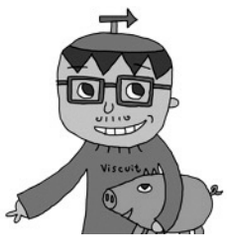
特集 ■インタラクション技術の新展開

万人のためのプログラミング



原田康徳　NTT
HARADA YASUNORI

1. はじめに

ビスケットというビジュアル言語を開発している．この研究の目的は三つある．

1) 子供を中心に，誰でもプログラミングのエッセンスを理解してもらう．
2)「例による書き換え」という新しいプログラミング技術を，様々なメディアに応用する．
3) コンピュータを本質的に創造的な表現手段として捉え，その基礎技術を確立する．

現在の研究ステージは 1) の途中くらいである．2)，3) は未着手ではあるが，本解説で，これまでの到達点と，方向性について整理したい．

2. プログラミング教育

そもそも，子供にプログラミングを教える必要があるのだろうか．プログラミングなどは専門家がやればいいことで，一般ユーザには使いやすいコンピュータを提供していればよい，という考え方もある．

2008 年春に公開された文部科学省の新しい学習指導要領によると，技術家庭の「D 情報に関する技術」において，

(3) プログラムによる計測・制御について，次の事項を指導する．
　ア　コンピュータを利用した計測・制御の基本的な仕組みを知ること．
　イ　情報処理の手順を考え，簡単なプログラムが作成できること．

という項目が追加された．子供にプログラミングを教えたいという立場からすると，このような決定は一つの追い風ではある．しかし，国が決めたからやるという消極的な理由ではなく，我々はもっと積極的な意見として，プログラミングを教える必要性を感じている．

インターネット／コンピュータが広く普及し，小学生でも使いこなす時代である．しかし，まだ発展途上のインターネットやコンピュータには，非常に多くの危険が存在する．そこで，ネチケット，情報モラル教育といった，ルールやお作法が盛んに教育されている．しかし，ルールを闇雲に丸暗記するのではなく，コンピュータに対する直感を身につけて，その結果ルールが必要なんだという理解をするべきだと考える．例えば，自転車を例にしよう．自転車は便利な乗り物だが，乗り方によっては危険でもある．例えば急な下り坂ではブレーキが利かないかもしれない．スピードを出しすぎたら急には止まれないかもしれない．その危険との境界線を認識しているのは，我々が運動方程式を理解しているからではなく，ましてや自転車を乗るときのルールとして暗記しているからでもない．物理世界への直感を持っているから，どこまでが安全でどこからが危険かという線引きができるのである．もし物理世界への直感がなければ，どうなるだろう．20 度の下り坂では乗るのは危険です，曲がりたいときは 15m 先からブレーキをかけましょう，ただし，速度が 15km/ 時以上のときは 20m 先からブレーキをかけましょう，といった，細かなルールを沢山暗記しなければ怪我をする人が続出するだろう．

コンピュータやインターネットの一つの欠点は目に見えないことである．そのため，仕組みを理解するには，数学の理解と同様に抽象的な物事を理解する能力が必要である．そのため，直感を持てている人は非常に少ない．しかも，まだ技術的に発展途上である．これはブレーキがほとんど効かない自転車のようなものである. 安全に使うには，非常に多くのルールを暗記しないといけないだろう．

一般ユーザにとって，コンピュータは便利ではあるけれど，中身のまったくわからないことへの恐怖感はない

のだろうか．コンピュータに対する直感は，プログラミングをして，コンピュータを自分で制御する，という経験で身につくだろう．積み木遊びで物理世界への直感を身につけて行くように，コンピュータによるプログラミング遊びで直感が身につけばよいと考える．コンピュータ，インターネットを安全に使うためにはまだまだルールに頼らなければならないだろうけれど，このような直感があると非常にルールも教えやすいだろう (そんなに堅苦しい理由だけではなく，プログラミングがこれほど楽しいものである，という秘密を，誰にでも味わってもらいたい，という理由もある).

