化：組化変換」
さらにもう 1 つ重要な変換としては、組化変換（くみかへんかん）。組化変換は運算なのですが、同じ数列があって、それに対して、私は要素の和を求めたい。と同時に、要素の最大値を計算したい。そうすると同じデータを 2 回見なければならない。
実は、組化によって、1 つの fold に持っていくことができるんです。
というのはデータを 1 回投げるだけで同時に 2 つ結果を計算することができる。
これが組化変換。このような、いろんなルール、運算の理論を開発しなければいけない。

・スライド 39「連続の部分列和の問題」
先ほどの部分列和のリストの最大値だとすると、これは元々プログラムなのですが、こう書いたら、融合変換と組化変換をやれば、効率的に 1 つの fold にもっていくことが出来るのです。この運算は、今は図で描いていて楽しいです。数学の式の変換をしたい人、運算をもっと知りたい人のために、このスライドを用意しました。

・スライド 40「プログラムの運算」
実際にはこんな感じできれいに、…ちょっときれいかどうかわかりませんが、分かりにくいかもしれませんが、きれいに上から変換していくのです。効率の悪いものから、効率のいいものにもっていくことができる。非常に数式の運算のような感じで、プログラムもこうやっていろんな理論を使って運算していくことが出来るのです。

・スライド 41「最大部分列和問題の賢い解法」
最後に、出来たこのプログラムは言語で書くと、この（スライド 41）のような感じです。
これは、私が皆さんに始めて紹介する賢い解法です。

・スライド 42「最適化のための運算ルールの開発」

これを見ると、有効に使うためには、たくさんの運算の理論、運算のルールが必要なのです。92 年に留学してから、こういうルールの開発をやってきました。たくさん既に開発しています。

基本運算ルールは、融合運算とか、組化運算とか、蓄積運算とか。ある特別な問題に関しては、その運算ルールを開発してまいりました。

そしてその夢のような運算器を目指して頑張って作っています。将来皆さんがプログラムを書けば、自動的に効率の良いものができるような・・・。

・スライド 43「積み木のように並列プログラミング」

時間がないですが、あと少し情報として提供したいのですが、今日の話は逐次的な計算の話でしたが、今、世の中にはたくさん並列計算機があります。

先程の Google も同じですが。私達は並列計算機では、2002 年から 2005 年で並列プログラムは積み木のような感じで作ることができるよということを、「さきがけ」の研究でやってまいりました。成果のビデオがありますので、興味があればご覧下さい。

・スライド 44「もっと勉強したいなら」

そして、もっと知りたい方、積み木で本当にいろんなプログラムが作れるのかと、興味がある方にはこの本をお薦めします。”Algebra of Programming” プログラミングの代数。代数とは小さいものから大きいものをつくるというような意味なんですけれども。

興味があったら、この本を読んでいただければと思います。

そして、東大ではこういうことに関して、講義をずっとやってきまして、資料を作っています。毎年の資料がここに全部ありますので、興味があれば、東大計数工学科の計算プログラムの数理の講義、資料はホームページに公開されています。2008 年までの資料がありますので、興味があれば見てください。

・スライド 45「まとめ」

最後にまとめます。

プログラミングは科学的な活動で、単に技法ではなくて、プログラムをたくさん書けば書くほど正しくていいものが作れるものではなく、しっかりした理論やしっかりした基礎を勉強しなければいけない。

それをまずひとつお伝えしたい。単に動くだけでは、これはプログラムではない。

具体的には Calculational programming という、新しいプログラミングの考え方、作り方を皆さんにご紹介しました。

このプログラミングは、基本計算パターンの構成によってプログラムを作っていく。そして、運算理論、まだたくさん開発しなければなりませんが、それを用いてプログラムを効率化する。

最後にこれを実現するために、いろんなことをやっています。

1 つは、プログラマーですが。

プログラミングの KUMON 教育。

私は日本の KUMON 教育に興味があって、非常に良いと思うのですが、今の KUMON 教育は、基礎学力を身につけます。数学、物理、英語など・・・

プログラミングも同じような感じできちんと KUMON 的な教育を、プログラムをやる人は受けてほしいと

思います。そういう教材は今ないんです。プログラミングだけなら、たくさん本があります。C 言語のプログラミングとか、Java プログラミングとか沢山ありますが、このようなものはまだ無いので、これは夢ですが、そういうものを作っていきたいと思います。計算パターンを勉強したり、ルールを勉強したりと。
そして、コンピュータ科学者としては、そのためにプログラム理論をきちんと構築していく。その多くの有用な運算ルールを開発し、その体系化をやっていく。これがコンピュータ科学者の仕事です。
そのようなものを開発することによって、もっと楽しく、もっとおもしろい、もっと有意義なプログラムを社会に送り出したいと思っています。
以上です。長くなってしまいましたが、ありがとうございました。

・スライド 46「ご清聴ありがとうございました。」
（拍手）