Vol.2012-SE-175 No.7
2012/3/15

レゴプログラム向け 状態遷移図設計支援システム

今井康平† 紫合治††

本論文では状態遷移図に対して状態指向という概念の提案を行った．そして，その状態遷移図の有用性を評価していくために，レゴのプログラミングに状態指向の状態遷移図の概念を取り入れた状態遷移図設計支援システムを提案・開発した．状態遷移図を作成することでそれに見合った C コードを生成することができるシステムである．そのシステムと既存のレゴプログラミングシステムの ROBOLAB を比較するための実験を行った．

State Transition Diagram Design Support System for LEGO Program

Kohei Imai† and Osamu Shigo††

This paper proposes a concept of a state oriented state transition diagram to describe a detailed status of each state in the diagram. To evaluate the usefulness of the state oriented concept, a state transition diagram design support system which applies a concept of state oriented diagram to Lego programming has been developed. The system automatically generates C source code of Lego system from the state transition diagram. Also, to evaluate an applicability of the system, the experiment for comparing the developed system with the ROBOLAB, which is standard Lego programming environment, was conducted.

1. はじめに

安全な組込みシステムを開発するためには事前にシステムの可能な振る舞いを十分に詳細にわたって把握しておくことが必要になってくる．従来，その振る舞いの設計には状態遷移図が多く用いられている．随って，誤りのない状態遷移図を作成することが安全な組込みシステムを作成する上で重要になってくる．

現在, 状態遷移図を書く際に UML の状態遷移図が幅広く活用されている．しかし，UML で定義されている状態遷移図では状態が多くなればなるほど, どのような入力の経緯によってその状態に遷移したか，また，その状態でシステムの出力等がどうなっているか把握が困難になってしまう．このようなことが原因で，システムの振る舞いを誤解して認識してしまうことが考えられ，システムの開発段階での誤りを誘発してしまう可能性がある．

本論文では，状態遷移図に状態指向という概念を取り入れ，状態遷移図を視覚的に見やすくし，より状態の把握が容易にできるようにした．その状態遷移図の有用性を確認するため，状態指向の状態遷移図の概念をレゴのプログラミングに取り入れたシステムを開発した．そして，そのシステムと既存のレゴ用プログラミングシステムの ROBOLAB を比較し評価を行った．

2. 状態指向の状態遷移図

本論文では，状態遷移図を「遷移指向の状態遷移図」と「状態指向の状態遷移図」の２つに大別している．遷移指向の状態遷移図は，UML で用いられている状態遷移図で，遷移に対して入力や出力のイベントを記述することができ，状態に対しては状態名以外記述しない. 対して, 状態指向の状態遷移図では遷移に対して何も記述しない．必要な情報は状態に対して記述される．入力を条件として扱い，条件として記述された入力を満たしている間はその状態を維持するというように解釈をする．そして条件を満たさなくなった時に状態遷移が起こる，といった状態遷移図になる．

2.1 遷移指向の状態遷移図

遷移指向の状態遷移図に分類されるものとして, UML の状態遷移図やミーリマシン, ムーアマシン等は遷移に対して入力や出力が記述されるため，それらは全て遷移指向の状態遷移図に分類する．遷移指向の状態遷移図は，遷移に記述されている入力が起きた時，遷移に記述されている出力を起こし遷移先の状態に遷移する，という考え方

† 東京電機大学大学院　情報環境学研究科
Graduate School of information Environment, Tokyo Denki University.
†† 東京電機大学　情報環境学部
School of Information Environment, Tokyo Denki University.

