ビスケットへの絵の継承の導入

Viscuit with Picture Inheritance

原田　康徳†

ビスケットは柔軟なマッチングを導入した，図形配置書き換え型ビジュアル言語である．書き換えは一定間隔で連続して行われるため，図形配置が連続して変化し，実行結果はアニメーションとなる．柔軟なマッチングにより，図形の配置が厳密に一致していない場合でも柔軟に書き換える．その結果，単純な書き換え規則でも表現力の高いアニメーションが得られる．そのビスケットに，新たに図形の継承を導入する．図形Aが図形Bを継承することで，図形Bとして定義されていた書き換え規則はすべて図形Aとしても使用できるようになる．実装は，書き換え規則を図形Bから図形Aに置き換えたものを自動生成することで，実行される．単純で強力な実装の結果，グループを用いた継承，多重継承，条件付継承などを自然に表現できるようになる．

1．はじめに

ビスケット1)は，子どもやコンピュータの専門家ではない人に，コンピュータプログラミングの楽しさを体験してもらうために作られた，ビジュアルプログラミング言語である．文部科学省のサイトに登場した「プログラミン」というビジュアルプログラミング言語があるが，単にプログラミングそのものを楽しく経験できればよいというだけでなく，コンピュータが持つ底知れない力を直感的に感じてもらえるようにビスケットは設計されている．そのため，非常に少ないプリミティブしかなく，その組み合わせ方の工夫で様々な応用ができる，という点を重視している．

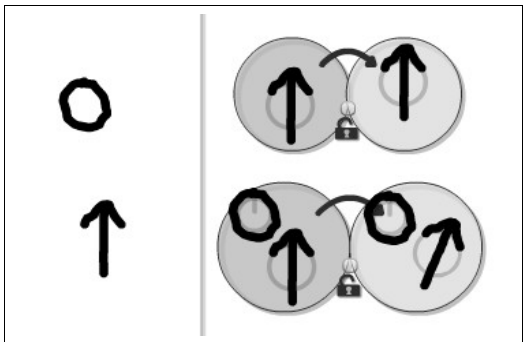

図1ビスケットの書き換え規則(めがね)と実行例

ビスケットの例を示す．図1は右側が書き換え規則，左側が書き換え対象である．書き換え規則はその形状から「めがね」と呼ばれている．めがねは左側にある絵を右側の絵に書き換える規則を示している．絵の種類だけじゃなく，めがねの中の絵の相対的な配置も考慮して書き変えられる．図の上のメガネは，矢印が上にずれるように書き換える規則である．書き換えは一定間隔で連続して行われるため，矢印はまっすぐ進むアニメーションになる．下のメガネは，矢印が丸にぶつかりそうになると，よける，という書き換えである．複数の図形があるとき，それらの配置は厳密に一致しなくても，その配置のずれを考慮して書き換えが行われる．相対的な配置が厳密に一致した場合は指示された通りに，相対的な配置がずれている場合はそのずれが大きいほど，書き換えの結果も指示されたものからずれる．

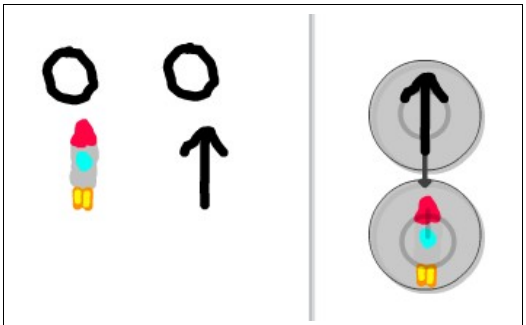

図２絵の継承（ゆきだるま）

2. ビスケットの継承

新たに導入した継承は縦に並んだ2つの円で表現する(図2)．我々はこれを，「ゆきだるま」と呼んでいる．下の円に入れられた絵は上の円に入れられた絵と同じ動きをする．ロケットは矢印の動きを継承する．そこでロケットは，矢印と同じようにまっすぐすすんで，丸にぶつかりそうになるとよける動きをする．

