エキスパートシステム

茂野 真弓

2008 年 10 月 29 日

目 次

1　エキスパートシステム

エキスパートシステムとは、ある分野の専門家 (一人または複数人) が持つ知識を利用して推論を行い、専門家に近い判断を下すためのシステムである。

エキスパートシステムに知識を与えるのは専門家であり、専門家ではないユーザは問題を解決するためにエキスパートシステムを利用する。このように、エキスパートシステムには、知識を与える専門家とエンドユーザの 2 種類のユーザがいる。

専門家がシステムに与える知識は、システムが扱える形式のプログラムである必要がある。ただし専門家がプログラマであるとは限らない。そこで、プログラマではない専門家がどのような形式で自分の知識をシステムに入力すればよいのかが問題となった。このことから、エキスパートシステムは

- 専門家の知識を扱かえる表現法を見つけること
- 専門家の知識を利用した解法を見つけること

を目的として研究された。

専門家の知識を表現する方法として、Prolog の fact や rule が考えられる。しかし、Prolog は一般的な知識ベースのシステムとしては次の部分が欠けている。

不確かさを用いての推論
: Prolog の世界には真/偽しかない。計算が真である間は計算を続けるが、偽になったとたん計算を打ち切ってしまうので、偽についての推論はできない。それに対し、専門家は「〜のような」や「90%確かである」などの曖昧な表現をすることがある。

理由の説明
: Prolog は入力された質問に対する解を出力するが、その解がどのように求められたものかは出力しない。ユーザに解決方法を説明できるシステムは信用度が高くなる。

制御構造の定義
: Prolog は質問が入力されたら、あとは後方連鎖でルールを使って勝手に解を見つけにいく。知識ベースのシステムでは、問題によっては様々な制御が必要になる。例えば、医療診断を行うエキスパートシステムでは、患者に関する情報を得るのにルールを使うのではなく、医療診断を行う医者に対して質問する必要がある。このように、後方連鎖でルールを使う以外の方法で情報を得る手段が必要になる。

初期のエキスパートシステムは様々な手法でこれらの問題に取り組んだ。最終的には専門家から知識を得て、その知識を問題解決に使い、さらに推論の説明を行うのに特化したプログラミング環境を構築し、それをエキスパートシステムの核とするという手法がよく利用されるようになった。

2　EMYCIN

2.1　MYCIN と EMYCIN

MYCIN は初期のエキスパートシステムのひとつであり、血液感染細菌の抗生物質治療を行うために設計された。次に示すのは、MYCIN で使用されるルールのひとつである。

```
(defrule 52
  if (site culture is blood)
     (gram organism is neg)
     (morphology organism is rod)
     (burn patient is serious)
  then .4
     (identity organism is pseudomonas))
```

ルールの形式は、ルールを定義する際に使うマクロである defrule から始まり、ルールの番号、if、前提、then、certainty factor、結論という順である。上記の例のように、ルールの前提が複数個並ぶことがある。また、ルールの結論も同様に複数個並ぶことがある。certainty factor はルールの信頼度にあたるものである。よって上記のルールは、「培地が血液かつ、細菌のグラム染色による分類の結果がネガティブかつ、細菌の形状が棒状かつ、患者の痛みがひどい場合、細菌は pseudomonas である (信頼度 0.4)」ということを表す。

MYCIN がこの上記のルール 52 から細菌が pseudomonas であるという結論を出すには、ルール 52 の前提を満たす必要がある。EMYCIN では、前提を満たすための手段として、各前提についてルールを使う方法 (後方連鎖) だけではなく、ユーザに質問するという方法もある。
例えば、ルール 52 の前提を満たすためには

Q. 培地はどこ?

A. 血液

Q. 細菌のグラム染色による分類の結果は?

A. ネガティブ

Q. 細菌の形は?

A. 棒状

Q. 患者の痛みはひどいか、ひどくないか?

A. ひどい

というやり取りがシステムとユーザの間で成され、ユーザの回答から pseudomonas という結論が得られる。
上の例ではユーザの回答はルール 52 の前提を満たしているが、ユーザの回答がルール 52 の前提を満たさない場合は、ルール 52 の計算を打ち切って他のルールを使った計算が開始される。

システムが返してくる結論は、「結論 A の 信頼度は 0.6, 結論 B の信頼度は 0.3」という複数の結論とその信頼度を求めることができる。これは人間が「結論 A である確率は 60%, 結論 B である確率は 30%」というような複数の候補を推論できるのと同様である。

