ヒューマンインタフェース 41−19
（1992. 3. 3）

認知機構の汎用シミュレータ

大野健彦、乗松敏雄、木村 泉
東京工業大学理学部

John R. AndersonのＡＣＴ*に基づく、人の認知機構の汎用シミュレータについて述べる。このシミュレータはＡＣＴ*の宣言的記憶、プロダクション記憶、プロダクションの発火および学習を表現するとともに、もとのＡＣＴ*において言及されながら実現されてはいなかった時系列をサポートしている。またRumelhartおよびNormanの設計に倣った手のモデルが組み込まれている。シミュレータはＣで書かれ、目下のところは文字ベースの会話型インタフェースによっている。グラフィック表示の機構は目下開発中である。

A General-Purpose Simulator for Human Cognitive Mechanisms

Takehiko Ohno, Toshio Norimatsu, and Izumi Kimura

Tokyo Institute of Technology, Department of Information Science
2-12-1 Ookayama, Meguro-ku, Tokyo 152 JAPAN

A general-purpose simulator based on John R. Anderson's ACT*, a comprehensive model of the human cognitive mechanisms, is described. It models the declarative and production memories of ACT*, as well as the firing and learning mechanisms of productions. Temporal strings, considered but left unimplemented in original ACT*, is available. A model of hands generally following the design of Rumelhart and Norman is also provided. Written in C, the simulator has a character-based conversational interface. A set of graphic aids is under development.

1．はじめに

コンピュータシステムの利用者インタフェースをよいものにするためには、システムに対面する人間の側の機構を理解することが必要である。そのための一つの強力な手段は、人の認知機構に関するモデルを利用することである。その種のモデルとしては、たとえばCardら[1]のModel Human Processor（ＭＨＰ）が著名である。しかしＭＨＰには、記憶、学習、並列的情報処理（「ながら処理」）などの局面ではもの足りない点がある。Newellら[2]はＭＨＰの有力後継候補として、John R. AndersonのＡＣＴ*[3]を挙げた。

ＡＣＴ*はプロダクションシステムに基づくきわめて広範なモデルである。Andersonはこれを用いて、言語の習得を含むさまざまな現象の統一的説明を試みた。筆者らの一人は、ＡＣＴ*のアイディアを利用すれば日本語ワープロ関連のいくつかの現象（たとえばキーボードの習得に際して起こる配列間の干渉現象[4]）を定性的に説明できる、ということに気づいた。

しかしながらＡＣＴ*は、同時進行する事象の間の競合関係に基礎を置く数量的モデルであって、そのような競合関係が強く問題になるような事象では、定性的説明には限界があり、シミュレーションなどの方法を用いて数量的説明をすることが必要となる。そのためにはシミュレータがほしい。

Andersonは以前にＡＣＴ*の、ある程度一般的なシミュレータを開発した由であるが、それは現在ではもはや入手不能である[5]。そこで筆者らは、ＡＣＴ*の一般性の高いシミュレータを、われわれ固有の応用を頭に置きながら新たに開発することにした。本文ではできたシミュレータの利用者から見た姿、内部構造、および開発上出会った問題点（主として文献[3]の解釈に関する）を論ずる。

2．ＡＣＴ*の枠組み

ＡＣＴ*についてのくわしいことは文献[3]に記されているとおりである。この節では本文のシミュレータの働きを理解するのにぜひ必要なことだけを、ごく手短に説明しておく。

図１はＡＣＴ*による人の認知機構の基本的構成要素を示す。宣言的記憶（declarative memory）は、一般的な叙述的知識をたくわえる場所であって、ネットワーク構造をなしている。宣言的記憶はきわめて大量の知識をたくわえることができるが、そのときどきにアクセスできるのは、その一部に当たる作業記憶（working memory）の内容だけである。作業記憶の容量はごく小さい（これは人の情報処理機構に課せられた基本的制約の一つである）。

宣言的記憶の内容は、直接行動には結びつかない。行動のもととなるのはプロダクション記憶（production memory）の内容である。それは「作業記憶においてこれこれの条件が成り立っていたら、しかじかのことをせよ」という形のプロダクション（命令）の集まりである。プロダクション記憶においては、多数のプロダクションが互いに競合しながら、作業記憶内の知識と並列的に照合（matching）をおこない、最終的にはその時点の状況にもっともふさわしい一つが勝利を収めて発火（fire）する。発火したプロダクションは、知識を作業記憶につけ加える。