である．この状態遷移図では状態が多くなるにつれて，その状態にはどのような入力が起きて遷移したのか，どういった出力がされている状態なのか，といったことを把握するのが困難である．

2.2 状態指向の状態遷移図

状態指向の状態遷移図の考え方は従来の状態遷移図とは全く異なる．各状態に対して記述された条件を満たしている間はその状態を維持する．その状態の条件を満たさなくなった時に状態遷移が起こる．遷移先はその状態から派生している遷移の中でその時の条件を満たしている状態になる．状態を見ることで，その状態にはどういう条件の時に遷移するのか，その状態ではどういった出力が起きているのか一目で見ることができる．

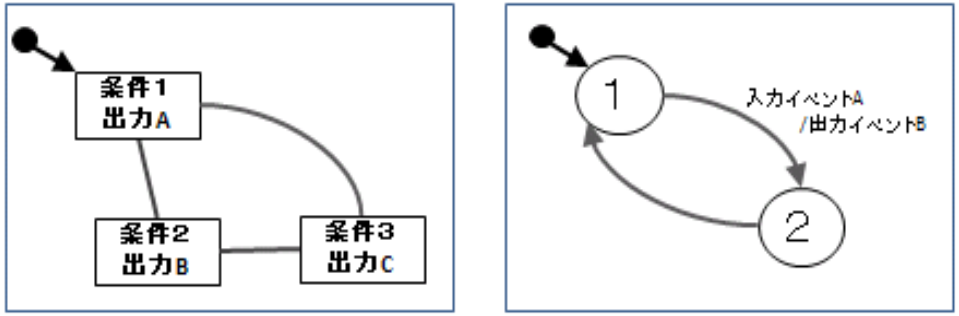

図 1　状態指向の状態遷移図(左)と遷移指向の状態遷移図(右)

3. レゴプログラミング

レゴのプログラミングはプログラミングの導入教育として用いられることも多い．NXT と呼ばれる本体部分にプログラムをアップロードし，プログラムに見合ったセンサとモータを NXT に接続させることでプログラムを実行することができる．

NXT にプログラムをアップロードする方法として，既存のシステム ROBOLAB を用いてアップロードする方法か C 言語で直接的にプログラムする方法がある．ROBOLAB はフローチャート形式でアイコンを並べるだけでプログラムができる．なので，初心者でも簡単にプログラムをすることができる．

しかし，ROBOLAB はプログラムが長くなると，その時はどういった入力がされてどういった出力が起きているのか，を把握するのが困難である．この問題は遷移指向の状態遷移図と同様の問題で，情報が多くなるほど理解が困難になってしまう．

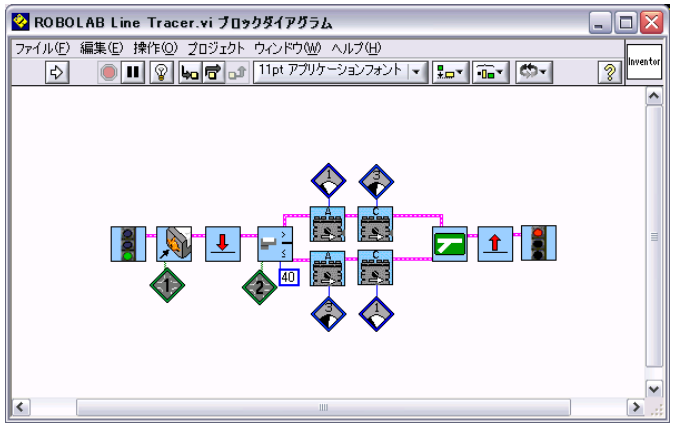

図 2　ROBOLAB によるプログラム例

図 2 のプログラムはライントレースを ROBOLAB により作成した図である．青信号がプログラムの始点を表し，赤信号が終点を表していて，その間にアイコンを並べてプログラムを記述するようになっている．ROBOLAB ではセンサのアイコンを使用する際にどのポートに接続しているかを定義しなければならない．センサのアイコンに対して 1 ～ 4 のラベルをつけることで定義することができる．センサの命令はセンサ毎に複数個存在し，アイコンで表されている．青信号の次のアイコンはタッチセンサを表しており，アイコン内の矢印の向きがセンサの方に向いているので，これはタッチセンサが「押されるまで待つ」ということを表している．なので，この命令ではタッチセンサが押されるまでは次の動作に移らないということを意味している．センサによる分岐ではライトセンサが 40 を超える値を受け取った場合は上に遷移，40 以下の値を受け取った場合は下に遷移というような意味になる．モータのアイコンの上部にはポートを表すため，A，B，C のいずれかが書いてある．モータは 2 種類あり，矢印が右を向いているものは順回転，左を向いているものは逆回転を表している．モータの出力レベルは図 2 のように 5 段階の出力レベルで設定する方法の他に，テキストボックスを付けて 0～100 で細かく出力を設定する方法の 2 パターンある．赤の上下の矢印は，プログラムを上向きの矢印から下向きの矢印まで飛ばすという意味で，主にループとして用いられる．