前置きが非常に長くなったが，それがビスケットというビジュアル言語を開発している直接的な動機である．

3. ビスケット

ビスケットの説明をする．しかし，紙面による説明には限界があるので，ぜひビスケットのサイトをご覧になっていただきたい．

図 1 はもっとも基本的なプログラムの例である．二つの円の並びは，ルールというプログラミングの単位である．これは，左辺の絵を右辺の絵に書き換える，という意味である．この例で，左辺と右辺の絵を比較すると，車が少し前にずれている．ビスケットでは一定間隔でこのルールを実行するので，これによって車が前進する．

図 2 はこのバリエーションで，ルールの右側の車が斜めになっている．その結果車は回転する．ここまでは，単純に変化の可視化であるから，それほど難しい話ではない．

図 3 は，三つのルールからなる．一つは図 1 と同じ車が前進するルール，他の二つは車が人にぶつかりそうになったら避ける，というルールである．システムはアニメーションの各ステップにおいて，この三つのルールの中からもっとも適したルールを選択して，それを実行する．このとき，厳密な絵の配置のマッチングを行うのではなく，並び方が大体近ければマッチする．また，選択されたルールの実行も，その並び方の近さに応じて変形させる．すなわちルールとほぼ同じ配置ならば，結果もほぼ同じ結果になるのに対し，かなり配置は違っているが，他のルールよりも配置が近いという場合には，そのずれに応じた変形になる．

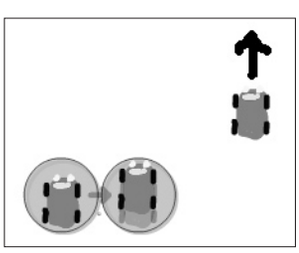


図 1

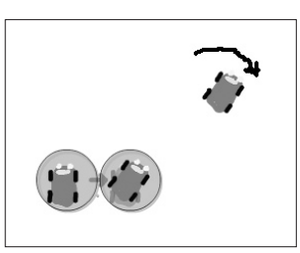


図 2

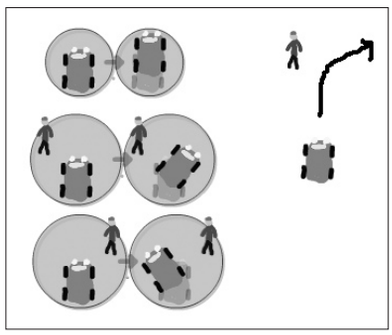


図 3

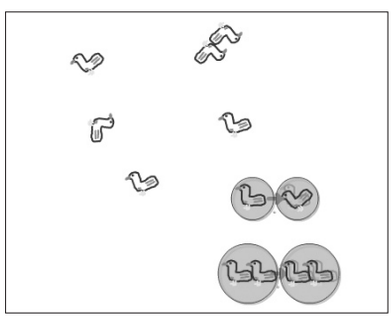


図 4

図 4 は，アヒルの行列である．上のルールでアヒルは同じ場所をぐるぐると回っている．何かのタイミングでアヒルが 1 列に並んだとき，下のルールが選択される．このルールは，アヒルが 2 羽並ぶと前身する，という意味である．隊列を組んだアヒルは画面内を歩き回り，他のアヒルに近づくごとにそれを隊列に引き込む．長い時間実行を続けると，最終的には全体が一つの列になる．

以上の例は，ルールの左右で絵の個数が変化しないものだった．しかし次の例 (図 5) では，1 人が 5 人に変化する，というルールである．これを実行すると，人の数が指数関数的に増大する．非常に危険なルールである．

今のシステムでは，このように絵の個数が変化するタイプのルールは，特殊なモードでのみ実行を許すことにしている．それは，ユーザが簡単に間違えてしまって，すぐに画面が絵でいっぱいになってしまうからである．しかし，この例は非常に教育的でもある．これはチェーンメールはなぜいけないことか，というのを直接説明している．一人の人がメールを 5 人に送ることを，みんながいっせいに続ければ，すごい勢いでメールが増えていってしまう．我々が普段，直感的に体験している「物が増える」という感覚は，ほとんどの場合，線形な範囲である．指数関数的な増え方は，一般人の想像をはるかに超える速度なのである．

図 4 程度までの，ビスケットのプログラミングを体験

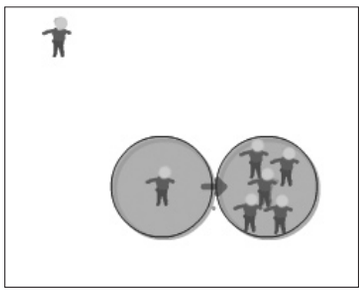


図 5

した子供ならば，図 5 のプログラムの意味はすぐに理解できよう．ここが，重要な点である．指数関数の計算やグラフによる理解は数学の基礎が必要であり，多くの子供が落ちこぼれる．しかし，ビスケットを使えば，そのような難しい基礎なしに，重要な考え方を理解させることができるのである．

