Vol. 33 No. 6　　情　報　処　理　　June 1992

解　説

仕様獲得と知識獲得

4. ソフトウェア設計における仕様獲得支援†

上　原　邦　昭††

1. はじめに

従来より，数多くの自動プログラム合成システムが開発されている．一般に，これらの自動プログラム合成システムでは，超高級言語（自然言語を含む）による非操作的かつ抽象的な仕様記述から，「プログラミングに必要な知識」を用いて段階的詳細化（progressive refinement）[1]を行い，詳細で操作的かつアルゴリズム的な仕様記述に変換することを目標としているものが多い．また，従来のシステムのもう一つの特徴として，すべての「プログラミングに必要な知識」はあらかじめ用意されていると仮定していることがあげられる．しかしながら，ソフトウェア開発の全工程をこのような単純な割り切った見方で捉えると，「ユーザの要求は前もって明確な仕様書として定義されている」ことになり，また「プログラミングに必要な知識もすべて用意されている」ことになる[2]．もちろん，このような仮定は現場で行われるソフトウェア開発とは大きくかけ離れたものである[3]．このため，過去にも優れた先進的な自動プログラム合成システムが開発されているにも関わらず，実際に利用されるほどには普及していない．また，プログラム合成機能を一般化して汎用性あるシステムにまで高められたものもない．

本稿では，「ユーザの要求」も「プログラミングに必要な知識」もソフトウェア開発の初期段階では不完全なものであると仮定し，これらの獲得を，エキスパートシステム開発における知識獲得になぞらえて，要求（仕様）獲得と呼ぶ[4]☆．この定義に従って，要求（仕様）獲得をいくつかの部分問題に分類し，それぞれの問題の特徴を検討する．さらに，「知識工学を代表とする人工知能の研究で提案された数々の手法はソフトウェア開発においても有用である」という命題のもとに，いくつかの萌芽的な研究を題材として，今後あるべき要求（仕様）獲得の姿について検討する．なお，以降で示す事例は，いずれも研究対象として要求（仕様）獲得そのものを直接的に扱ったものではない．むしろ，知識工学や機械学習のテーマにおいて，プログラム合成を対象として行われた研究事例を，筆者があえて要求（仕様）獲得の部分問題に当てはめたものである．

2. 要求（仕様）獲得の基本的な考え方

2.1 要求（仕様）獲得の分類

大きく分類すると，プログラム合成には(1)問題領域における基本的・共通的な知識，(2)ソフトウェア開発者から獲得される設計知識，(3)ユーザから獲得される開発対象に関する知識（要求知識）などが必要である☆．このように考えると，要求（仕様）獲得は，システムの問題領域が与えられたとき，プログラム合成を十分に達成することができる程度に設計知識や要求知識を獲得する問題であると定義することができる．さらに，この定義から，要求（仕様）獲得を以下の3種類の部分問題に分類することができる．

1) 要求知識と設計知識がともに不完全である場合
2) 要求知識が不完全である場合
3) 設計知識が不完全である場合

問題1は要求されるプログラムの新規性が強く，要求者も詳細に要求仕様を記述することができ

† Supporting Requirements Acquisition in Software Engineering by Kuniaki UEHARA (Department of Computer and Systems Engineering, Faculty of Engineering, Kobe University).

†† 神戸大学工学部情報知能工学科

☆ 1993年1月にはIEEE International Symposium on Requirements Engineering が開催されることになっている．

☆ 文献 2) では，(2)に関する知識獲得を設計仕様獲得，(3)に関する知識獲得を要求仕様獲得と呼んでいる．なお，(1)に関してはプログラム合成に最低限必要な知識であるとみなし，本稿では議論しない．

ず，システム自身にも設計知識に欠落があり，要求者の意図するプログラムを合成することができない場合である．問題2は，設計知識は十分にあるが，要求仕様が不完全でどのような（what）プログラムを合成すればよいか分からない場合である．問題3は，逆に設計知識が不完全で，どのようにして（how）要求仕様を満たすプログラムを合成すればよいか分からない場合である．

2.2 部分問題に対する基本的なアプローチ

1. 問題1に対するアプローチ

すでに述べたように，問題1は開発すべきプログラムがユーザにとってもソフトウェア開発者にとっても新規性の高いものである．したがって，プログラミング過程の定式化が困難であり，通常の段階的詳細化のアプローチは適用できない．また，従来のプログラム合成システムでは，暗黙のうちに正確で完全なプログラムを開発することが仮定されており，このような問題をどのように取り扱えばよいか，ほとんど議論されることもなかった．しかしながら，人間が同様のことを行う場合を考えると，多くの人は不正確でもいいからとりあえずプログラムを作り，そのプログラムを修正して，正しいプログラムを作り上げるようにしている．そして，プログラム開発の経験を積むに連れて，ソフトウェア開発の初期段階でよく起こした失敗を徐々にしなくなる．このような問題解決を実現するために有効なアプローチとして，事例に基づく推論（Case-Based Reasoning）の利用が考えられる．CBRによるアプローチとは，過去のプログラム開発事例を参照して，現在の問題に流用可能な事例を組み合わせ，さらにその事例を修正することによって新たなプログラムを作り出す手法である．