ROBOLAB によるプログラムのアップロード方法は 1 クリックで行うことができる．画面左上の矢印マークが正常な形をしている時はプログラムにエラーはないということで，矢印のボタンをクリックすることでプログラムのアップロードが可能である．

矢印が割れている場合は，アイコンが正確に繋がれていない等のエラーがあるためプログラムがアップロードできないことを表している．

4. レゴ用状態遷移図設計支援システム

レゴ用状態遷移図設計支援システムは，本論文で提案したレゴプログラミング向けの状態指向状態遷移図を作成することで，その状態遷移図に見合った C 言語のソースコードを自動で生成するシステムである．

状態遷移図の作成手順として，最初にコントロールボックスを設定することでそれに見合った状態を生成し，生成された状態に条件と出力を記述し，状態間を遷移で繋げることで作成することができる．

本提案システムを使用する際には cygwin をインストールしておく必要がある．

4.1 コントロールボックス

NXT にはセンサ用のポートが４つ，モータ用のポートが３つ用意されている．レゴのプログラミングではセンサとモータをどのポートに接続するかを定義する必要があるので，コントロールボックスはその役割を担っている．作成画面の右側にコントロールボックスがあり，コントロールボックス内にはボタンが存在している．このボタンは上段の４つがセンサ用のポートで，左からポート１，２，３，４となっており，下段の３つはモータ用のポートで左からポート A，B，C となっている．このボタンをクリックすることでセンサやモータの設定ができる．

センサの種類はレゴのセットに同封されている「ライトセンサ」,「タッチセンサ」,「ソナーセンサ」,「超音波センサ」の４種類選択することができる．設定されたポートは文字が変化し,「光」はライトセンサ,「触」はタッチセンサ,「音」はソナーセンサ,「波」は超音波センサ,「車」はそのポートのモータを使用していることを表している．コントロールボックスに使用するセンサやモータを設定することで，それに見合った状態を生成することができる．

コントロールボックス下部にあるラジオボタンは通常制御か倒立振子制御のどちらのプログラムを作成するかを設定する．倒立振子制御とはレゴが車輪 2 つでバランスを取りながら倒立をしながら動作をさせることができる制御のことである．倒立振子制御を行う場合はモータ２つと「ジャイロセンサ」が必要になるため，ポート１にはジャイロセンサ，モータ A，モータ C が自動で設定される．その他の制御を行う場合は通常を選択する．

