プログラムの自動採点を核とした学習支援システムの構築

石川 貴彦
北海道大学 工学系教育研究センター
〒060-8628 札幌市北区北13条西８丁目
e-mail ishikawa@ist.hokudai.ac.jp

赤間 清
北海道大学 情報基盤センター
〒060-0811 札幌市北区北11条西５丁目
e-mail akama@iic.hokudai.ac.jp

概要

アルゴリズム構築の方法を学ばせることを目的として、我々はプログラミング言語と学習支援システムの両面から、プログラミング教育の方法論を検討している。本研究では、アルゴリズムの構築方法を教示することができるET言語を採用し、アルゴリズムとしてのETプログラムを簡便に評価する、かつ適切な支援を行うための、プログラム自動採点システムを構築した。さらに、システムによる評価を補強し、自学を促進させるための学習者用ツールや、学習進度を把握するための教師用ツールを作成し、自動採点システムに付加して支援体制を充実させた。これにより、プログラムの自動採点を核とした学習支援システムを実現・提供することができた。

１．はじめに

プログラミング教育は、アルゴリズム構築能力を育成したり、問題発見・解決能力を養ったりするうえで効果的な題材である。しかしながら、その実践は難しいものと認識されており、学習者に対して、プログラミング教育の効果を十分にもたらしていないことが課題となっている。

教育実践の難しさを解消するために、これまで様々な取り組みが行われてきた。大まかに分類すると、

(1) 教育用言語の開発に関するもの
(2) 学習支援システムの開発に関するもの

の２つが挙げられる。(1)の例として、BASICやLogo、近年では若葉[1]などがあり、特に、枝葉末節あるいは言語依存の知識の習得の負担を軽減することを主な目的として開発されたものである。また、(2)の例としては、一般的によく知られているC言語やJava言語に焦点を当て、それをチュートリアルに従って順序よく学ぶためのシステム[2]などがある。プログラミング教育に対する取り組みを概観すると、多くは言語か支援システムのどちらか一方に着目し、実践を容易にしようとしている傾向にある。すなわち、プログラミング教育における部分的な改善は、各々の取り組みで実現されているものの、言語と支援システムの両面からプログラミング教育全体を見直し、改善に取り組んだ例はあまり見られない。

プログラミング教育全体を見据え、適切な実践を行うには、まずプログラミングにおいて何を教えることが重要であるかを定義し、次にそれに適応する言語と、言語の特性に見合った学習支援システムを総合的に検討する必要がある。

我々は、与えられた問題を正しく解くための方法（アルゴリズム）を学ぶことが、プログラミング教育において普遍性の高い内容であり、特に重要であると考えている。そして、

① アルゴリズムの作り方を教示できるプログラミング言語を採用し教材化する
② 支援体制のシステム化を図り、効果的なプログラミング教育の方法を与える

という２点に着目して、プログラミング教育の方法論について検討・実践を重ねている。

本研究ではET言語を採用し、アルゴリズムの学習を支援するシステムについて検討・実装を行った。さらに、いくつかのサブシステムを付加して、プログラミング教育における、支援システムの１つの枠組みを構築した。

Construction of a Learning Support System based on Automated Evaluation of Programs.
T. Ishikawa and K. Akama
Hokkaido University.

2．教育で用いる言語と支援方法

2.1 教育実践に用いるプログラミング言語

アルゴリズムを構築する方法を教示できるプログラミング言語として、我々は ET 言語[3]に注目している。ET 言語は、与えられた問い合わせを、正当で簡単な部分問題の集合に置き換える等価変換ルールを追加・実行することにより、問題解決を行うルール型言語である。

ET 言語の基礎となる等価変換による計算モデルは、アルゴリズムの正しさを広く確保するための十分条件に関する科学的知見があり、それは、

- 正当性･･･等価変換ルールだけからなる
- 停止性･･･等価変換ルールの集合によって引き起こされる計算は停止する
- 被覆性･･･想定される問い合わせのすべてに答えることができる

という性質を満たすようにルールを作成・追加して、アルゴリズム（ET プログラム）を構築するというものである。

また、ET 言語に似た言語として、Prolog などの論理型言語があるが、プログラムの部品としての「独立性」に関して違いがある。Prolog は、節の組み合わせによって問題が表され、それをプログラムとしているので、１つの節をプログラムの完全な部品として扱うことはできない。ET 言語は、各ルールが適用可能かどうかを、他のルールとは無関係に議論できる。そのため、ルールを追加・削除してプログラムを改善したりすることが容易である。