また、計算中のルールを簡単な英語に変換して出力させる機能がある。この機能を使うと、ユーザはどのルールで推論を行っているのか確認することができる。例えば上記のルール 52 で推論を行っている場合にこの機能を使うと、

```
Rule 52:
  If
    1) THE BURN OF THE PATIENT IS SERIOUS
    2) THE GRAM OF THE ORGANISM IS NEG
    3) THE MORPHOLOGY OF THE ORGANISM IS ROD
    4) THE BURN OF THE ORGANISM IS SERIOUS
  Then there is weakly suggestive evidence (0.4) that
    1) THE IDENTITY OF THE ORGANISM IS PSEUDOMONAS
```

が出力され、ルール 52 の前提、結論、信頼度が表示される。

EMYCIN は MYCIN が扱う分野に依存しない部分を抽出して形成されたエキスパートシステムの枠組であり、特定の分野の知識を与えられることでその分野のエキスパートシステムとして動作する。例えば、EMYCIN に MYCIN 用の知識を与えると、MYCIN が動作するようになる。

前述したとおり、エキスパートシステムのユーザには専門家とエンドユーザという 2 種類のユーザがおり、それぞれエキスパートシステムとの関わり方が異なる。
専門家は、知識を持たない EMYCIN にシステムの動作や推論のためのルールといった知識を与え、ある分野に特化したエキスパートシステムを定義するという役割を持つ。
これに対しエンドユーザは、専門家によって定義されたエキスパートシステムを問題を解くために利用する。ただし、エンドユーザもシステムにある種の情報を与える役割を持つ。その情報とは、問題ごとに異なるものであり、専門家があらかじめシステムに与えておくことができないものである。例えば、これから治療する患者についての情報 (性別や年齢など) は問題ごとに異なり、普通はあらかじめシステムに与えることができない。これらの情報については、システムがエンドユーザに質問してくる。エンドユーザは質問に回答することで、システムに情報を与える。

2.2 EMYCIN の仕組み

EMYCIN は Prolog と同様に後方連鎖でルールを計算していく。しかし、Prolog とは次の点で異なる。

不確かさの扱い

全てのルールが真か偽ではなく、ルールと不確かさを示す certainty factor (以下 cf と呼ぶ) というものを関連づけることで不確かさを扱う。

計算結果のキャッシュ

計算結果をキャッシュするので、同じ計算を繰り返さない。Prolog では計算結果のキャッシュを行わないため、どんなに大変な問題でも同じ計算を繰り返す。

ユーザへの質問

計算を行うために必要な情報を得るために、システムがユーザに質問する手段がある。

説明機能

システムがどのような計算をしているのか、ユーザに説明する機能がある。

コンテキストの利用

Prolog では論理変数を利用して計算するが、EMYCIN では代わりにコンテキストというものを利用する。

ここでは、EMYCIN と Prolog の違いを中心にして EMYCIN の仕組みを説明する。そして最後に EMYCIN に MYCIN の知識を与え、ユーザがどのように EMYCIN に推論をさせるかの例を見ていく。

2.2.1 不確かさの扱い

EMYCIN は true, false という真偽値ではなく、cf と呼ばれる値を扱う。cf は `-1` から `+1` の間の値である。`1` は真を表し、`1` に近いほど信頼度が高い。`-1` は偽を表し、`-1` に近いほど信頼度が低い。`0` は、真よりなのか偽よりなのかが全くわからないこと、つまり信頼できるかどうかが全くわからないことを表している。cf の理論を定義するには、and, or のような論理演算が必要である。まず最初にどのように 2 つの cf を結合してひとつの cf にするかを説明する。

患者が X という病気を持っている可能性を求めるという例で考えてみる。2 つの病気の検査 A, B を浮けた人たちがいるとする。検査 A では 60%の人が病気 X であるという結果で、検査 B では 40%だった。この 2 つの結果をひとつに結合するには、2 つの検査結果の依存関係がわからなければ、正確な解が出せない。
例えば検査 A が病気 X であると判定した人たちの集合を集合 A, 同様に検査 B が病気 X であると判定した人たちの集合を集合 B としたとき、2 つの集合に共通部分がなければ、テストを受けた全ての人が病気 X であると判定されたことになるので、100%の人が病気 X だということになる。一方、集合 B が集合 A の部分集合である場合は、病気 X である人数は集合 A に含まれる人数と一致するため、60%の人が病気 X だということになる。

EMYCIN で実際に使われている cf の結合方法は次のとおりである。

```
combine(A, B) =

      A + B - AB; (A, B > 0)
      A + B + AB; (A, B < 0)
      A + B / (1 - min(|A|, |B|)) (otherwise)
```

この公式を使うと、先ほどの問題は `combine(0.60, 0.40) = 0.76` となる。この結果は 0.6 と 1.0 との間の値であることがわかる。また、この結果は A と B が独立しているときの確率 `p(A or B)` と同じである。
しかし、cf は確率とは別のものである。確率では 2 つの事象の依存性や独立性を考慮する必要があるが、cf では依存性や独立性を考慮せずに上の公式を使って cf を結合する。
信頼度が最も高いときの cf は 1 であり、true と表す。信頼度が最も低いときの cf は -1 であり、false と表す。信頼できるのか、できないのか全くわからない場合の cf は 0 であり、unknown と表す。次に示すのは cf の結合関数の特徴である。