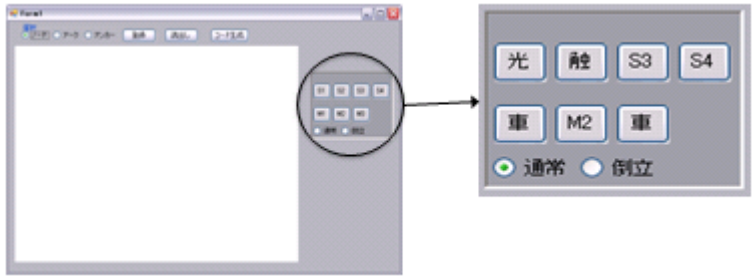

図 3　コントロールボックス

4.2 状態

状態は状態名，条件部，出力部で構成されている．

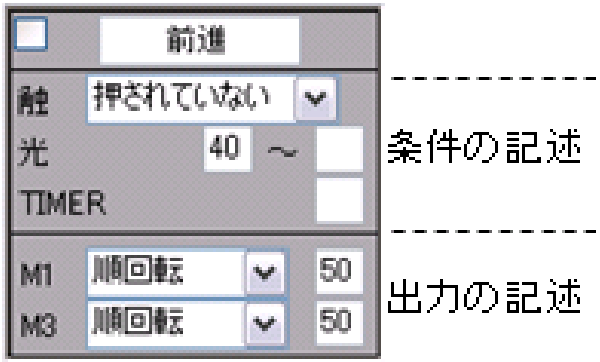

図 4　状態遷移図の状態

4.2.1 条件部

条件部ではコントロールボックスで設定されたセンサの条件とタイマー処理について記述することができる．ここに記述することは，その状態を維持するための条件である．なので，各センサの受け取る値が，条件部に記述されている条件を満たしている間はこの状態を維持し，条件を満たさなくなった場合にこの状態を抜けることになる．なお，条件部で設定できるセンサの値は ROBOLAB に合わせた値であるため，通常の C 言語で記述される API の値とは異なる．以下に各センサの条件の説明を記述する．

・ライトセンサ・・・0～100 の値を設定．値が大きくなるほど反射率が高い（明るい色からの反射）．ライトセンサが設定された値以外の数値を受け取った時に状態を抜ける．
・タッチセンサ・・・「押されている」か「押されていない」を設定．タッチセンサが設定した方と逆の値を受け取った時に状態を抜ける．
・サウンドセンサ・・・0～100 の値を設定．値が大きくなるほど音が大きい．サウンドセンサが設定された値以外の数値を受け取った時に状態を抜ける．
・ソナーセンサ・・・0～200 の値を設定．ソナーセンサ計測距離データ[cm]を取得することができる．設定された値を受け取っている間は状態を維持し，それ以外の値を受け取った場合状態を抜ける．
・タイマー処理・・・設定された時間が経過したら状態を抜ける．設定する値は秒単位である．

条件部にセンサの条件を記述しない場合は，そのセンサは状態を維持する条件に影響しない．また，複数の条件が設定された場合，そのどれか一つでも条件を満たさなくなった場合は状態遷移が起こる．

4.2.2 出力部

出力部ではモータの出力を設定することができる．「順回転」，「逆回転」，「停止」のいずれかを選択することで出力の向きを設定し，その出力レベルを設定することでそれに見合った出力がされる．「停止」を選択した場合には出力レベルを設定しても反映されない．

4.2.3 初期状態

状態名の左側にあるチェックボックスでは初期状態を設定することができる．チェックが入った状態が初期状態として認識される．この状態遷移図では初期状態を複数設定することができ，プログラム実行時の条件で初期状態が変わってくる．

4.3 遷移

状態遷移図を作成する際に，状態間を結ぶ線を遷移とする．状態遷移は全ての状態に対して起こるものでなく，遷移によって結ばれた状態間でのみ起こり得る．

4.3.1 双方向性の遷移

双方向性の遷移は，結ばれた状態間が互いに行き来できることを表している．この遷移は「アーク」が選択されている時に記述することができる．

4.3.2 一方向性の遷移

一方向性の遷移は，結ばれた状態間の状態遷移が矢印の方向にのみ起こり，逆の方向には状態遷移が起こりえないことを表している．この遷移は「アンカー」が選択されている時に記述することができる．

図 5　双方向性の遷移(左)と一方向性の遷移(右)

4.3.3 遷移先の優先度

状態遷移が起こる際に，遷移先の状態はその時の条件を満たしている状態になる．条件を満たしている状態が複数個存在した場合，どの状態に遷移するか想定できない．そのような問題が起こらないように本提案ツールでは，特定の状態に優先的に遷移させることができるように，遷移の長さを意図的に変化させることで状態遷移の優先度の設定を行える．状態から遷移先との遷移の長さが短くなるほど優先度が高くなり，遷移の長さが長くなるほど優先度は低くなる．

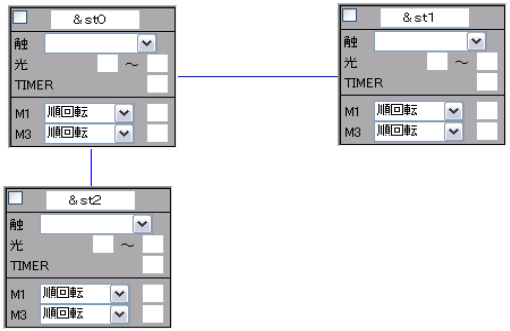

図 6　遷移先の優先度の設定

図 6 では&st0 から状態遷移が起こり，且つ&st1 と&st2 の両方が条件を満たしている場合，遷移の短い&st2 に優先的に状態遷移する．このように遷移の長さを変えることで状態遷移の優先度を設定することができる．