この十分条件に関する科学的知見とルールの独立性が、正しいアルゴリズムを構築するための方法と、小さなステップで実践する方法をもたらす。これを実現する ET 言語を学ぶことが、アルゴリズム構築の学習において有用であると考える。

2.2 ET プログラムの作成過程

ET プログラムは等価変換ルールの集合であり、我々はこれをアルゴリズムとみなしている。このことを説明するために、「２つのリストを結合して、新たなリストを求めるプログラムを作りなさい。」という問題を取り上げる。具体例として、リスト[1, 2, 3]と[4, 5]を結合したリスト Z を求めることを考える。問い合わせ *Ans* は、節を用いて次のように記述される。

Ans :(append (1 2 3) (4 5) *Z)
← (append (1 2 3) (4 5) *Z).

ちなみに、*で始まるシンボルは変数を表している。問題文と *Ans* から、「Z の最初の要素は１であり、Z の１より後のリストは、[2, 3]と[4, 5]を結合したリストである」ことが思いつき、次の等価変換ルールを記述することができる。

r1: (append (*A | *X) *Y *Z)
→ (= *Z (*A | *Z1)), (append *X *Y *Z1).

この *r1* ルールは、「リスト(*A | *X) と *Y を結合したリストが *Z であるという問題は、*Z を (*A | *Z1)というリストにするという問題、かつリスト *X と *Y を結合したリストが *Z1 であるという問題に置き換えることができる」という手続的意味を持つ。このルールを ET の処理系で実行すると、*Ans* のボディアトム（←の右にあるアトム）に *r1* が数回適用され、次の書き換えが起こる。

Ans : (append (1 2 3) (4 5) (1 2 3 | *D))
← (append ( ) (4 5) *D).

この結果は、「空リストとリスト[4, 5]を結合したリストが D ならば、リスト[1, 2, 3]と[4, 5]を結合したリストは[1, 2, 3 | D]である」という意味を持つ。なお変数*D は、実行において自動的に生成された中間変数である。append アトムを処理して*D が求まれば答は得られる。したがって、今までのルールの適用とは無関係に、この結果に適用するルールを発見し追加すればよい。この場合は、「空リストとリスト Y を結合したリストが Z であるとき、Z は Y である」ことを思いつき、

r2: (append ( ) *Y *Z) → (= *Z *Y).

というルールを記述することができる。*r1* と *r2* の２つのルールを蓄積し、再び実行すると、

Ans : (append (1 2 3) (4 5) (1 2 3 4 5))) ←.

という結果を得る。これは、（無条件で）[1, 2]と[3, 4, 5]を結合したリストが[1, 2, 3, 4, 5]であるこ

とを意味し、問題の答を表している。さらに、*r1* と *r2* の２つのルールによるETプログラムは、2.1節の十分条件（任意の長さのリストに対して、正しく答を求めることができる）を満たしており、アルゴリズムの本質的な内容を示している。

2.3 どのような支援が必要か

ここでは、ET言語によるプログラミング教育を実践する際の支援方法について検討する。一般に、プログラミングは、

(1) 文を記述してプログラムを得る（作成）
(2) プログラムを実行して答を求める（実行）

という、２つの段階から構成される。教育を考えると、(1)に関しては、テキストや例題を読ませたり、４択問題で基礎知識を確認させたりする方法などがあり、教育コンテンツを事前に準備できれば、(1)に対する支援は概ね達成できる。(2)に関しては、作成したプログラムを教師が実際に動かしてみたり、目視したりして、プログラムの正否を検証する。この場合、事前に準備できるものはあまりなく、学習者の進度に応じて適宜行うことが考えられる。

したがって、プログラムの実行の評価は、教育実践において大きな負担となり得る部分であるといえる。ここをある程度自動化できるならば、評価に要する時間の短縮や、採点ミスの減少などが期待され、学習者に対して、迅速かつ的確に結果を伝達できるものと思われる。