- 必ず -1 から +1 の間の数値を計算する
- unknown との結合は、元の値がそのまま結果になる
- true との結合は true (ただし、false は除く)
- true と false の結合はエラー
- 正負がぴったり逆の値を結合すると、unknown になる
- 両方とも正の値であれば、より大きな正の値になる (ただし true は除く)
- 正の値と負の値の結合は、２つの値の間の値になる

ここまでは、A ならば C、B ならば C という２つのルールが持つ cf を結合する方法を見てきた。つまり、次のような場合の cf の結合方法である。

```
A => C
B => C
```

ここで、ルールの前提部分の cf がどのように決定されるのか、つまり、上記のようなルールの A, B の cf がどのようにして決定されるのかを説明しておく。EMYCIN がゴールに合ったルールを適用するとそのルールの前提部分もゴールとして扱われることになる (後方連鎖)。それぞれの前提が満たされるかどうかは、ルールを適用するか、またはユーザに質問するかのどちらかで計算される。例えば、前提 A をルールを適用して計算する場合、適用したルールから得られた cf が前提 A の cf となる。前提 B をユーザに質問して計算する場合、ユーザの回答から前提 B の cf を決定する。ユーザの回答には、(1) 値だけ、(2) 値とその cf 、という 2 通りの形式があり、(1) の場合の cf は 1 として扱われる。(2) の場合は、ユーザが回答した cf が前提 B の cf となる。これ以降に登場するルールの前提の cf は、ここで説明した方法で決定される。

では、A かつ B ならば C、という場合の cf の結合を考える。つまり、次のようにひとつのルールに前提が複数個あり、各前提の cf を結合する方法である。

```
A and B => C
```

この場合、EMYCIN は A と B の cf の最小値を結合結果とする。

また、ルール自体の信頼度というものもある。EMYCIN は次のようなルールを使うことができる。

```
A and B => .9C
```

これは、「A かつ B ならば C。その cf は 0.9」であることを表す。EMYCIN は単純に `A and B => C` の cf を求め、さらにルール自体の cf を掛けたものを全体の cf とする。上記の例では、A,B の cf がそれぞれ 0.6、0.4 だとすると、A and B の cf は 0.4 なので、0.9 と掛けて 0.36 が得られる。次に 0.36 と C の cf を結合する。もし C の cf が 0、つまり C について全くわかっていない場合は、結合結果として 0.36 が得られる。もし C の cf が 0.76 ならが、結果は

```
0.36 + 0.76 - (0.36 * 0.76) = 0.8468
```

となる。

EMYCIN は cf を結合しながらルールを後方連鎖で計算していくシステムである。もしシステムが扱う cf が 1, -1 の２つだけなら、EMYCIN は Prolog と同じ動きをする。-1 と 1 の間の cf を扱うときだけ、EMYCIN は Prolog と違う動作をするのである。

Prolog では、真偽値を２の目的で使っている。true はルールでの計算が成功したことを表す”yes”という解を得るため、false は探索の打ち切りを決定するためである。探索の打ち切りとは、もしルールの前提の中にひとつでも false があれば、ルールのそれ以降の前提を計算するのは 無意味なので、そこで計算を終えることである。
EMYCIN は、探索の打ち切りをする基準が Prolog とは異なり、ルールの前提部分の cf が 0.2 以下になると false だと見なして探索を打ち切る。この理由は、EMYCIN で Prolog と同様に false(-1) のときだけ探索を打ち切るとすると、大量のルールを使って計算しなければならず、しかも得られる結果は cf(信頼度) が小さいもの、つまりあまり意味のないものが多くなってしまうからである。

2.2.2 計算結果のキャッシュ

Prolog は同じ質問が２回されると、どれだけ大変な問題でも、２回とも同じ計算をする。一方、EMYCIN は１回目は普通に計算し、２回目からは計算せずに１回目で得られた計算結果をとってくるだけである。つまり、EMYCIN は同じ計算を繰り返さない。計算結果のキャッシュはハッシュテーブルで実現された DB を使って行われる。

EMYCIN はコンテキストと呼ばれるもののインスタンスと、それが持つパラメータ (属性) を扱うように設計されている。コンテキストとパラメータは専門家が定義する。コンテキストとは推論を行う対象となる現実世界の実体である。複数の種類のコンテキストについて推論を行うには、それぞれ別のコンテキストが定義する必要がある。また、ひとつのコンテキストに対して複数個のパラメータを定義できる。EMYCIN では推論対象であるコンテキストのパラメータの値を求めながら計算を進めていく。あるコンテキストのあるパラメータの値が求められると、そのコンテキストのパラメータの値が「計算結果のキャッシュ」で説明した DB に登録され、同じコンテキストの同じパラメータについては計算を繰り返さない。また、パラメータの値だけでなくそのパラメータの値が既にわかっていること、ユーザに質問して値がわかった場合はユーザに質問済みであることも DB に登録される。このようにコンテキストは、システムが行っている推論がどこまで進んでいるかという、システムの推論の状態を表す。