4.4 C コードの自動生成

作成された状態遷移図から，その状態遷移図に見合った C コードを自動で生成することができる．プログラム作成画面の「C コード生成」ボタンを押すことで C コードを生成することができる．

4.4.1 生成される C コード

生成される C コードにはメイン用のプログラムと状態用のプログラムがある．以下に生成されるメインプログラムの一部と状態用のプログラムを示す．

```
TASK(OSEK_Task_bg){
  int i;
  State **nStates, *st;
  nStates = initStates();
  while( nStates != NULL ){
    int noAct = 1;
    for(i = 0, st = nStates[0];
        st!=NULL; i++, st = nStates[i] ){
      if( st->condition() ){
        st->init();
        do {
          st->action();
          systick_wait_ms(5);
        } while( st->condition() );
        noAct = 0;
        nStates = st->nextStates();
        break;
      }
    }
    if( noAct ){
      ERRORMSG();
      nStates=NULL;
    }
  }
}
```

図 7 生成される C コード（メイン）

図 7 はメインプログラムで，この C コードは必ず生成される．メインのプログラムでは条件を満たしている状態の検索，現在の状態が条件を満たしているかの判断が行われている．その時の状態が条件を満たしていればその状態を維持させ，条件が満たさなくなった場合にその時の条件を満たしている状態をその状態の次状態候補から探す．次状態候補とは状態から遷移する可能性のある状態のリストのことである．そして，条件を満たしている状態が存在すればその状態に遷移し，存在しなければエラーとなりプログラムを終了させる．

```
int condition0() {
  状態 0 の条件
}
void action0() {
   状態 0 の出力
}
void init0() {
  状態 0 の初期処理
}
State **nextStates0();
State st0 = { condition0, action0
             , init0, nextStates0 };
State **nextStates0(){
  次状態候補のリストを返す
};
```

図 8　生成される C コード（状態）

図 8 は状態用の C コードである．状態の C コードは状態遷移図で生成された状態の数だけ生成される．condition 関数には各センサの条件が記述される．この関数では，その時の条件を満たしている間は 1 を，条件を満たさなくなった場合に 0 を返却する．action 関数には各モータの出力が記述される．init 関数にはタイマーの初期処理等が記述される．nextStates 関数にはその状態から状態遷移が起こる可能性のある状態のリストが記述される．

この状態は condition 関数，action 関数，init 関数，nextStates 関数の 4 つからなる構造体である．

Vol.2012-SE-175 No.7
2012/3/15

図 9　設定された状態

設定された状態に見合った C コードがセンサ毎に生成される. 例として図 9 の状態から生成される C コードがどのようなものか以下に示す.

・ライトセンサ・・・ecrobot_get_light_sensor(NXT_PORT_S1) < 600

本提案ツールではライトセンサの値は 0～100 で記述されているが生成される C コードでは 600 と規定値を大幅に超えている．これは，本提案ツールで設定される値は ROBOLAB で定義されている値に合わせたものであるが, 実際の C コードでは 0～1023 で扱われるため，設定された値を変換して出力しているからである．なお，C コードでは 0 に近づくほど明るく，1023 に近づくほど暗くなることを表している.

・タッチセンサ・・・ecrobot_get_touch_sensor(NXT_PORT_S2) == 0

タッチセンサでは押されているか，押されていないかを判別している．C コードではタッチセンサが受け取る値が 0 の場合は押されていないことを表し，受け取る値が 1 の場合はタッチセンサが押されたことを表している.

・サウンドセンサ・・・600 <= ecrobot_get_light_sensor(NXT_PORT_S3)

サウンドセンサもライトセンサと同様に C コードでは 0～1023 で記述される. なお, 0 に近づくほど大きな音を表し，1023 に近づくほど小さな音を表している.