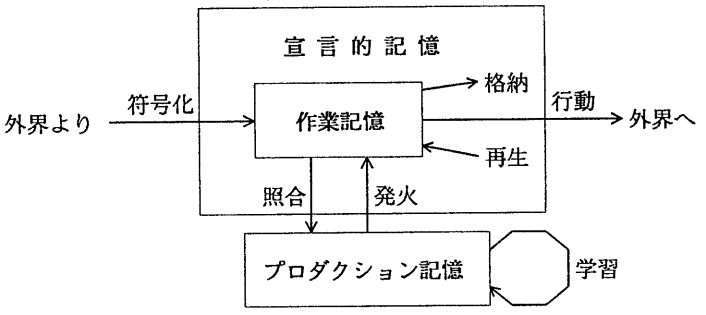


図１　ＡＣＴ*の基本的構成要素

外界からの刺激（たとえば見たもの、聞いたもの）は、符号化（encoding）されて作業記憶に入る。その結果として外界に対して行動（performance）が起こることもある。たとえばキーを叩いたり、「おはよう」といったりする、といった行動がである。

作業記憶に新しく入ってきた知識は、ある確率で宣言的記憶に恒久的に入り込む（格納 ── storeされる）。また宣言的記憶の中で眠っている知識は、プロダクションの働きによって再生（retrieve）される。宣言的記憶およびプロダクション記憶の各要素（節点）は、強さ（strength）と呼ばれる（長期的）パラメータを持っている。一般に強さは、その節点が使われれば使われるほど強まり、放っておかれると経過時間に反比例して徐々に弱まってゆく。

宣言的記憶およびプロダクション記憶の各要素は、強さのほかに活性値（activation）と呼ばれる（短期的な）パラメータを持つ。宣言的記憶についていえば、活性値は外界からの刺激によって与えられることと、プロダクションの発火の結果として与えられることがある。前者は刺激が存続する間、供給され続ける。後者は一般には１回限り供給されるものであるが、ゴール要素（goal element）と呼ばれる特別の節点（同時には１個だけ活性化）においては、それがゴールであることをやめるまでやまない。

活性値はネットワークの枝を伝わって、隣接の節点に伝播する。その際各隣接節点が受け取る活性値は、節点の強さに比例する。活性値は、ＲＣ回路に見られるような、指数型の減衰項を含む１階の微分方程式に従うものとされており、すばやく伝わり、供給がとだえれば急速に減衰する。作業記憶とは、宣言的記憶のうちの、その時点で活性値が０でない部分にほかならない。宣言的記憶の中に眠っていた情報をプロダクションによって取り出すことが可能なのは、活性値が伝播するお蔭である。

プロダクション記憶における活性値は、照合（図１）に際して作業記憶から流れ込み、データフローネットワークを伝わって伝播する。宣言的記憶とプロダクション記憶の本質的な違いは、前者では隣接節点からの抑制的入力が存在し、後者では存在しない、ということである。プロダクション記憶における抑制的入力の役割については、あとで実例を示して説明する。

プロダクション記憶は学習（learning）の機構を持つ。それを図１では、「学習」と付記した矢印で示している。使われるたびに強さが増すという前述の機構も、学習機構の一つである。このほか、宣言的記憶から照合によってある要素を取り出すプロダクションが頻繁に使われていると、もとのプロダクションにその要素をはじめから埋め込んでできた、新しいプロダクションが生成される（この過程を手続き化という）などの、いくつかの機構がある。

３．例題の説明

図２は文字が並んだものを見せられて、それがまともな英単語になっているかどうか判断する、という作業（lexical decision task）をしている人の頭の働きをシミュレートするためのプロダクション群（文献[3]の97ページ、表3.2を大幅に簡略化したもの）であって、本文のシミュレータへの入力に適する形で記述してある。