例えば、MYCIN の場合は patient(患者) というコンテキストとそのパラメータとして name が専門家によって定義されている。patient のインスタンスは、それぞれ name パラメータを持っている。この name というパラメータについてはエンドユーザが値を知っていると考えられる。name パラメータのようにエンドユーザが値を知っているパラメータについては、ユーザに質問することで値を得るように専門家によって定められている。システムはユーザに質問すると、ユーザの回答結果を推論中のコンテキストのパラメータ値として DB に登録する。さらに、そのコンテキストのパラメータ値を質問したこと、そのコンテキストのパラメータ値が判明していることという 2 つの項目も DB に登録しておく。こうしておくと、推論を続けて同じ patient コンテキストの name パラメータの値を求めることになった場合に、同じ質問をする前に DB を確認して、質問したこ

とが登録されている場合は、質問しない。つまり、同じ質問が繰り返さないようになっている。

また、 MYCIN には organism(細菌) というコンテキストとそのパラメータとして identity がある。この identity というパラメータについては、普通はエンドユーザが値を知らない。エンドユーザが値を知らないパラメータについては、ルールを適用することで値とその信頼度を求めるよう、専門家によって定められている。システムは、ルールを適用して得られた結果 (値と cf) を推論中のコンテキストの値として DB に登録し、さらにそのコンテキストのパラメータ値が判明していることも DB に登録しておく。すると、次にそのパラメータ値を得ようとした場合に、パラメータ値が判明していることが登録されているか DB を確認し、登録されていれば、そのパラメータ値を求める計算をしない。つまり、同じ計算を繰り返さないようになっている。

ちなみに、パラメータの値とその cf は、`((値１ cf1)（値２ cf2）...)` という形式で、DB に登録される。

以上のように、 DB には判明したインスタンスのパラメータ値とその cf、パラメータ値が既にわかっていること、パラメータ値についてユーザに質問したことなど、問題のインスタンスについて推論した情報が全て登録される。例えば MYCIN では、患者について知りうる全ての情報がデータベースに登録される。一度システムを終了してからシステムを再起動すると、データベースは初期化される。
ただし、システムの動作を決定する情報については、このデータベースには登録しないため、情報が失われることはない。次の 3 つがシステムの動作を決定する情報である。

- エキスパートが定義した全てのルール
- パラメータとなる構造体。パラメータの構造体はパラメータ名をもち、そのパラメータ名でインデックス付けされる。
- 何のコンテキストをどの順で推論するかを表すコンテキストのリスト

2.2.3 ユーザへの質問と説明機能

EMYCIN ではルールから解をみつけられなかったとき、ユーザに質問して解を得る機能がある。Prolog でも質問を出力し、その回答を入力として受け取るようなルールを書くことができるので、この違いは根本的なものではない。EMYCIN では知識ベースの設計者が定義した簡単な質問文を使って質問を行う。もし質問文が定義されていなければ、デフォルトの質問文が使われる。

また、ユーザはシステムがしてきた質問に対して逆に質問することができる。ユーザが入力できる質問と、システムが返す返答は以下のとおりである。

?: ユーザに期待する解の候補を出力する

rule: 計算中のルールを簡単な英語の文章に変換して出力する

why: rule と同様に計算中のルールを出力するが、前提のどこまで計算されているか、これからどの前提を計算するのかという詳細を出力する

help: ヘルプ用文字列を出力する

why を入力したときは、rule のときと同じように計算中のルールが出力されるが、計算の過程がわかる形式で出力される。まず、ルールの前提部分のうち既に値がわかっているものがどれなのかを出力し、続けてルールの残りの部分を出力する。これにより、どこまで計算ができていて、これから何を計算するのかをユーザに説明できる。

2.2.4 コンテキスト

コンテキストは推論を行う対象となる現実世界の実体であり、EMYCIN はコンテキストが持つパラメータ (属性) の値を求めることで計算を進めていく。前述したとおり、あるコンテキスト A のあるパラメータ a の値をユーザに質問して得た場合は、ユーザの回答結果をコンテキスト A のパラメータ a の値として登録し、さらにコンテキスト A のパラメータ a の値をユーザに質問したこと、コンテキスト A のパラメータ a の値が判明したことを DB に登録する。ルールを使って求めた場合は、ルールから得られた結果をコンテキスト A のパラメータ a の値として登録し、コンテキスト A のパラメータ a の値が判明したことを DB に登録する。このように、コンテキスト A について推論した全ての情報が DB に登録される。つまり、コンテキストはシステムが行っている推論がどこまで進んでいるかという推論の状態を表す。