以上より、本研究ではETプログラムを実行して、それを簡便に評価する自動採点システムを構築する。これまでの自動採点システムには、過去の事例を用いてプログラムの実現方法（指定された文を使っているか等）を検査するもの[4]や、構文チェッカーなど様々なものがあるが、本システムでは、問題の書き換えというET言語の特性を活かして、プログラムの実行結果と解答との照合による動作チェックを行う。

3. プログラム自動採点システム

3.1 システムの概要

本システムは、Webベースシステムになっており、Webサーバ、CGIプログラム、データベースの３層構造で構成されている（図１）。教師は、プログラム作成課題と、採点のためのテストデータを準備し、データベースに登録しておく。学習者はWebページに提示された課題を読み、テキストエディタでプログラムを作成し、ET言語の処理系で実行する。学習者自身の判断で完成と認めたならば、Web上の投稿フォームにプログラムを入力し（エディタで作成したものを貼り付けて）送信する。システムは一定時間毎に、送信されたプログラムに、いくつかのテストデータを適用させて実行結果を得る。そして、実行結果と解答との照合を行い、採点結果をデータベースに登録する。採点結果はWebページ上で各学習者に対して適宜提示される。

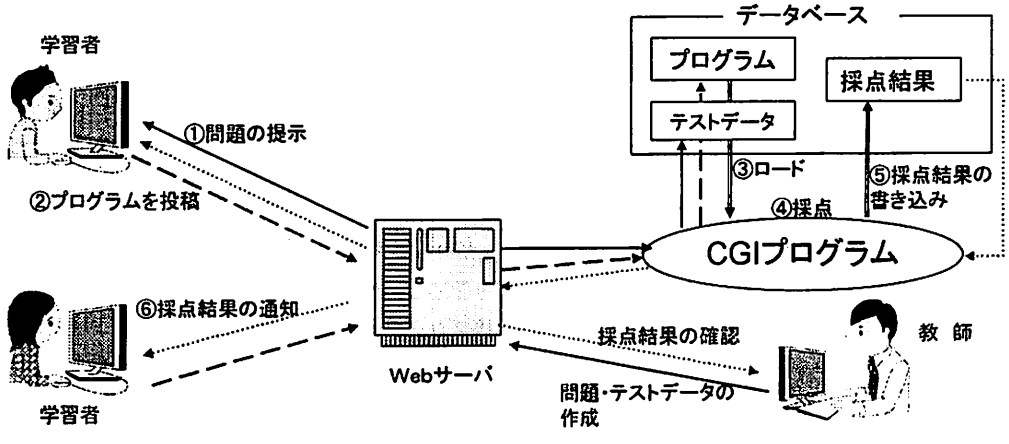

図１　プログラム自動採点システムの概要

3.2 テストデータの作成

本システムにおけるテストデータは、投稿プログラムを検証するための問い合わせと、求めるべき解答のペアを矢印でつないだものである。問い合わせは、投稿プログラムの実行によって生じる実行結果を得るために必要であり、求めるべき解答は実行結果との照合のために使われる。例えば階乗の値を求めるプログラムでは、

(fact 5 *X) ⇒ (fact 5 120)
(fact 0 *X) ⇒ (fact 0 1)
(fact 6 *X) ⇒ (fact 6 720)

と具体例をいくつか列挙し、システムに登録する。テストデータを作成するための観点は、

- 問題文で定義した範囲以外の値を用いない（ここでは負の数や小数は用いない）
- 登録したテストデータ全体によって、プログラムとして求めるべき範囲を包括する
- 特殊なケース（ここでは0の階乗が1であること）が適用する例を含ませる

ことなどが挙げられる。ET プログラムは、再帰的に繰り返し計算するルールと、再帰を停止させるルールから構成される場合が多く、階乗の計算では、1以上の値が与えられた場合には、再帰ルールが数回適用された後、停止ルールが最後に適用して答を求め、0が与えられた場合には停止ルールのみが適用して答を求める。

したがって、テストデータは少なくとも2例が必要になり、与える値が1以上の場合と0の場合でプログラムの求める範囲を包括する。しかしながら、(fact 5 *X) → (= *X 120). というルールを記述して、5の階乗が120であることだけを求める場合も考えられるため、再帰ルールが適用する例を増やすと採点の信頼性が向上する。