ゆきだるまによる継承の指示は，単に絵の種類の情報だけでなく，絵の相対的な位置関係も重要である．

† NTTコミュニケーション科学基礎研究所

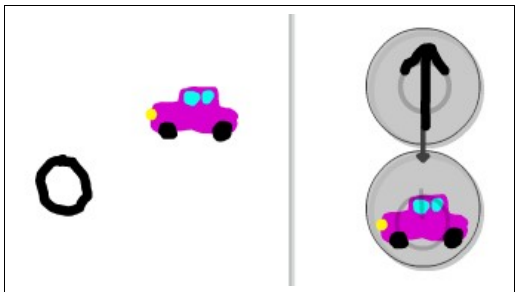

図3継承で相対的配置が影響する例(間違い)

図3のようなゆきだるまは，思ったような車の動きをしてくれない．本当は車は絵の進行方向(左)に進んでほしいが，車は上にすすんでしまう．ビスケットは絵の持っている本来の意味は知らない．車の絵がどっちに進むべきかは自動的にわかるはずはなく，ロケットの絵と同じように上に動いてしまうのである．

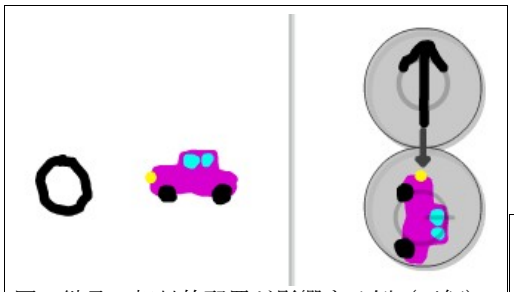

図４継承で相対的配置が影響する例（正解）

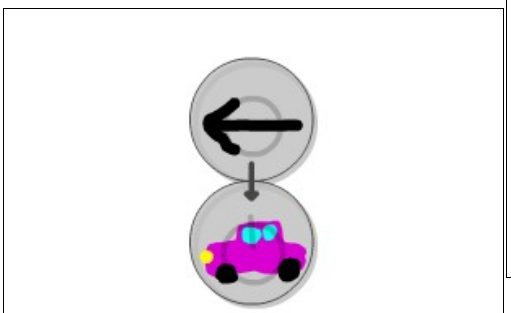

図5継承で相対的配置が影響する例(別の正解)

正しく車を動かしたい場合には，図4のように，ゆきだるまの中の車を上向きに置く．これによって，車は進行方向に進むようになるし，丸にぶつかってもよけるようになる．

ゆきだるまで重要な情報は，絵の相対的な配置であるので，図５のように，矢印のほうを左に向けてゆきだるまを作っても良い．むしろこちらのほうがわかりやすいかもしれない．

3. 絵のグループ化

複数の絵を特定に配置させたものを一つの絵として扱いたい場合がある．これを継承で表現する．

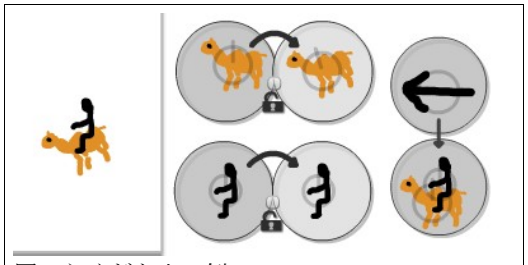

図６らくだと人の例

図6では2つの絵，らくだと人に対してめがねが定義されている（図中央の２つのめがね）．それぞれ，らくだは後ろ向きに回転する．人はゆっくりと前にすすむ，という意味である．図右のゆきだるまでは，らくだに人が乗った状態で，矢印と同じように動く，と定義されている．人とらくだが単独でいるときは，それぞれの動きをするが，何かの偶然でそれらの絵が重なると，まっすぐ進みだし，障害物に当たったらよけるようになる．