コンテキストはパラメータを持ち、パラメータは値とその cf(信頼度) を持つが、値と cf は推論前にはわかっていない。Prolog では、論理変数の値をルールを unify していくことで決定する。EMYCIN では、コンテキストのパラメータの値をルールを適用するか、またはユーザに質問することで決定する。どちらの方法で値を決定するかは、専門家によってコンテキストの各パラメータごとに定義されている。例えば MYCIN では、patient(患者) というコンテキストのパラメータ name はユーザに質問することで値を決定するよう定義されており、organism(細菌) というコンテキストのパラメータ identity についてはルールを適用していくことで値を決定するよう定義されている。

普通は推論対象となるコンテキストの種類は複数個あり、EMYCIN にそれらのコンテキストをリストにして渡すと、それらのコンテキストについて順に推論を行うエキスパートシステムを作成することになる。例えば、MYCIN を作る際は、推論を行う対象である patient(患者)、culture(培地)、organism(細菌) の 3 つのコンテキストを定義して EMYCIN に渡す。

次は、MYCIN で patient、culture、organism の 3 つのコンテキストが出現するルールの例である。

```
(defrule 52
    if (site culture is blood)
       (gram organism is neg)
       (morphology organism is rod)
       (burn patient is serious)
    then .4
       (identity organism is pseudomonas))
```

システムは各コンテキストの最新のインスタンスを、コンテキスト名 (シンボル) をキーにして DB (計算結果のキャッシュに使う DB と同じ) に登録しておく。こうしておくことで、例えば上記のルール 52 の `(site culture is blood)` の部分を計算するとき、コンテキスト名 culture(シンボル) をキーにして DB から culture の最新のインスタンスを取得して、このインスタンスの site パラメータの値を計算することになる。同様に `(gram organism is neg)` の部分を計算するとき

には organism(シンボル) をキーにして organism の最新のインスタンスを取得し、gram パラメータの値を計算する。

コンテキストのインスタンスはコンテキストの種類によって patient や organism のような名前を持ち、さらに番号づけられる。この番号はそのコンテキストの最新のインスタンスの番号を表す。また、コンテキストのパラメータには２種類ある。ひとつは、ユーザが最初から知っているパラメータである。例えば、医者は普通は患者の名前、年齢、性別などを知っている。これらのパラメータについては、ユーザに質問することで値を決定する。もう一つは、普通はユーザが知らないパラメータであり、そのパラメータ値はルールを適用して後方連鎖のプロセスを通して決定される。

MYCIN では、patient, culture, organism という順で推論を行い、前述したようにそれぞれの最新のインスタンスが DB に登録される。例えば、patient, culture, organism について順に推論を行っている状態で、それぞれのインスタンスを patient-1, culture-1, organism-1 と表すとする。この状態は、「ひとりめの患者 (patient-1) のひとつめの培地 (culture-1) におけるひとつめの細菌 (organism-1) について推論する」ことを表す。この状態から「ひとつめの細菌」についての推論が終わると、「ふたつめの細菌」(つまり他の細菌) についての推論をすることができる。つまり、「ひとりめの患者 (patient-1) のひとつめの培地 (culture-1) におけるふたつめの細菌 (organism-2) について推論する」ことができる。「ひとりめの患者 (patient-1) のひとつめの培地 (culture-1)」における全ての細菌についての推論が終わると、今度は「ひとりめの患者 (patient-1) のふたつめの培地 (culture-2)」における細菌についての推論を同様に行うことができる。このように、コンテキスト間の関係は patient を根にした木構造になる。

2.2.5　後方連鎖

EMYCIN は Prolog と同様に与えられたゴールに合うルールを適用する。ルールを適用すると、ルールの仮定部分もゴールとして扱い、再帰的にルールを適用していく。

ただし、EMYCIN と Prolog の後方連鎖は全く同じではない。Prolog で 適用するルールは、その head 部が ゴールと unify するものである。あるルールで unify が成功すれば、ゴールが真となる。このため、Prolog では深さ優先で後方連鎖していく。
EMYCIN では成功するルールのうち、同じ結論を持つルール全ての cf を結合して最終的な cf とする。つまり、ゴールに合うルールを全て適用して、成功するルールの cf を求めてからでないと、最終的な cf が求められない。このため、EMYCIN は幅優先で後方連鎖する必要がある。