図において、「prod...if...then...end」という形をしているのがプロダクションである。最初のプロダクションは、「もし刺激（stimulus）が英単語であるかどうか調べることがゴールであるなら、その刺激に似た英単語があるかどうか探すというサブゴールlookupと、刺激が見つかった英単語の綴りに一致しているかどうか確認するというサブゴールverifyを立て、それが済んだらもとの『英単語であるかどうか調べる』というゴールは不活性化せよ」という意味を持つ。「lexical_decision」とあるのは、プロダクションの単なる名前である。また[100]とあるのは、プロダクションの強さの指定である。ここでは簡単なモデルでそれらしいシミュレーションを実現するために、プロダクションおよび記憶内容の強さを随所でやや便宜的に調節してある。

「LVstimulus」の「LV」は「局所変数」という意味を持ち、作業記憶の中の適当な要素に結合（bind）される。適当な要素が多数あれば、それぞれについてプロダクションのインスタンス（分身）が生成される。

サブゴールlookupは、競合する二つのプロダクションgood_candidateおよびbad_candidateを起動す

```
prod lexical_decision [100]
if goal(is_a_word(LVstimulus))
then
     subgoal(lookup(LVstimulus))
     subgoal(verify(LVstimulus))
     pop
end

prod candidate_verification [100]
if goal(is_a_word(LVstimulus))
   and candidate(LVword)
then
   subgoal(verify(LVstimulus))
   pop
end

prod good_candidate [100]
if goal(lookup(LVstimulus))
   and similar(LVstimulus, LVword)
then
   candidate(LVword)
   pop
end

prod bad_candidate [100]
if goal(lookup(LVstimulus))
   and not similar(LVstimulus, LVword)
then
   no_similar_word(LVstimulus)  # この処理は省略
   pop
end
```

```
prod good_spelling [100]
if goal(verify(LVstimulus))
   and candidate(LVword)
   and spell(LVword, LVstimulus)
then
   extern print(yes)
   pop
end

prod bad_spelling [100]
if goal(verify(LVstimulus))
   and candidate(LVword)
   and not spell(LVword, LVstimulus)
then
   extern print(no)
   pop
end

                                    # 以下辞書（仮）
similar({f, e, a, t, h, e, r}, feather) [100].
similar({f, e, a, t, h, c, r}, feather) [40].
similar({f, e, e, t, h, e, r}, feather) [40].
similar({h, e, a, t, h, e, r}, feather) [40].
similar({h, e, a, t, h, e, r}, heather) [100].
similar({h, e, a, t, h, c, r}, heather) [40].

spell(feather, {f, e, a, t, h, e, r}) [100].
spell(heather, {h, e, a, t, h, e, r}) [100].
```

図２　プロダクションの例 ― 刺激が英単語になっているかどうか調べる

る。プロダクションのこのような競合する対は、ＡＣＴ*においてyesかnoかを問う場面では絶えず出てくるものである。プロダクションgood_candidateは「ゴールがlookupであって、ある英単語LVwordが刺激語LVstimulusに似ていれば、その英単語は単語候補（candidate）である、と主張せよ」という内容を持っている。そうやって見つかった候補は、サブゴールverifyに関連して利用される。

一方プロダクションbad_candidateは「ゴールがlookupであってLVstimulusと似たLVwordが見つからなければ、候補はないと主張せよ」という内容を持つ。ここで面白いのは、このプロダクションのif節の「not similar」の「not」は本物の否定演算子ではない、ということである。「not」とついた条件を持つプロダクションは、つねに「not」とつかないプロダクション（ここではgood_candidate）と対になっており、「not」とついた条件節は実在せず、その代わりにプロダクション全体が対の相手と相互抑制関係に立つ。いまの場合、もしgood_candidateの方からsimilarなLVwordが見つかったという報告がいつまでも上がってこなければ、待ちかねたbad_candidateがしびれを切らして発火してしまう、という関係になっている。プロダクションgood_spellingおよびbad_spellingの関係も同様である。そこで使われている「extern print(...)」という形のものは、本来のＡＣＴ*にはないものであって、「...」のところのメッセージを端末に書き出す、という働きをする。