4. 継承の実装

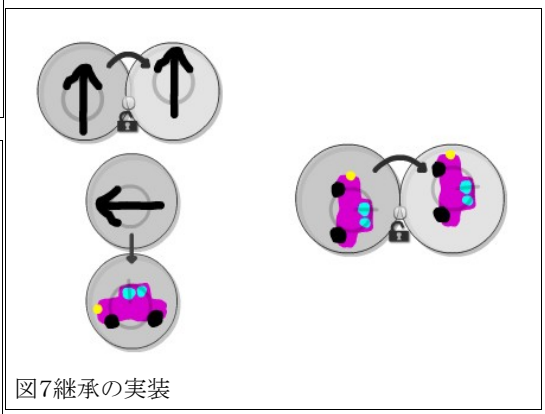

図7継承の実装

図7の左側のようなめがねとゆきだるまがあったとき，システムは図右のようなめがねを自動生成する．生成方法は以下のとおり．

1)すべてのゆきだるまに対して，
2)ゆきだるまの上にある絵Ａを含んでいるすべてのめがねを集める．
3)そのめがねの左右それぞれに対して，絵Ａをゆきだるまの下にある絵の組で置き換えた新しいめがねを作成し追加する．
4)追加されたためがねに対して，1)から3)を実行し，新たに追加されるめがねが無くなるまで繰り返す．

継承が多段であっても，このアルゴリズムは動作する．

ビスケットの実装では，何かのめがねかゆきだるまを修正するごとに，この自動生成を行っている．ただし，非常に無謀な継承をされた場合に備えて，自動生成できる個数に上限を設けている．上限に達するような自動生成が行われる場合というのは，あまりちゃんと設計されておらず，グチャグチャに使われていることがほとんどなので，生成が不完全であっても，その動作がおかしいと気づくことはまずない．完全な動作よりは，グチャグチャな操作でも重くならずに動作する，という点を重視している．

5. 継承の応用

5. 1 多重継承

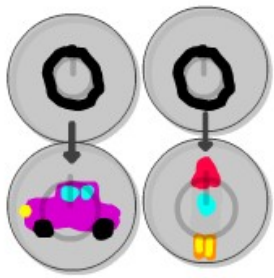

図8多重継承の例

車とロケットは，丸（障害物）を継承する．図1，図2，図4とこの例をあわせると，車やロケットは矢印のように動くだけでなく，障害物としても認識され，車やロケットに当たるとよけるという動作もする．この2つの多重継承の結果，組み合わせ的に非常に多くのめがねが自動生成される．矢印が丸にぶつかる，というめがねを種にして，矢印と丸をそれぞれロケットや車などの絵に置き換えた組み合わせで生成する．車とロケットが継承しただけで 3x3 で9個のめがね（つまり8つ生成）になる．

多重継承など，複雑な継承を行うと，生成されたためがねに矛盾が生じる可能性がある．しかし，ビスケットの処理系は書き換えの都度，もっともマッチングの評価関数が高いものを選択して書き換えるので，矛盾があったとしても問題はない．

5. 2 センサーとアクション

ビスケットの機能として，キーバッドを使った制御の機能がある．図9のめがね（図上）はキーパッド（左ボタン）を押している間だけ有効で，矢印が回転するという意味である．

図9のゆきだるま（図右）は，1ボタンを押している間，

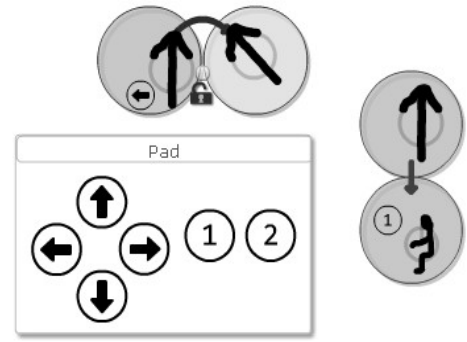

図9キーパッドの例

人は矢印の動きを継承するという意味になる．実装では，矢印が含まれているめがねを，人と1ボタンの組で置き換えるだけである．ボタンなどのセンサーはめがねの左側（条件側）に入れることになっている．この単純な置き換えによる実装では，めがねの右側（アクション側）にある矢印も1ボタンと人に置き換えてしまうため，右側にもセンサー命令を入れてしまうことになるが，無視される．