他にも、「与えられたリストの要素が2種類 A、B (A≠B)で構成されているかどうかを判定するプログラムを作成しなさい」という課題では、

(elem (1 2 1 2 2)) ⇒ (true)
(elem (3 3 3)) ⇒ (false)
(elem (7 8 9 0)) ⇒ (false)
(elem (a b)) ⇒ (true)
(elem (k)) ⇒ (false)

というようにテストデータを登録する。これは、

- リストの構成要素が2種類よりも少ない
  [(elem (3 3 3))、(elem (k))]
- リストの構成要素が2種類よりも多い
  [(elem (7 8 9 0))]
- リストの構成要素が2種類
  [(elem (1 2 1 2 2))、(elem (a b))]
- リストの長さが2以上
  [(elem (a b))、(elem (3 3 3))、(elem (7 8 9 0))、(elem (1 2 1 2 2))]
- リストの長さが2より小さい
  [(elem (k))]
- リストの要素が数である
  [(elem (3 3 3))、(elem (7 8 9 0))、(elem (1 2 1 2 2))]
- リストの要素が文字である
  [(elem (a b))、(elem (k))]

という項目を5つの例で検査している。このように、テストデータにいくつかの観点を含ませることで、様々なケースを考慮したプログラムの作成を学習者に意識づけている。

3.3 プログラムの評価のしくみ

投稿プログラムの採点にあたり、システムは最初に未採点プログラムのシンタックスチェックを行う。このチェックを通過すると、採点が開始される。採点は ET 言語がサポートする多重計算機構（ワールド機構）と、ワールド機構の中で計算を行うための call アトムを用いて行われる。ワールド機構は、ET プログラムの実行空間を複数操るための機構であり、本システムでは、採点プログラムの実行の中で投稿プログラムを実行するために用いている。call アトムは、

(call *query *w *msec (*atoms *mes))

と記述し、*query には問い合わせを、*w には投稿プログラムが格納されているワールド機構の識別子を、*msec には実行の制限時間を与える。このアトムが処理されると、*atom には問い合わせに対する実行結果が返り、*mes には実行結果を表すシンボルが返る。実行結果を表すシンボルに

は以下の３つがある。

true … 実行が成功して終了
false … 実行が失敗して終了
exception … 実行中にエラーが発生して終了

この３種類のシンボルと実行結果、そして、テストデータの解答を用いて評価を行う。階乗の値を求めるプログラムが正解で、テストデータが

(fact 5 *X) ⇒ (fact 5 120)

ならば、実行結果とシンボルは、

(fact 5 120) ← .	実行結果
true	シンボル

となる。返り値を求めるタイプの問題は、テストデータの解答と、実行結果のヘッド（←の左にあるアトム）が一致し、シンボルは true となる。

また、前述した elem2 のプログラムの実行が失敗を返して正解となる場合、テストデータが、

(elem2 (3 3 3)) ⇒ (false)

ならば、実行結果とシンボルは、

(elem2 (3 3 3)) ← .	実行結果
false	シンボル

となる。正誤判定を行うタイプの問題は、実行結果のボディに（←の右に）アトムがなく、シンボルは false を返す。

このような方法で、プログラムの実行結果を分類し、各ケースについて独自の識別子を定義した。

- correct … 正しい答が求まる（解答と実行結果のヘッドが一致かつシンボルが true、または、解答とシンボルが false かつ実行結果のボディアトムがない）
- incorrect … 正しくない答が求まる（解答と実行結果が不一致かつシンボルが true、または、解答とシンボルが不一致かつ実行結果が空リスト）
- undefined … ワールド機構の中で１度もルールが適用されない（実行結果のヘッドとボディが一致）
- incomplete … 実行が途中で終わる（実行結果のボディが空リスト以外）
- false … 実行が失敗した場合（解答が false 以外かつシンボルが false）
- exception … 実行中に例外が発生した場合（シンボルが exception）
- timeout … 実行制限時間のオーバー（call アトムに指定した制限時間を越える）
- syntax_error … シンタックスエラー（ワールド機構にプログラムをロードすることができない）

次に、識別子を要素に持つリストを得る。例えば、テストデータが１つのプログラムに対して３つ登録され、全て正しい答が求まった場合は、

(correct correct correct)