本来のlexical decision問題では、宣言的記憶に英単語に関する膨大な知識が入っていることが前提になるが、ここでは代わりに図２の最後の数行のような小細工で間に合わせることにした。すなわち単語としては、featherとheatherだけがあるものとし、そのさまざまな綴り誤りをsimilarという名の命題、という形で宣言的記憶におぼえ込ませた。ただし綴り誤りの程度に応じて、命題の強さを調節した。なおピリオドで終わっている行はシステムに対する指令であって、宣言的記憶の内容を調整する機能を持つ。「{f, e, a, t, h, e, r}」などは時系列（前述）である。この場では、"feather"という文字列のことだと思って差し支えない。

最後にプロダクションcandidate_verificationについて説明する。このプロダクションは「プロダクションlexical_decisionと同じ状況のもとで、もし

```
% act_goal.p lex.p ―― ①シミュレータの呼び出し

ACT system Ver. 3.0 Beta release

act[0]>message instance ―― ②以下毎回プロダクションのインスタンスを打ち出す
act[0]>message firedproduction ―― ③また毎回プロダクションの原型を打つ
act[0]> ―― ④コマンド入力時に単に改行すれば1ステップ進む
                    <<<< Changed Instances >>>> ―― ⑤さまざまなインスタンスができた
Instance#                    Name    Acti. ( Diff ) Upflow Evid.  MAX   Val
#1007f700(          lexical_decision) 1.413 (+1.413) +1.413 0.000  6+1   6+1
#1007f800(    candidate_verification) 1.842 (+1.842) +1.842 0.000  8+2   8+2
#1007f900(           good_candidate) 0.741 (+0.741) +0.741 0.000  9+2   3+2
#1007fa00(           good_candidate) 0.294 (+0.294) +0.294 0.000  9+2   3+2
#1007fb00(           good_candidate) 0.294 (+0.294) +0.294 0.000  9+2   3+2
#1007fc00(           good_candidate) 0.428 (+0.428) +0.428 0.000  9+2   3+2
#1007fd00(           good_candidate) 0.798 (+0.798) +0.798 0.000  9+2   3+2
#1007fe00(           good_candidate) 0.135 (+0.135) +0.135 0.000  9+2   3+2
#10070200(            good_spelling) 1.170 (+1.170) +1.170 0.000 11+2   5+2
#10070100(             bad_spelling) 0.471 (+0.471) +0.471 0.000  8+2   2+1
Sum of activation = 7.586
act[1]> ―― ⑥もう1歩進む。[...]の中はシミュレーションの抑制サイクル番号
                    <<<< Current Instances >>>> ―― ⑦活性値の奪い合いが起こっている
                Instace Name        Acti.( Diff ) Evid. Inc. (Diff )  Max  Val
#1007f700(          lexical_decision) 1.413(+0.000) 0.000 0.00 (+0.00)   7   6+1
#1007f800(    candidate_verification) 1.842(+0.000) 0.000 0.00 (+0.00)  10   8+2
#1007f900(           good_candidate) 0.741(+0.000) 0.000 0.00 (+0.00)  11   3+2
#1007fa00(           good_candidate) 0.294(+0.000) 0.000 0.00 (+0.00)  11   3+2
#1007fb00(           good_candidate) 0.294(+0.000) 0.000 0.00 (+0.00)  11   3+2
#1007fc00(           good_candidate) 0.428(+0.000) 0.000 0.00 (+0.00)  11   3+2
#1007fd00(           good_candidate) 0.798(+0.000) 0.000 0.00 (+0.00)  11   3+2
#1007fe00(           good_candidate) 0.135(+0.000) 0.000 0.00 (+0.00)  11   3+2
#10070200(            good_spelling) 1.170(+0.000) 0.000 0.00 (+0.00)  13   5+2
#10070100(             bad_spelling) 0.471(+0.000) 0.000 0.00 (+0.00)  10   2+1
Sum of activation = 7.586
act[2]>mes_in ―― ⑧message instanceと同じ。コマンド名は適宜省略可能。以下インスタンスの打ち出しを省略
act[2]>fire ―― ⑨以下プロダクションが一つ発火するまで続ける
Fired candidate_verification(#1007f800) [InhibitionCycle:9] ―― ⑩第9ステップでインスタンス1007f800が発火
Prod candidate_verification (#1007f800) [100.000] ―― ⑪それは強さ100
  IF goal[1](is_a_word[1](@@3{h, e, a, t, h, e, r})) ―― ⑫@@3は時系列の番号
    and candidate[1](feather)
  then...
                                    ―― ⑬そのインスタンスのもとになったプロダクション
PRODUCTION candidate_verification(#1004b080) [100.000000] 0
if
  goal(is_a_word(LVstimulus)) ... { goal[1](is_a_word[1](@@3{h, e, a, t, h, e, r})) }
  and candidate(LVword) ... { candidate[1](feather) }
then
  subgoal(verify(LVstimulus))
  pop
end

act[9]>fire ―― ⑭もう一つ発火するまで続ける
Fired bad_spelling(#1007f500) [InhibitionCycle:19] ―― ⑮19サイクル目にbad_spellingが発火した
Prod bad_spelling (#1007f500) [100.000]
  IF goal[1](verify[1](@@3{h, e, a, t, h, e, r}))
    and candidate[1](feather)
  then...

PRODUCTION bad_spelling(#1004b380) [100.000000] 0
if
  goal(verify(LVstimulus)) ... { goal[1](verify[1](@@3{h, e, a, t, h, e, r})) }
  and candidate(LVword) ... {  }
  and not spell(LVword, LVstimulus) ... { spell[100](heather, @@3{h, e, a, t, h, e, r}) |  }
then
  extern print(no)
  pop
end
no ―― ⑯extern print(no)からのメッセージ
act[19]>quit ―― ⑰実行終了、ＯＳに戻る
%
```