同様に，音を出すなどの，めがねの右側（アクション側）に入れる特殊命令がある．これを継承に含めることは可能である．この場合もめがねの左側にも入れられてしまう可能性があるがそれは無視される．

現在のビスケットのセンサーとしては，まだキーバッドくらいしかないが，光センサーといった物理的なセンサーや，Web 上のリソースを用いた条件式（例：天気が晴れなら）などもセンサーとして拡張可能である．これらのセンサーによる条件が真の時だけ継承が有効というのは，非常に面白い応用が可能であろう．

6. 議論

これが子どもに受け入れられるかどうか．我々が，子どもにビスケットを教えるときの基本的な方針は，彼らが必要とするまでは与えない，ということである．ただし 100% 求められるまで教えないやりかただと，進歩が非常に遅いので，新しいことを少しだけみせることは見せるが．それもあまり急激なジャンプはさせないように気をつけている．時間の制約などで，無理やり難しいことを教える場合もあるが，大体失敗している．その場は真似はするけど，彼らが必要としていないことであれば，すぐに忘れてしまうからである．

そういった方針からすると，この「ゆきだるま」は，ある程度複雑なプログラムを作った子が，それをもっとすっきりとさせたいという要求が出たときに教えられる，ということになる．

6. 1 子どもが作った例

たとえば，図10のようなシューティングゲームを作っ

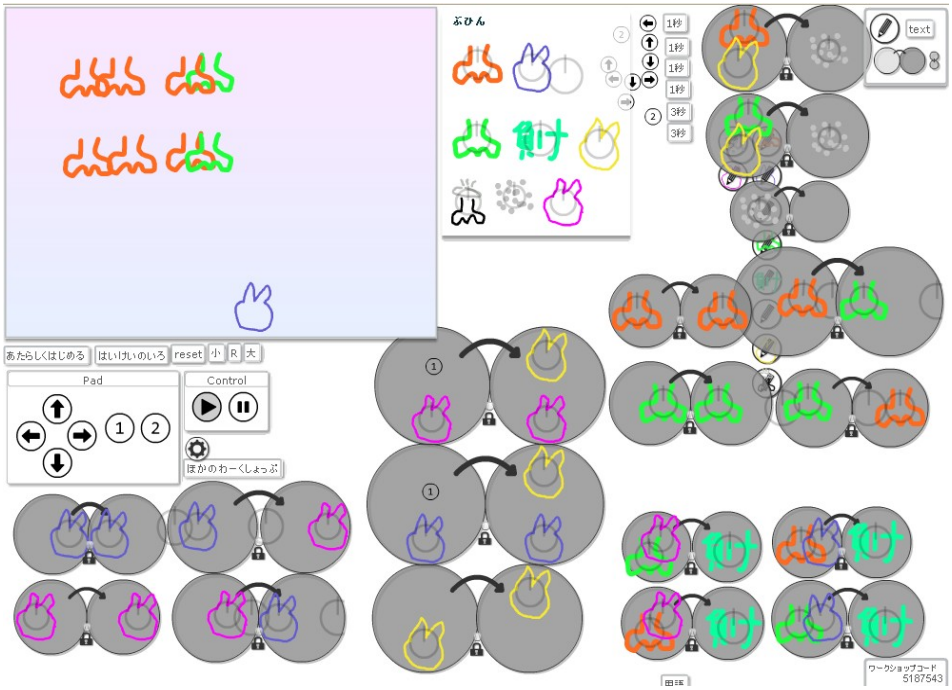

図10　中2が作成したシューティングゲーム(敵と砲は2色で，右か左に動いて見えない壁にぶつかると，反対の色になる．これによって左右に往復する．砲は1ボタンで弾を発射し，敵は弾に当たると消える．敵と砲がぶつかると「負け」になる)