EMYCIN は次の流れでインスタンスのパラメータ値を求め、DB に登録する。

パラメータの定義によって異なるが、ユーザに質問するか、ルールを適用するかしてパラメータ値を得る。最初に実行したほうで解が得られなかった場合は、もう一方を試す。ルールを適用する場合は、値を得ようとしているパラメータに関する全てのルールをそれぞれ使って計算する。cf が閾値 (0.2) を越えるパラメータ値がひとつでも見つかれば、探索は成功である。

ルールの計算は、各前提を順番に後方連鎖またはユーザに質問することで計算していき、得られた cf を and 結合して、閾値を越えていれば成功となる。成功したら、得られた値と cf をペア (普通は複数個のペア) にしてリストを作り、DB に登録する。もしも同じ値が得られるルールが複数個あった場合は、結合関数を利用してそれらの cf を結合し、結果をその値の cf とする。また、「計算結果のキャッシュ」で説明したとおり、一度求めたインスタンスのパラメータ値については、求めたことを DB に登録しておく。次に同じパラメータ値を求める場合は、DB を確認して求めたこ

とが DB に登録されていれば計算しないようになっている。つまり、同じ計算が繰り返されない。

2.3 MYCIN の使用例

ここで EMYCIN に MYCIN 用の知識を与え、MYCIN を利用してみる。まず MYCIN で使用する 3 つのコンテキスト patient, culuture, organism を EMYCIN に与えることで、MYCIN の推論の制御を定義する。次で定義する関数 mycin を呼び出すと、MYCIN が起動する。

```
(defun mycin ()
  "Determine what organism is infecting a patient."
  (emycin
    (list (defcontext patient  (name sex age)  ())
          (defcontext culture  (site days-old) ())
          (defcontext organism ()              (identity)))))
```

EMYCIN に与えられたコンテキストのリストは、リストの順にパラメータ値を計算することになる。

defcontext の各引数は、第一引数がコンテキストの名前、第二引数が前もって値がわかるであろうパラメータ名のリスト、第三引数がゴールとなるパラメータ名のリストである。返り値はコンテキストの構造体である。上記の MYCIN の定義では、まず patient について name, sex, age の値を求め、次に culture について site days-old の値を求める。そして organism について identity の値を決定するための推論を始めることになる。

次に、パラメータの定義をする。

```
;;; Parameters for patient:
(defparm name patient t "Patient's name: " t read-line)
(defparm sex patient (member male female) "Sex:" t)
(defparm age patient number "Age:" t)
(defparm burn patient (member no mild serious)
  "Is ~a a burn patient?  If so, mild or serious?" t)
(defparm compromised-host patient yes/no
  "Is ~a a compromised host?")

;;; Parameters for culture:
(defparm site culture (member blood)
  "From what site was the specimen for ~a taken?" t)
(defparm days-old culture number
  "How many days ago was this culture (~a) obtained?" t)

;;; Parameters for organism:
(defparm identity organism
  (member pseudomonas klebsiella enterobacteriaceae
          staphylococcus bacteroides streptococcus)
```

```
  "Enter the identity (genus) of ~a:" t)
(defparm gram organism (member acid-fast pos neg)
  "The gram stain of ~a:" t)
(defparm morphology organism (member rod coccus)
  "Is ~a a rod or coccus (etc.):")
(defparm aerobicity organism (member aerobic anaerobic))
(defparm growth-conformation organism
  (member chains pairs clumps))
```

defparm は最大 6 個の引数をとる。第一引数はパラメータ名、第二引数はどのコンテキストのパラメータであるか、第三引数はパラメータの値の型 (デフォルトで t)、第四引数はユーザに値を質問するときに使うプロンプト、第五引数は値をユーザに質問するのとルールを利用して求めるのとのどちらを先に行うか、第六引数はユーザの回答を読み込むときに使う関数である。

よって、

```
(defparm name patient t "Patient's name: " t read-line)
```

は、「コンテキスト patient 用の name というパラメータを定義する。name の値の型はなんでもよく (t)、name の値を得る場合は、まずユーザに質問する (だめだったらルールから探す)。ユーザに name の値を質問するときは `"Patient's name: "` という文字列を出力し、回答を読み込む場合は read-line を使う」ことを意味する。

次は、MYCIN のルールである。

```
(defrule 52
  if (site culture is blood)
     (gram organism is neg)
     (morphology organism is rod)
     (burn patient is serious)
  then .4
     (identity organism is pseudomonas))

(defrule 71
  if (gram organism is pos)
     (morphology organism is coccus)
     (growth-conformation organism is clumps)
  then .7
     (identity organism is staphylococcus))

(defrule 73
  if (site culture is blood)
     (gram organism is neg)
     (morphology organism is rod)
     (aerobicity organism is anaerobic)
  then .9
```

```
     (identity organism is bacteroides))

(defrule 75
  if (gram organism is neg)
     (morphology organism is rod)
     (compromised-host patient is yes)
  then .6
     (identity organism is pseudomonas))

(defrule 107
  if (gram organism is neg)
     (morphology organism is rod)
     (aerobicity organism is aerobic)
  then .8
     (identity organism is enterobacteriaceae))

(defrule 165
  if (gram organism is pos)
     (morphology organism is coccus)
     (growth-conformation organism is chains)
  then .7
     (identity organism is streptococcus))

(defrule 52
  if (site culture is blood)
     (gram organism is neg)
     (morphology organism is rod)
     (burn patient is serious)
  then .4
     (identity organism is pseudomonas))
```