・ソナーセンサ・・・ecrobot_get_sonar_sensor(NXT_PORT_S4) <= 100

ソナーセンサは ROBOLAB と同様の値を扱うので設定された数値がそのまま反映される.

・タイマー・・・startTimer(3000)

タイマーに設定された値を msec に直し，タイマー関数で設定された時間が経過した時に状態を抜ける.

・モータ・・・nxt_motor_set_speed(NXT_PORT_A, 50, 1)

モータの出力を設定する．順回転の場合は設定された値がそのまま反映され，逆回転の場合は設定された値を負の値に直して記述される.

上記の C コードが生成され，各センサの C コードは condition 関数に記述され，モータの C コードは action 関数に，タイマーの C コードは init 関数に記述される.

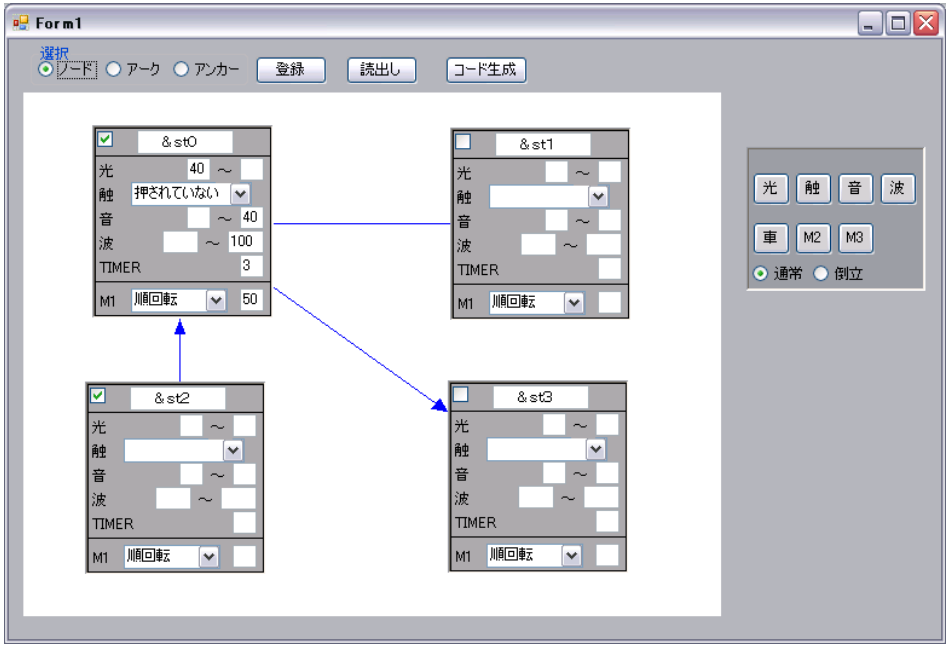

図 10　状態遷移図例

図 10 のような状態遷移図を作成した場合に生成される C コードを以下に記述する.

・&st0 の次状態候補・・・static State *ns[] = { &st1, &st3, NULL };

&st0 から遷移する可能性のある状態が&st0 の次状態候補のリストに加わる．図 10 の状態遷移図では&st0 から派生している遷移は 3 本である．しかし，&st2 に派生している遷移は一方向性の遷移で，&st0 から遷移する可能性はない．なので&st0 から遷移する可能性のある状態は&st1 と&st2 になるので，&st0 の次状態候補リストには&st1 と&st2 の 2 つの状態が加わる.