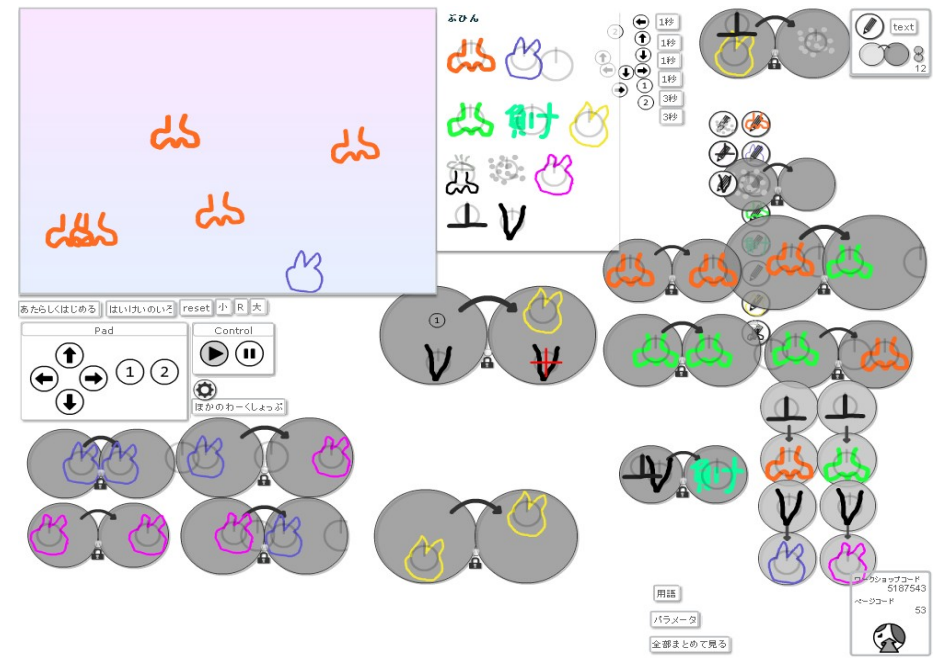

図11　著者が改良した継承を利用したシューティングゲーム

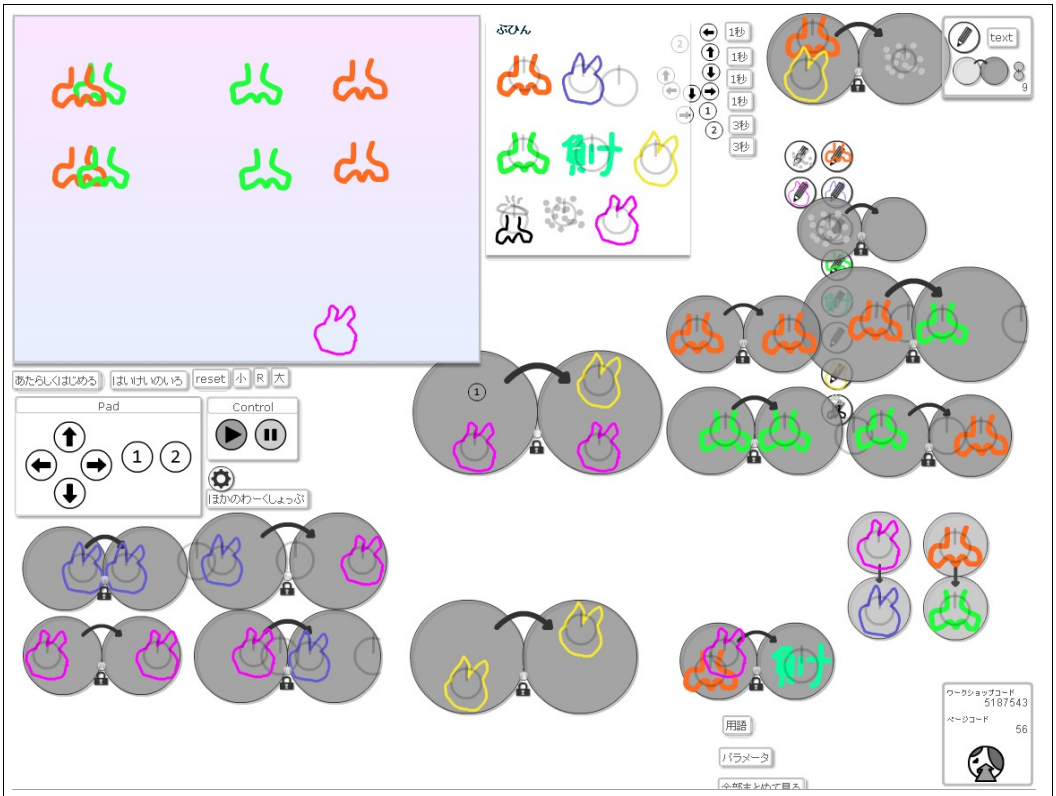

図12抽象クラスを使わない方法（うまく動かないかもしれない）

た中2の子がいた．敵の絵は2種類，砲の絵も2種類あり，それぞれ左右に自動的に動く．ボタン1を押すことで弾が発射し，敵を倒す．敵に砲がぶつかると終わりである．最初，自分で砲を左右にコントロールできるようにしていたが，あまりにもゲームが簡単すぎたので，砲の動きも自動化して，弾を発射するタイミングだけコントロールできるように，制約をもたせてゲームの難易度を上げている．

砲と敵が左右に動くとき，絵の色を状態として作っている．画面の左右には，透明な壁の絵を置き，そこにぶつかると方向を変えるという仕掛けがある．弾を発射，弾と敵が当たると爆発，敵と砲が当たると負け，の部分は，動いている方向に関係ない処理であるが，状態を色で示しているため，プログラムを見てわかるように，似たようなめがねが並んでいる．

これくらいなプログラムを作った子にゆきだるまの機能を提供できれば，プログラムをすっきりとさせられるだけでなく，より複雑な継承の応用に発展したであろう．この時点ではゆきだるまはまだ実装されていなかった．

図11は抽象クラスを用いてゆきだるまを使用した，著者が改良したプログラムである．ここで，敵と砲は抽象的な図形に対して動きを定義し，それを継承するようにしている．これは図10と同じ動作をする．