defrule の形式は、ルールの番号、if、ルールの前提部分、then、ルールの cf、最後に結論である。

前提部分、結論部分は (パラメータ名　コンテキスト 演算子 値) が複数並ぶ。ここで使用できる演算子は is (equal のエイリアス) だけとする。

defrule によって定義されたルールは、ルールの形式が正しいか、cf が閾値を下回っていないか、前提や結論のパラメータ値が適正であるかをチェックされ、各結論のパラメータ名をキーにしてルール専用の DB に登録される。

では、これらのルールを入力した状態で、MYCIN を利用してみる。MYCIN は mycin 関数で呼び出す。関数 emycin に与えられたコンテキストのリストと、パラメータの定義により、patient のパラメータである、name, sex, age をシステムがユーザに質問する。そして culture のパラメータである、 site, days-old を質問する。

```
CL-USER(8): (mycin)
------ PATIENT-1 ------
```

```
Patient's name:  Sylvia Fischer
Sex: female
Age: 27
------ CULTURE-1 ------
From what site was the specimen for CULTURE-1 taken? blood
How many days ago was this culture (CULTURE-1) obtained? 3
```

上記の例では、ユーザは回答の信頼度である cf を指定していない。cf を指定していない場合は、システムは cf が 1 であると解釈する。cf を指定する場合は、(値 1 cf1 値 2 cf2 ...) という形式で回答する。

続けて、organism のパラメータについて質問が始まる。

```
------ ORGANISM-1 ------
Enter the identity (genus) of ORGANISM-1: unknown
The gram stain of ORGANISM-1: ?
A GRAM must be of type (MEMBER ACID-FAST POS NEG)
The gram stain of ORGANISM-1: neg
```

organism の identity パラメータについて質問され、unknown と答えると、システムはルールを使って identity パラメータの値を求めようとする。また、? を入力すると、解の候補が出力される。

```
Is ORGANISM-1 a rod or coccus (etc.): rod
What is the AEROBICITY of ORGANISM-1? why
[Why is the value of AEROBICITY being asked for?]
It is known that:
    1) THE GRAM OF THE ORGANISM IS NEG
    1) THE MORPHOLOGY OF THE ORGANISM IS ROD
Therefore,
Rule 107:
  If
    1) THE AEROBICITY OF THE ORGANISM IS AEROBIC
  Then there is suggestive evidence (0.8) that
    1) THE IDENTITY OF THE ORGANISM IS ENTEROBACTERIACEAE
```

システムが細菌の親気性を質問してきたところで、なぜその質問をしてきたのか、why を入力することで逆にシステムに尋ねる。すると、システムは計算中のルールのうち、何がわかっているか、そしてもし細菌が親気性のものであれば、その細菌が ENTEROBACTERIACEAE と結論づけられることを出力する。

```
What is the AEROBICITY of ORGANISM-1? aerobic
Is Sylvia Fischer a compromised host? yes
Is Sylvia Fischer a burn patient?  If so, mild or serious? why
[Why is the value of BURN being asked for?]
It is known that:
```

```
    1) THE SITE OF THE CULTURE IS BLOOD
    1) THE GRAM OF THE ORGANISM IS NEG
    1) THE MORPHOLOGY OF THE ORGANISM IS ROD
Therefore,
Rule 52:
  If
    1) THE BURN OF THE PATIENT IS SERIOUS
  Then there is weakly suggestive evidence (0.4) that
    1) THE IDENTITY OF THE ORGANISM IS PSEUDOMONAS
Is Sylvia Fischer a burn patient?  If so, mild or serious? serious
Findings for ORGANISM-1:
 IDENTITY: ENTEROBACTERIACEAE (0.800)   PSEUDOMONAS (0.760)
```

ルール 107 より、identity パラメータの値として ENTEROBACTERIACEAE が導かれる。その cf は前提部分が全て真であり、ルール自体の cf が 0.8 であることから、0.8 である。ルール 52, 75 はともに PSEUDOMONAS を導く。それぞれの cf は 0.6, 0.4 であり、結合すると、

```
  0.6 + 0.4 - (0.6 * 0.4) = 0.76
```