メインのプログラムでは，状態遷移する際にこのリスト内を左から順に状態を検索していき，条件を満たしている状態があればその時点でその状態に遷移してしまう．なので，リストの後ろになるにつれて状態遷移が起こる優先度が低くなる．前述の遷

移の優先度について，遷移の長さの順にリスト内をソートするようにプログラムしてある．

・初期状態・・・static State *ns[] = { &st0, &st2, NULL };

状態内のチェックボックスにチェックが入っている状態は初期状態として設定された状態である．図 10 では&st0 と&st2 が初期状態として設定されている．なので，&st0 と&st2 が initStates 関数のリストに加わる．initStates 関数は図 7 と図には記述されていないが，初期状態のリストを返却する関数である．

4.4.2 C コードのアップロード

本提案ツールでは状態遷移図の作成のサポートから，その状態遷移図に見合った C コードの自動生成までを行うことができる．しかし，C コードを生成するまでで，その生成された C コードを NXT にアップロードすることは現段階では不可能である．なので，その生成された C コードのアップロード方法として cygwin を用いる．生成された C コードを cygwin が読み込んでコンパイルし，NXT にアップロードすることでプログラムを実行することができる. 以下に本提案ツールと cygwin との相関図を示す．

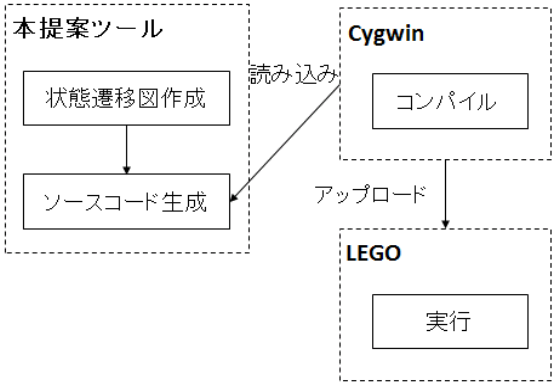

図 11　システムの相関図

4.4.3 その他

プログラム作成画面上の「登録」「読出し」ボタンで作成した状態遷移図のセーブとロードができる．

5.　比較実験

本論文では状態指向の状態遷移図の有用性を検証していくため，本提案ツールと ROBOLAB の比較実験を行った．本提案ツールは状態指向の状態遷移図として，ROBOLAB はフローチャートで状態という概念はないが，遷移に対して出力や入力を記述することから．遷移指向の状態遷移図に見立てて比較を行った．

5.1　実験概要

被験者は Java や C 言語の経験はあるがレゴのプログラミングは未経験の学生 6 人．この 6 人の学生に ROBOLAB と本提案ツールの使用方法を十分に説明してから，この 2 つのツールを用いて同じプログラムを作成してもらった．

作成してもらうプログラムとして簡単なライントレースを問題にした．先にプログラムをしたツールの方が時間がかかると思われるので，最初の 3 人には ROBOLAB から，残りの 3 人には本提案ツールからプログラムを作成してもらった．

今回の実験で計測したものはプログラム全体の作成時間，最初にプログラムを実行してから正しい動作するまでの修正時間, 正しい動作をするまでの修正回数を測った．修正時間は最初にプログラムを実行させた時点で正しい動作をした場合 0 秒となる．その場合は修正回数も 0 となる．

また，実験後に本提案ツールに対するアンケートを採った．

5.2　実験結果

表 1　ROBOLAB の実験結果

ROBOLAB	A	B	C	D	E	F	平均
作成時間（m:s）	17:00	27:40	28:50	22:00	37:30	17:20	25:03
修正時間（m:s）	3:45	7:26	4:02	2:00	0:00	8:20	4:15
修正回数（回）	1	2	1	1	0	4	1.5

表 2　提案ツールの実験結果

提案ツール	A	B	C	D	E	F	平均
作成時間（m:s）	21:30	8:10	20:30	21:30	24:27	14:00	18:21
修正時間（m:s）	12:00	0:00	12:30	9:06	0:00	11:07	7:27
修正回数（回）	1	0	1	1	0	3	1

実験結果を表 1，表 2 に示す．実験結果からプログラムの作成時間の平均は本提案ツールの方が早いという結果になったが，平均修正時間に注目すると ROBOLAB の方が短いという結果になった．平均修正回数は本提案ツールの方が少なく済んだ．

アンケートは以下のような回答が多かった．

・１つの状態で条件や出力の記述が容易にできる
・シンプルで見やすい
・プログラムの流れが読みづらい
・プログラムが複雑になるとわかりづらくなる
・コンパイルに手間がかかる