図3　シミュレータの動作例 ― 刺激が英単語かどうか答える

candidateがすでに存在していたら（ある単語について命題candidateが成り立っていたら）lookup抜きの処理をせよ」という意味を持っている。もし事前に作業記憶の中に命題candidateを満たす単語が入れてあった（たとえば被験者に、前もって羽の絵を見せ、さてはfeatherだなと思わせるなどの方法によって）とすると、このプロダクションの前提条件は、処理の開始時点で完全に満たされることになる。したがってこのプロダクションは、第1のプロダクションと競合関係に立つ。

そのような場合には前提条件がたくさんあるプロダクションの方が強いことになっているので、この場合はcandidate_verificationが勝ち、ただちに綴りの確認フェーズがはじまることになる。

4．シミュレータの動作例

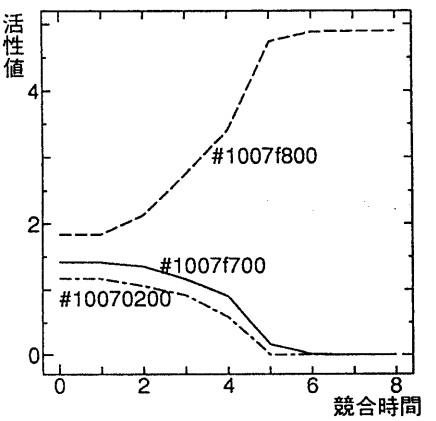
図3は本文のシミュレータの動作例である。下線部が利用者の打ち込んだもの、それ以外はシミュレータが打ってきたものであり、①②③……は注釈である。なおあらかじめ図2のプロダクションがファイルlex.pに、また次のような指令がファイルgoal.pに入れてあるものとしている。

```
setgoal(is_a_word({h, e, a, t, h, e, r}).
wm candidate(feather).
```

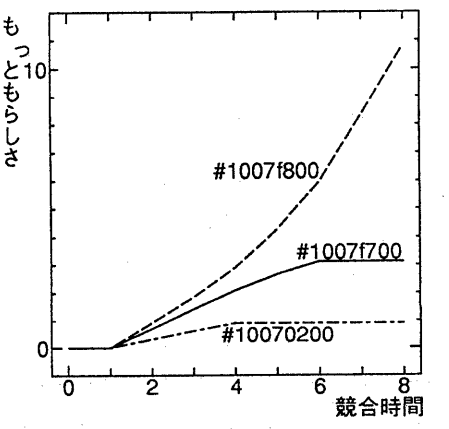
これで "heather" という文字列が英単語かどうか調べることがゴールとなり、またコンピュータは「さてはcandidateはfeatherだな」と思い込むことになる。