となる。

この出力のあと、システムはさらに他の organism が計算中の culture に含まれるかを質問してくる。y を入力して他の organism があることをシステムに知らせると、2 つ目の organism の identity を求める計算を始める。

```
Is there another ORGANISM?y
------ ORGANISM-2 ------
Enter the identity (genus) of ORGANISM-2: unknown
The gram stain of ORGANISM-2: (neg .8 pos .2)
Is ORGANISM-2 a rod or coccus (etc.): rod
What is the AEROBICITY of ORGANISM-2? anaerobic
Findings for ORGANISM-2:
 IDENTITY: BACTEROIDES (0.720)   PSEUDOMONAS (0.646)
```

今度は、検査結果が曖昧なせいで gram パラメータの値が正確なものがわからなかったため、neg は 0.8、 pos は 0.2 の cf であるという条件付の回答をする。続けて、細菌の形状は rod、親気性は anaerobic と答えると、ルール 73 から BACTEROIDES、ルール 75, 52 から PSEUDOMONAS が導かれる。BACTEROIDES の cf は、前提部分で gram パラメータの値 neg の cf が 0.8 であり、ルール自体の cf が 0.9 なので、

```
 0.8 * 0.9 = 0.72
```

となる。PSEUDOMONAS の cf は、まず、ルール 75, 52 それぞれの cf を同様に計算すると、それぞれ

```
 0.6 * 0.8 = 0.48
 0.4 * 0.8 = 0.32
```

となり、結合すると、

```
0.48 + 0.32 - (0.48 * 0.32) = 0.6464
```

である。(出力のときは小数第 3 位まで出力している。)

このあと、システムは他の organism,culture, patient がないかを質問してくる。y を入力すると、そこから新く木が作られることになる。ここでは、全てに n を入力して終了する。

```
Is there another ORGANISM?n
Is there another CULTURE?n
Is there another PATIENT?n
NIL
```

実際に使用された MYCIN は約 400 個のルールを利用して推論を行う。

3 考察

MYCIN は実際の医者と同等、またはそれ以上の診断を下すことができるという説がある。EMYCIN は MYCIN 以外にも、次のエキスパートシステムの開発に利用された。

- PUFF (肺機能障害診断)
- SACON (工学的構造計算)
- GRAVIDA (産科の診察)
- CLOT (血液障害の診断)
- VM (呼吸管理)

ただし、EMYCIN を利用して開発されたエキスパートシステムの利用には注意が必要である。

例えば、EMYCIN では cf の正しさを保障できないという問題がある。非常にちいさな cf を持つルールであっても、同じ結論を持つルールと結合すると cf が大きくなり、ありえない結論が導かれることがある。例えば、あるタブロイド紙に『エルヴィスが Kalamazoo で生きている』という記事が載っており、この記事の信頼度 (cf) が 0.1 だとする。このルールひとつでは、この事象が true と判定されることはない。しかし、同じタブロイド紙をたくさん集め、それぞれ別のルールとして定義すると、これらのルールを 298 個結合した時点で cf が 0.95 となり、記事にかなりの信頼性があることになってしまう。(これを避けるためには、同じタブロイド紙の記事については cf を結合しないようにする必要がある)

また、EMYCIN が出した結論についても、正しい理解が必要となる。例えば、10,000 人に 1 人の割合で発症する病気 X があるとする。ある患者 A が病気 X の検査を受け、検査の結果がポジティブだったとする。実施した検査は 99% 正確だと医者が言えば、 A はほぼ間違いなく自分が病気 X にかかっていると思うだろう。しかし、これは医者の説明に間違いがある。10,001 人をランダムに選ぶと、このうち 1 人は病気 X であると考えられる。その 1 人は検査でポジティブとなる。この検査は、「99% 正確である」、つまり「1% 間違いを犯す」ことを意味し、それ以外の 10,000

人にも同じ検査をすると、100 人がポジティブになる。つまり、検査でポジティブと判定されるのは全部で 101 人であり、実際に病気 X に感染しているのは 1 人である。このことから、A が病気 X に感染している確率は約 1% である。訓練を受けた医者でさえ、このような間違いを犯すことがある。

ここで、EMYCIN を利用することの利点、欠点をまとめる。

利点

- MYCIN の開発では、一定の成果が得られている。
- cf は直感的なものであり、ルールの定義がしやすい。
- cf が閾値を上回る解、つまり、ある程度の信頼性がある解を複数個得られる。
- 解の信頼性が表示される。

欠点

- cf 自体の正しさが保障されていない。
- 一般的に複数のルールを利用して解を得るため、適切でない解が得られたとき、どのルールが原因であるかを特定するのが難しい。
- ルールとして定義していないことは推論できない。

4 参考文献

Paradims of Artificial Intelligence Programming: Case Studies in Common Lisp (Peter Norvig)
エキスパート・システム 基礎概念と実例 (J.L. アルティ M.J. クームス著/太原育夫訳)