5.3 考察

結果から，本提案ツールの方がプログラムの作成時間は短かった．また，修正回数も少なかった．状態が少ない場合，簡潔に状態遷移図を記述することができたため，初めて使う人にとってわかりやすかったからであると思われる．実験後に採ったアンケートでは，本提案ツールは状態が少ない時はわかりやすい，記述することが少なくシンプルにまとめられる等の意見が多かった．この結果から初心者でも簡単に記述できることが考えられる．

しかし，修正時間は ROBOLAB に比べ時間がかかっている．この原因として考えられることは，ROBOLAB はフローチャート形式でプログラムの流れが読みやすい．なので，誤った動作をした部分をプログラム中から探すのが容易であると思われる．対して，本提案ツールの状態遷移図からはプログラムの流れを読むのが難しい．状態毎の条件や出力は一目で見ることができるが，プログラム全体で考えると時間軸がないためプログラムの流れが把握しづらいことがこの差を生んだと思われる．アンケートには，プログラムのミスを探すのが困難であるとの意見が多く，状態が多くなると状態遷移図自体が見づらくなるという意見もあった．状態と遷移が増えることで状態遷移図が見づらくなってしまうので，そのような事態を避けるため複数の状態をグループ化することで，グループ内の状態間は遷移可能と定義し，遷移を減らし状態遷移図を見やすくするよう検討をしている．また，プログラムのアップロードも手間がかかり，被験者に余計な負担をかけてしまった．本提案ツールでも，1 クリックで cygwin にコマンドをとばし，プログラムのコンパイル・アップロードが可能になるよう検討している．

6. おわりに

本論文では，状態指向の状態遷移図の概念を基に，レゴのプログラミングに対応した状態指向状態遷移図の提案を行った．その状態遷移図は，状態に条件部出力部という概念を加えることで，遷移には何も記述しない方式の状態遷移図になる．そして，レゴプログラミング用の状態指向状態遷移図を作成することで，作成された状態遷移図に見合った C コードを自動生成するツールを開発した．

開発した本提案ツールを用いて ROBOLAB との比較実験を行った．今回の実験では，本提案ツールでプログラムした場合と，ROBOLAB でプログラムした場合の時間を測定した．実験結果から，本提案ツールでプログラムを作成した場合の方が短時間でプログラムができている．アンケートにも，状態遷移図が簡潔に書くことができて見やすかった，という意見が多かった．簡単なプログラムであれば初心者でも容易にプログラムミングできることがわかった．しかし，プログラムの作成時間は本提案ツールの方が短時間で完成したが，本提案ツールは修正時間に時間がかかっている．これはプログラムの流れが読みづらい，プログラムが複雑になると状態遷移図が見づらくなることが原因であると思われる．そういった視覚的な部分の改善が今後の課題である．また，今回の実験では状態の数が少なかったためこのような結果になったことも考えられるので，状態が多くなった場合の例についても実験を行っていく必要があると思われる．

今回のレゴの実験から，状態指向の状態遷移図の方が容易に状態遷移図を作成することができると思われるが，タッチセンサのみ本提案ツールでは記述が困難であると感じられた．押されたら状態遷移が起こるようにするには，状態に条件として記述するのは難しく思える．なので，レゴのプログラミングについては，状態指向の状態遷移図にタッチセンサのみ遷移指向の考え方でのプログラミングが望ましく思える．

本論文ではレゴのプログラミングについて行ってきた．しかし，本論文だけではレゴのみに固執した考えになってしまうので，状態指向の状態遷移図の有効性を評価していくためには，レゴのみでなく自動販売機や電子レンジの制御システム等，他の組込みシステムについての評価も行っていく必要がある．

7. 参考文献

[1]紫合　治、“状態指向のステートチャート”、FOSE2005
[2]ROBOLAB　2.9　ガイドブック、株式会社アフレル、2006
[3]今井　康平、紫合　治、“レゴ向けの状態遷移図設計支援システム”、第 73 回情報処理学会全国大会
[4]nxtOSEK/JSP　ホームページ、http://lejos-osek.sourceforge.net/jp/index.htm
[5] SDL-CCITT Specification and Description Language、ANDERS ROCKSTROM、ROBERTO SARACCO