図において全体の筋書きは、まずシミュレータを起動し、シミュレーションを1歩進め、プロダクションcandidate_verificationが第5ステップで発火するまで途中を飛ばし、さらに第12ステップでプロダクションbad_spellingが発火するまで進行し、結果「no」が打ち出されたところで終了する、というものである。「羽の絵を見せられ」たために、「heather」（ギョリュウモドキ）という正しい英単語を見せられたにもかかわらずまんまとだまされ、それは英単語ではないと口走った、という結果になっている。途中messageコマンドを打ち込んで、インスタンスの自動的な打ち出しを起こしたり止めたりしている。

インスタンス間の競合は、互いに活性値（図3にActi.とある数字）を奪い合い、その結果各インスタンスの、ある基準によって算定された「もっともらしさ」（evidence ― 図のEvid.）が一定の境界値を越えるまで続く。その様子を図3の第1～8ステップについて図4に示す（すべてを示すと複雑になりすぎるので主要なものだけを示す）。



(a)活性値



(b)もっともらしさ

図4　インスタンスの競合における活性値(a)ともっともらしさ(b)の変遷

5．基本機能の実現

シミュレータの実現方法について詳細を記すことは、ページ数の関係で不可能である。本節と次節では、これまでの記述からは自明でないような考慮点を2、3点描的に示しておくことにしたい。

時間の扱い　ＡＣＴ*の働きは、公式には時間微分を含む微分方程式によって規定されており、時間軸は連続である。しかし実際の計算では、微分をある程度差分で近似せざるを得ない。すなわち適当な時間刻みを導入することが必要である。

Anderson[3]もそうしているようにわれわれは、活性値の伝播は短い時間内に迅速に起こってすみやかに平衡に達すると仮定し、この短い時間を基本的時間刻み（**抑制サイクル** ── inhibition cycle）として採用することにした。プロダクション間の競合に関する計算は、抑制サイクルを単位としておこなう。その際活性値については、本来ならば無限の時間を要して達成されるはずの平衡値を用いる。

発火の条件　プロダクションは一定の方程式に従って活性値（図3のActi.の欄参照）を取り合い、そのある種の積分値が状況に応じて定まる、ある境界値を越えたときに発火する。この積分値が、「もっともらしさ」（evidence、図3のEvid.）である。境界値は、実際に作業記憶の内容と結合した局所変数の個数を局所変数の総数で割った比など、いくつかの要素に依存して定まる。宣言的記憶内の活性値については一定のカットオフ値を設け、その値未満の活性値はプロダクションのインスタンスの生成には寄与しないものと見なしている。カットオフ値を一定としているために、ときたま非常に多くのインスタンスを考慮しなければならくなることがある。

データフローネットワーク　プロダクションの競合は、原著[3]では多階層のデータフローネットワークによってなされることになっているが、計算時間の節約のため、ここでは1階層のネットワークで近似している。このため、競合が一方的に傾きやすいという傾向が出ている。この点は今後の課題である。

6．学習機能

このシミュレータは、ＡＣＴ*の学習機能をある程度忠実に実現している。すなわちプロダクションの**合成**（composition）、**手続き化**（proceduralization）、**一般化**（generalization）、**個別化**（discrimination）、および**強化**（strengthening）が、簡略化した形ではあるものの、一応実現されている。図5に図3の過程で学習によってできた新しいプロダクションの例を示す。これはプロダクションcandidate_verificationにおいて、LVstimulusを文字列"heather"と、またLVwordを単語featherと置き換えてできた手続き化プロダクションであって、できたてでまだ実績がないので、1という弱い強さが与えられている。ただし上述のように、現状では実現は簡略化されたものであって、文献[3]に記述されたとおりではない。たとえば手続き化において、もとのプロダクションに「もしLVstimulusが6文字以上であれば」という記述があり、LVstimulusに（図5の場合と同様）"heather"が結合したとすれば、この条件はつねに成立するから省いてしまってよいはずであるが、そのような書き換えを一般的な機構によって可能にすることはきわめて困難であるので、それはいまのところ手つかずになっている。合成、一般化、個別化、強化についても、それぞれむずかしい問題がある。

また文献[3]には以上のほか、宣言的知識の解釈実行およびアナロジーに基づくプロダクションの生成が挙げられているが、これらについても一般的取り扱いがむずかしいため、まだ手つかずの状態である。