4. 例による書き換え

ビスケットの中心技術を「例による書き換え」と呼んでいる．動きの実例を与えて，それを状況に応じて拡大解釈し，動かす．得られた動きが気に入らなければ，さらに実例を追加し，動きを修正する．ここでは，「例による書き換え」を一般化してその応用を試みる．例として，図 6 のようなグラフが与えられたとき，その計算をするプログラムを考えてみよう．

従来のプログラミングでは，このグラフを計算する式を

y = x*x;

のように直接与える．これは対象のモデルと計算式が明確に定義できる場合に有効で，計算結果も正確である．それに対して，「例によるプログラミング」という技術がある．これは，

1->1

2->4

3->9

という入力と出力の組を与えて，y=x*x という式を推測

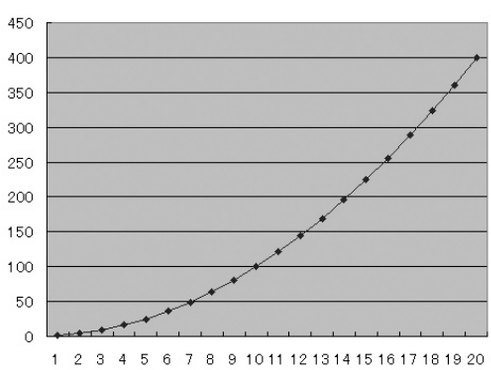


図 6

する技術である．これは計算式が，この技術によって推測可能な範囲であれば成功する．複雑な式や，そもそも式で正確に表現できないような場合は難しい．

一方でビスケットの原理である「例による書き換え」は y=x*x というプログラムの表現形式を求めない．その代わりに，ここで欲しい計算をそのつど直接実行してしまうのである．

例で説明しよう．最初に

1->1

という計算例を与える．これに対して，

2->?

という質問を出すと，システムは今の知識で値を推測して，たぶん 2 という答えを返すだろう．この答えは求めているものとは違うので，

2->4

という新しい計算例を教える．システムには二つの例が与えられて，入力が 1 や 2 の周辺では求めているものに近い答えが返るだろう．それ以外の入力では，システムの内部モデルにも依存するが，2 点を通る直線というモデル化 P して，それにしたがって答えを返すかもしれない．それで次の実行，

3->?

では 7 を返してしまう．この値も求めているものとは違うので，計算例として

3->9

を新たに与える．システムのモデルが，2 次式の場合であれば，これらの知識で完全な解に到達できる．

しかし，内部モデルが 1 次式でしか推測しない場合でも，例が二つのときよりはよい振る舞いをするだろう．例えば，2.9 の入力に対して，入力が最も近い二つの計算例 2->4, 3->9 を選ぶ．この 2 点を通る直線として 2.9 の入力に対して，8.5 を得る．求めている答えは 8.41 であるから，まあ近い．

重要な点は内部のモデルの正確さではなく，与えた例の周辺ではいい答えを返すということである. この例では，我々が精密な取り扱いに慣れている数値を用いたので，精密なモデルを使うほうが良いように思える．しかし，ビスケットのような図形の配置書き換えでは精密さはそれほど必要としていない．実は，ビスケットの内部モデルは 1 次式ですらない．それでも十分な動きをするのが面白い．

ビスケットの絵の配置に対する書き換えを説明する．簡単のため，2 枚の絵がルールの左右にそれぞれ入れられている場合を考える．1 枚の絵は位置と角度による 3 自由度，2 枚の絵の場合には 6 自由度，しかし，マッチングでは平行移動を無視するので 2 自由度減って，2 枚の絵の配置は 4 自由度である．ルールは 4 次元空間の 1 点から，別の 4 次元空間への 1 点への写像と考えること