しかし，抽象クラスを用いずに，敵や砲のどちらかの色で主の動きを定義し，それをもう一方の色が継承するようにするとどうなるだろうか（図12）．青の砲は左に，ピンクの砲は右に動くようになっている．しかし，たとえば，青の砲で全体の動きをつくり，ピンクの砲がそれを継承するようにすると，ピンクの移動のめがねは左に動くものと，右に動くものの両方ができてしまう．2つのめがねの左側の絵が同じものの場合，それぞれの絵の絶対角度が異なると，書き換え対象の絵の絶対角度と比較して，角度が近いほうのめがねが使用される．角度がまったく同じであれば，どちらのめがねが実行されるかわからない．図12は，元のプログラムの角度がしっかりと作られていたため，自動生成されためがねの角度も正確に同じ向きになり，正しく動くかどうかわからない．

一般的なオブジェクト指向では，スーパークラスのメソッドをサブクラスで上書き(override)することで，スーパークラスの動作を完全に消すことができる．しかしビスケットの継承では，すべて残して継承してしまうた

め，矛盾した動きも残ってしまい，このような問題が生じる．つまり，いまの実装では，図11をつくらなければダメなのである．このような，図11と図12の違いを子どもが理解して使用できるとは思えない．

この問題を解決するために，すでに継承前の絵に対してめがねが定義されているときには，継承によって自動生成されるめがねに制約を与える，という仕様がよいのかもしれない．つまり，生成しようと思っているめがねの左側のパターンと近いめがねがすでに定義されていたら，生成しないというものである．ここで「近い」という点がビスケットらしい仕様だと思う．