ＡＣＴ*に関する基本文献[3]の大部分は、言葉に

```
PRODUCTION proc_of_candidate_verification(#10060c00) [1.000000] 0
if
  goal(is_a_word(@@@{h, e, a, t, h, e, r})) ... {  }
  and candidate(feather) ... {  }
then
  subgoal(verify(@@@{h, e, a, t, h, e, r}))
  pop
end
```

図5　プロダクションの手続き化（図3の第12ステップで「print production」指令により出力）

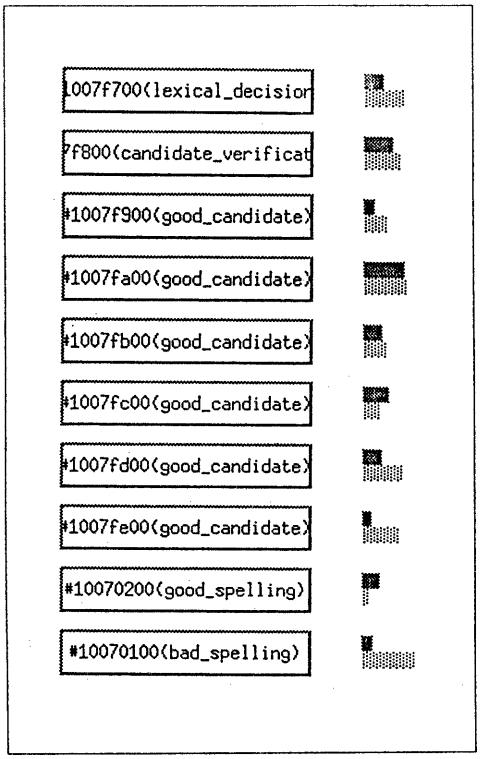


図6　グラフィック表示の例

よる説明から成っており、ところどころ解釈が必ずしも自明でないように見える場所がある。プロダクションの学習に関しては、それが特に著しい。今後とも検討の必要がある。

7．今後の問題および付言

本文のシミュレータはやっと使いものになりはじめた段階であり、まだこれから調整が必要である。ＡＣＴ*には多数のパラメータがあり、よいシミュレーション結果を得るためには、それらをうまく決めてやる必要がある。しかしパラメータの調整を恣意的におこなうことは、結果の信用性を失うもとである。なるべく多くの問題に適合するような、パラメータのただ一つの組を見つけ出し、固定することが望ましい。それにはもっと多くシミュレーション経験を積む必要がある。

図3のインタフェースは、率直にいってやや古風である。もっと現代風に、グラフィック表示を取り入れることが望ましい。その方向の仕事は現在進行中である。たとえば図6のような、競合するインスタンスの活性値ともっともらしさを示すグラフを出力する機能は一応できている。

システムはCで記述した。その規模は、1992年2月現在、シミュレータ本体が20000行、キーボード用の手のシステムが2800行となっている。現在主としてSONY Newsワークステーション上で使用している。占有記憶容量は最小1.5ＭＢ程度であるが、問題が大規模になってプロダクション数がふえてくると急速に膨張し、25ＭＢ程度に及ぶことがある。

謝辞

このシミュレータの開発では、前年度に村本英治氏（ＡＣＴ*部分）および深澤安伸氏（手の機構）が作られた原型を利用した。下郡信宏氏はシミュレータ開発のマネージャ役を務めてくださった。また図6に示したグラフ表示は浦野幹夫氏が作成してくださった。感謝する。

本研究に関し、文部省科学研究費補助金重点領域研究(1)「高機能高品質ソフトウェアの評価法の研究」（研究代表者牛島和夫）による補助を受けた。

参考文献

1. Card, Stuart K., Thomas P. Moran and Allen Newell: The Psychology of Human-Computer Interaction, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983.
2. Newell, Allen and Stuart K. Card: The Prospects for Psychological Science in Human-Computer Interaction, HCI, Vol.1, No.3 (1985), pp.209-242.
3. Anderson, John R.: The Architecture of Cognition, Harvard University Press, Cambridge, Mass., 1983.
4. 木村　泉: ＱＷＥＲＴＹローマ字打ちとＳＫＹ配列の相互干渉, 情報処理学会研究報告 Vol.91 No.18, HI研究会 91-HI-35 (Mar. 1991), pp.59-66.
5. Anderson, John R.: Personal communication.