というリストが得られる。このリストを用いて最終判定を行う。この場合は、全てのテストデータにおいて correct であるので、プログラムは正しいと判定し、データベースに「合格」というメッセージを登録する。また、得られたリストの要素全てが incorrect のときは、「全ての質問に対して誤った結果を返しました」というメッセージを出力し、１つ以上 incorrect が含まれているときは、「ある質問に対して誤った結果を返しました」というメッセージを登録するようにした。このようにして、リストに含まれる識別子の個数からも、メッセージの区別を行うようにした。さらに、ケアレスミスが誤りの原因として考えられるものを優先した。

- 「未採点」
  … プログラムを評価する時間に達していない
- 「ルールの書き方に誤りがあります」
  … シンタックスチェックが不通過
- 「ルールの適用条件・数式に誤りがあります」
  … リストの要素１つ以上が exception
- 「無限ループあるいは標準計算時間のオーバー」
  … リストの要素１つ以上が timeout
- 「指定した述語名・引数の数になっていません」
  … リストの要素全てが undefined
- 「全ての質問に対して計算が途中で失敗します」
  … リストの要素全てが incomplete、または要素全てが false
- 「全ての質問に対して誤った結果が得られます」

… リストの要素全てが incorrect
・「プログラムに処理できないアトムがあります」
… リストの要素1つ以上が incomplete
・「ある質問に対して計算が途中で失敗します」
… リストの要素1つ以上が false
・「ある質問に対して誤った結果が得られます」
… リストの要素1つ以上が incorrect
・「ある質問に対するルールが不足しています」
… リストの要素1つ以上が undefined
・「合格」… リストの要素全てが correct
この評価基準を用いて、投稿プログラムの実行結果とテストデータとの照合によって得られたリストを判定し、結果を学習者に提示した。

4．学習支援のためのサブシステム

4.1 プログラムの簡潔さの評価

プログラムは、正しいものができればそれで終わりというものではない。完成プログラムをもとにして改善を行い、簡潔化や高速化を試みる機会を与えることも、プログラミング教育にとって重要である。ET 言語は、ルールの入れ替えや変更が、プログラム中の他のルールに影響しないので、プログラムの改善は容易に行うことができる。

この ET 言語の特長を踏まえ、プログラム改善を促すための1つの方法として、プログラム記述の簡潔さに着目した評価方法を提案する。それは、プログラムを構成する全てのルールをトークンに分解し、それを「評価値」として簡潔さを測る方法である。評価値は、

評価値 = プログラム中の括弧（と）の個数
+ 述語の個数 + 引数の個数
+ 縦棒の個数

で求められる。例えば、指定した要素がリストの中にいくつあるかを求めるルールは、

```
c1 : (count *N *L *X),
     {(= (*A | *R) *L), (== *A *N)}
       → (count *N *R *X1), (:= *X (+ *X1 1)).
c2 : (count *N (*A | *R) *X),
     {(== *A *N)}
       → (count *N *R *X1), (:= *X (+ *X1 1)).
c3 : (count *N  (*N | *R) *X)
       → (count *N *R *X1), (:= *X (+ *X1 1)).
```

と3通り記述することができる。*c1* ルールは、括弧の数が 14、述語の数が 6、引数の数が 14、縦棒の数が 1 であるので、評価値は 35 となる。*c1* ルールの中括弧で囲まれた (= (*A | *R) *L)アトムは単一化を行うだけなので、ヘッドの count アトム（先頭のアトム）の*L を直接 (*A | *R) と表してもよい。それが *c2* ルールであり、評価値は 30 となる。*c2* ルールの中括弧で囲まれた (== *A *N) アトムは*A と*N が等しいかどうかを検査する条件判定のためのアトムである。このアトムを除いてヘッドを(count *N (*N | *R) *X) と記述し、第1引数の*N と先頭要素である*N が等しいときに適用するルールにすれば、中括弧内での処理は不要になる。それが *c3* ルールであり、評価値は 25 となる。

このように、同じ計算を行うルールであっても、ルールを構成する要素が少ないほど、簡潔なプログラムであることを指摘している。しかしながら、評価値が小さいものが、必ずよいプログラムであることを明確に決定づけるものではない。実際の学習では、合格したプログラムをさらに短く記述できるか検討させる以外にも、異なる評価値を持つプログラムを作成させて、様々なアルゴリズムを発見することを促すなど、アルゴリズムを見出すための牽引的な役割として活用している。