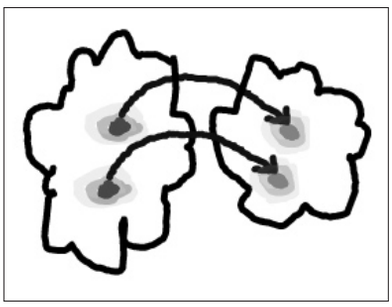


図 7　＊口絵にカラー版掲載

ができる. 書き換えられる対象も同じ4次元空間の1点で, 左辺の点の周辺を右辺の点の周辺に写像される (図 7). この技術は連続的な値を持つ対象であれば，何にでも適用できる．例えば，ロボットにダンスを教えることを考える. ロボットの自由度がNであれば, ロボットのあるポーズは N 次元空間の 1 点として捉えられる．ポーズの変化はN次元からN次元への写像である. いくつかのポーズの変化を教えることで，ポーズの時系列 (ダンス) を教えることができる．関節の時間変化 (角速度) も考慮し，2N 次元空間の 1 点として捉えると，より滑らかなダンスが得られるだろう．対象が多少離散的であっても可能である．ピアノにフレーズの例を教えて，延々と新しいフレーズを生成させよう．時間的につながった二つのフレーズを複数個システムに教える．1 曲まるまる与えて，1 小節を一つの単位として，つながった 2 小節をフレーズの組としても良いかもしれない．フレーズに対して，フレーズ間の距離と生成の計算方法が定義されているとしよう．生成の計算方法とは，A->B という変化があったとき，A に近い A' を与えたとき B に近い B' を求めることである．単純な方法では，フレーズ間の距離は音程の時間的変化の違いと，絶対的な音程の違いをミックスしたもの，生成の計算は，各時間ごとのずれをそのまま適用したもので良いだろう．例えば A' は A と比べて 3 拍目が半音高ければ，B' は B の 3 拍目を半音高くする，ということである．このようにして，ある程度複雑な例を与えたら，延々と新しいフレーズを生成し続けるピアノができる.

以上，ビスケットの基本的な動作原理である「例による書き換え」の中身とその応用の可能性について述べてきた．これらの応用に関しては, 可能性は感じられるが, まだ，どれも実装してはいない．構想のみである.

5. 創造的な表現手段としてのコンピュータ

さらに，構想の範囲を超え，妄想に近い話をお許し願いたい.

油絵の具や，写真の発明と同様に，コンピュータは新しい表現のメディアとしても発展してきた．これまでにもコンピュータを使った芸術作品が数多く見られるものの，まだ従来のメディアの延長線上でしか捉えられていないように思われる．専門家だけでなく，一般ユーザにとってもコンピュータによる新しい表現の追求はまったく手付かずである.

ビスケットは，動く絵本，アニメーションを簡単に作ることができるツール，として認知されている面もある．ある特定のタイプのアニメーションに関して，ビスケットが非常に手軽に作成できることは事実である．しかし，アニメーションという従来から存在している表現の視点から見ただけでは，ビスケットの特徴は捉えきれていない．プログラミングの楽しさという視点がまったく伝わっていない.

粘土はどんな形でも作ることができる．しかし粘土で作った歯車では大きな機械は作れない．強度的, 精度的, 物性的に様々な問題があるからである. コンピュータは, 対象を限定するけれども，プログラミングによって作られたものはそのまま完璧にその対象内で動作する．素材としての性質は申し分ないのだけれど，粘土のように自由自在に形を作れるようなインタフェース，言語機構はまだ発明されていない.

絵を描く，写真を撮る，日記を書く，といった創造的な活動と同様にプログラミングが捉えられる日がくるであろう．プログラミングの定義，解釈は今よりもさらに広く捉えられ,「例による書き換え」の他にも様々なプログラミング技術が発明されるだろう．それらによって，人間の創造性はさらに大きく発揮できることを願っている.

参考文献

[1] 文部科学省：新しい学習指導要領

[2]http://www.mext.go.jp/a_menu/shotou/new-cs/index.htm
Yasunori Harada, Richard Potter : Fuzzy Rewriting -- Soft Program Semantics for Children --,HCC 2003, IEEE. (2003)

[3] 原田康徳，加藤美由紀，Richard Potter : Viscuit: 柔軟な動作をするビジュアル言語，WISS 2003.(2003/12/4)

【略歴】

原田康徳（HARADA Yasunori）
日本電信電話株式会社
NTT コミュニケーション科学基礎研究所　主任研究員
1963 年生まれ．1992 年 北海道大学大学院 工学研究科情報工学専攻博士後期課程卒業，　博士 (工学). 同年日本電信電話株式会社 NTT 基礎研究所入社．1998 年～2001 年科学技術振興事業団さきがけ研究員．2004 年～2006 年 IPA 未踏ソフトウェア創造事業プロジェクトマネージャ．2000 年より現職.