６．２　再帰的構造の定義

以下は，今回のビスケットの拡張では禁止されているが，それを許すとしたら，という仮定の話である．

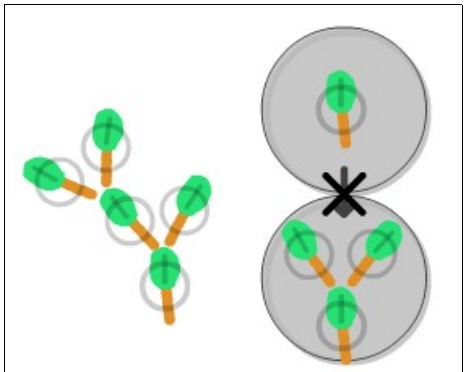

図１３再帰的な形の定義

図１３は，継承で上と下で同じ絵が使われている例である．現状のビスケットでは継承の木で循環は禁止しているので，それが生じた場合はこの図のように×のマークが示される．ここでは仮にこの循環を許したとしよう．この例は，木を３つ下の円のように配置した図形は，一つの木と同じである，と定義している．再帰的な図形配置である．図の左側にあるような配置をした木のグループは一つの木と同じ動きをする．

現状の実装では，継承によってめがねをあらかじめ自動生成して，それから実行する，という方法をとっているため，このような循環した定義は自動生成を停止させることができない．

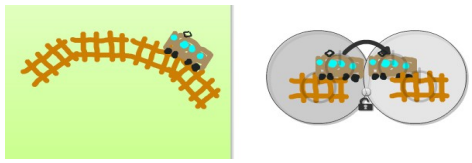

図14　線路と電車の例

しかし，マッチングは有限であるため，生成を事前に行うのではなく，書き換えのタイミングで継承をたどりながらパターンマッチをするような実装ができれば，循環を許した継承の実装は可能である．

これは，形式文法の２次元版に発展する．

6．3　絵の対象性の定義

図14のような線路と電車の例がある．一般的な印象として線路はどちらの向きでも電車を走らせることができると思うだろう．しかしこの例では線路には方向があるので，反対向きに電車を置いても進まない．そのためには，線路の反対向きでも電車が進むというめがねが必要である．

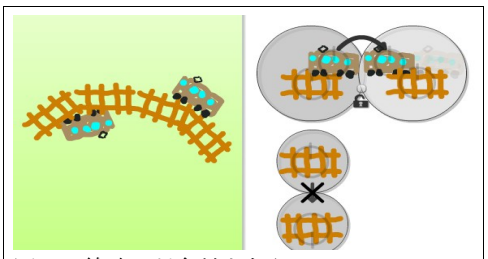

図15　線路に対象性を与える

図15のように正確に180度に回転させた線路を継承させることができるようにする．これは，線路は反対向きでも同じ，という意味になる．重要な点は線路が正確に180度に回転していることである．

この論文の執筆時点では，実際のワークショップの現場で，まだ子どもに対してゆきだるまを教えてはいない．かなりレベルの高い子でも，ゆきだるまが必要性を理解できるようになるためには，時間が必要である．まずは，すべての組み合わせが衝突するケースを抜けがなく列挙できるようになって，それを完璧にりかいできるようになってから，ゆきだるまを教えるべきであろう．

図9のような条件付きの継承は2)3)の影響を受けている．

まとめ

新しく書き換え規則を自動生成して動作する継承について述べた．

参考文献

1) Yasunori Harada, Richard Potter : Fuzzy Rewriting -- Soft Program Semantics for Children --, HCC 2003, IEEE.

2) Yasunori Harada, Kenichi Yamazaki, Richard Potter: CCC: User-Defined Object Structure in C. ECOOP 2001: 118-129

3) **原田康徳，山崎憲一：** CCC **--データの内部表現に依存しないオブジェクト指向，情報処理学会論文誌：プログラミング**, Vol.42, No.SIG2(PRO 9), pp. 48-60 (2001).