4.2 投稿プログラム数の推移

学習者のプログラム作成を通じて、教師は学習進度を把握することができるように、1日あたりの投稿プログラム数を棒グラフで表示するようにした（図2）。投稿数の推移を見ることで、学習進度がクラスに適しているかどうかを判断する目安を与えている。

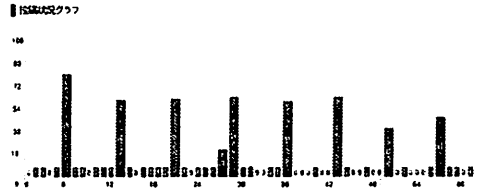

図2　投稿状況グラフ

4.3 ランキングの公開

学習者のプログラム作成を促進させ、同じ問題に対する様々なプログラムを教示するために、評価値のランキングを表示するページを設けた。

合格と判定されたプログラムは、Webページ上にて学習者のハンドルネームと投稿時間、評価値が表示される（図3）。

長さランキング

名前	投稿時間	長さ	プログラム
nayuta	2004/05/25 16:36	50	プログラム
konta	2004/05/25 16:58	50	プログラム
YMD	2004/06/22 19:20	50	プログラム
saku	2004/05/19 20:15	52	プログラム
mure	2004/05/25 16:52	55	プログラム
hetare	2004/05/25 17:37	55	プログラム
orz OTZ ○凡	2004/05/18 17:13	57	プログラム
LOVE	2004/05/25 16:42	57	プログラム

図3　ランキングの公開

そして、合格となった学習者の人数（1人の学習者が2度合格になっても1人分とカウントする）が、クラスの一定人数に達したとき、合格のプログラムが公開されるしくみになっている。なお、合格者と判定された場合でも、プログラムを再投稿することが可能である。

5．学習支援システムを用いた授業実践

5.1 プログラミング演習の概要

ET言語と自動採点システムを用いて、本学の1年生22名を対象にプログラミング演習を実施した。演習は、授業時間中の教師の一斉指導・支援と、学習支援システムを用いた自学方式を組み合わせた、blended-learning[5]の形態で行った。演習で提示したプログラム作成課題の例としては、N番目のフィボナッチ数を求める計算や、素数の判定といった「数」を対象とした問題、与えられたリストの長さを求めたり、数リストの要素を昇順にソートしたりするといった「リスト処理」に関する問題を中心としている。

5.2 学習者のプログラムから見たシステムの評価

ある学習者のプログラム作成の過程を取り上げる。例えば、「与えられた正整数N以下で最も大きい7の倍数Xを求めるプログラムを作成しなさい」という課題では、最初に以下の投稿があった。

5月28日　13:15 作成

```
(nearest *N *X),{(not:= (mod *N 7) 0),(> *N 0)}
        → (:= *N1 (- *N 1)),(nearest *N1 *X).
(nearest *N *X),{(:= (mod *N 7) 0),(> *N 0)}
        → (:= *X *N).
```

このプログラムは、ETの組み込み述語であるnotの記述ミスによって不合格と判定されたものである。（実際には、自動採点システムはnot:=アトムを処理できるルールがないこと（incomplete）で不合格を返している。）これは、シンタックスの理解が不十分であったことが予測される。

次に、この受講者はnot述語を用いないプログラムをすぐに投稿している。

5月28日　13:24 作成

```
(nearest *N *X),{(:= 0 (mod *N 7)),(> *N 0)}
        → (:= *X *N).
(nearest *N *X),{(:= 1 (mod *N 7)),(> *N 0)}
        → (:= *N1 (- *N 1)),(:= *X *N1).
(nearest *N *X),{(:= 2 (mod *N 7)),(> *N 0)}
        → (:= *N1 (- *N 2)),(:= *X *N1).
(nearest *N *X),{(:= 3 (mod *N 7)),(> *N 0)}
        → (:= *N1 (- *N 3)),(:= *X *N1).
(nearest *N *X),{(:= 4 (mod *N 7)),(> *N 0)}
        → (:= *N1 (- *N 4)),(:= *X *N1).
(nearest *N *X),{(:= 5 (mod *N 7)),(> *N 0)}
        → (:= *N1 (- *N 5)),(:= *X *N1).
(nearest *N *X),{(:= 6 (mod *N 7)),(> *N 0)}
        → (:= *N1 (- *N 6)),(:= *X *N1).
```

このプログラムは合格と判定されたが、評価値は222と高い数値を示している。さらに、受講者はこれを改善し、最初に試みたnotを用いたプログラムを投稿している。

5月29日　19:01 作成

```
(nearest *N *X),{(< *N 7),(> *N 0)} → (:= *X 0).
(nearest *N *X),{(:= 0 (mod *N 7)),(> *N 0)}
                → (:= *X *N).
(nearest *N *X),{(not (:= 0 (mod *N 7))),(> *N 0)}
```

→ (:= *N1 (- *N 1)),(nearest *N1 *X).

このプログラムも合格と判定され、評価値も 80 となった。

自動採点システムにより、プログラム作成を自分のペースで取り組むことのできる環境を提供し、正否の評価を迅速に行うことができた。この例では、約1日で適切なプログラムを作成するための支援を達成している。また、プログラムの正否の評価以外に評価値を導入したことで、より適切なプログラムを作成する意欲を与えたと推察される。

5.3 採点結果の内訳から見たシステムの評価

授業実践において、システムが採点したプログラムの数について調査した。22 名が投稿し採点されたプログラムの総数は半期で 1831 あり、採点結果の内訳は、表1のとおりである。

表1　採点結果の内訳

メッセージ	割合(%)
合格	83.3
ある質問に対して誤った結果が得られます	4.5
ある質問に対するルールが不足しています	3.9
ある質問に対して計算が途中で失敗します	2.6
ルールの書き方に誤りがあります	2.2

多くのプログラムは合格と判定されており、いくつかのプログラムを取り上げて、教師が目視した結果からも、適切なプログラムであると認定された。よって、学習者のプログラムを、システムを用いて十分に評価することができたといえる。しかしながら、先程示した(fact 5 *X) → (= *X 120). という具体例の列挙によって偶然合格になったプログラムも若干みられた。この場合は質問に答えることのできる範囲が狭く、あまり適切なプログラムではない。このような場合における支援は教師が直接行い、システムが指摘できない部分をフォローすればよいと考えている。

また、誤りに関しては、ある質問に対するルールの誤りがほとんどであり、想定する質問の範囲を学習者が十分に認識していなかったことが原因であった。これは、課題提示における教師の説明に曖昧なところが見受けられ、誤りを引き起こしたと、授業実践の結果から判断した。システムは、そのことをプログラムの誤りとして評価しており、誤りの原因を的確に指摘していた。この場合は、学習者に対する支援だけでなく、教師に対する指導面の支援としても作用しており、教師の説明の改善を促した結果であると捉えることができる。

6．おわりに

本研究では、アルゴリズムの構築方法を教育するために ET 言語を用い、アルゴリズムとしての ET プログラムを簡便に評価するための自動採点システムと、システムをサポートするサブシステムを作成して、プログラムの自動採点を中核とした学習支援システムを構築した。

システムの運用により、多くのプログラムを迅速かつ的確に評価し、採点に要する時間の短縮化と、適切な学習支援を実現することができた。今後は、構文チェッカーなどの「作成」を評価するシステムを構築し、今回の「実行」を評価するシステムと組み合わせて、プログラムの評価の精度を向上することを考えている。

参考文献

1) 吉良智樹, 並木美太郎, 岩崎英哉：初心者入門用プログラミング言語「若葉」の言語仕様と処理系の実装, 情報処理学会「プログラミング」研究会論文誌, Vol.40, SIG10(PRO5), pp.28-38, 1999.

2) 高橋参吉, 佐野繭美, 橋本はる美, 牧野純, 松永公廣：Web 問題集を使った C プログラミングの授業設計, 教育システム情報学会誌, Vol.20, No.4, pp.392-397, 2003.

3) 赤間浄, 繁田良則, 宮本衛市：論理プログラムの等価変換による問題解決の枠組, 人工知能学会誌, Vol.12, No.2, pp.90-99, 1997.

4) 渡辺博芳, 荒井正之, 武井恵雄：事例に基づく初等アセンブラプログラミング評価支援システム, 情報処理学会論文誌, Vol.42, No.1, pp.99-109, 2001.

5) 佐々木弘記：教員研修における e ラーニングと集合研修のブレンディングに関する一考察, 日本教育工学会論文誌, Vol.28, Suppl, pp125-128, 2004.