2. 問題2に対するアプローチ

問題2では，設計知識が完全であると仮定されているために，ユーザからどのようにして要求知識を獲得すればよいかのみが問題となる．この場合，知識獲得の研究で提案された対話的知識移譲（interactive transfer of knowledge）によるアプローチを用いて，システムがユーザにインタビュしながら要求知識を獲得していく方法が有効である．特に問題2の特徴として，あらかじめ獲得すべき知識がどのような問題領域に属するかを知っていることが仮定されているために，その知識に従って効率的かつ体系的にインタビュを行う知的インタビュ技法が有効であると考えられる．

3. 問題3に対するアプローチ

問題3では，要求知識が完全であると仮定されている．言い換えると，ユーザはシステムに要求の具体的な例を与えることができると仮定されている．このような状況で，システムが与えられた例を満たすプログラムを作り出そうとした場合，システム内の設計知識には欠落があるので，必ずプログラム作成に行き詰まる箇所が存在する．ここで，仮になんらかの方法で行き詰まった箇所を同定できるものとすれば，次にどのようにして欠落した知識を補完するかということが問題となる．このような問題には，既存の設計知識と欠落した知識との間に類比を見つけだし，その類比から既存の知識を欠落した知識に適合するように修正する，類推による問題解決（analogical problem solving）[5]が利用できる．また，システムが要求の具体的な例に自分のもつ設計知識を適用し，欠落した部分を発見するためには，説明に基づく学習（Explanation-Based Learning）の考え方が利用できる．

以降では，各アプローチの基本的な考え方を示し，次にそれぞれのアプローチに基づいて開発された要求（仕様）獲得手法を紹介する．なお，当然のことながら，上記の部分問題と各アプローチは完全に1対1に対応しているわけではない．むしろ1対多となっており，一つの部分問題を解決するために複数のアプローチが援用されることが多い．たとえば，CBRでは過去の事例を検索するために類推が用いられるし，類推によって新たな概念が獲得される場合には，その概念を命名するためにユーザへのインタビューが必要になる．

3. CBRを用いたアプローチ

3.1 CBRの概要

人間は，獲得した知識をすべてルール化して整理しているわけではない．むしろ，ルール化される前の情報，すなわち過去に経験した成功および失敗の事例を直接構造化して蓄積しておき，新たな問題に遭遇した場合には，その問題と過去の事例を照合して，最も関連深い事例を用いて問題解決を行うことが多い．このような問題解決の仕方を，従来のルールに基づく推論（Rule-Based Rea-

soning) に対して事例に基づく推論 (Case-Based Reasoning) という. CBR は, 本来, 人間の記憶構造をいかにしてモデル化するかという Schank 学派の認知心理学から生まれたものである[6].

一方, 知識工学の分野では,

1) 専門家からルール形式として知識を獲得することが困難で, 断片的な知識しか獲得できない場合があること

2) 獲得されたルールがあまりにも一般的で教科書的な知識である場合, エキスパートシステムが初心者的な振舞いをし, 専門家の振舞いをシミュレートできないこと

3) ルールに基づく推論では, 過去に類似の問題解決を行っていても, 再び同様の推論を行わなければならず, 計算コストがかかること

などの問題が生じている. このような「ルールに基づく推論の限界」が原因となり, CBR は知識工学の観点からも急速に研究が進められている.

CBR の枠組みでは, 過去の事例は事例ベースに蓄積される. また, 事例には, 将来の問題解決に再利用できるように, 索引付け (indexing) が行われる. この索引付けでは, 新たな事例を他の事例と区別し, 後の検索をできるだけ効率よく系統的に行うことができるように, ある種の索引キー (index key) が割り付けられる*.

しかしながら, 検索された事例が, そのまま現在の問題に利用できることはほとんどない. 逆に言うと, もし現在の問題と同一な状況での事例がすでに事例ベースに含まれているならば, 新たに知識獲得が行われることはない. このため, 検索された事例には, 現在の問題に適用できるように, 修正・適合が施される. さらに, その事例に基づいて与えられた問題の解決が試みられ, もし妥当な評価が得られれば, 新たな事例として事例ベースの中に適切な形で蓄積される.

3.2 TA

Williams の提案した TA (Teaching Assistant)[8] は, 過去に遭遇したプログラムを事例ベースに蓄えておき, 新しいプログラムを作成する際には, 事例ベースから過去のプログラム事例を検索し, 必要ならばそのプログラムを修正しながらプログラミングを学習するシステムである. TA は以下の3種類の事例ベースをもっている.

1) システムがすでに「理解している」プログラムを体系化して蓄積しているプログラム事例ベース.

2) バグを含んだコードを正常に動作するコードに変換するためのパッチ事例ベース. なお, パッチ事例は正しいコードと誤ったコードの対からなる.

3) プログラムを作成する際に遭遇した各種のバグを体系化したバグ事例ベース. ただし, それぞれのバグにはプログラム中のどの部分が誤っているかという情報が含まれている.

ここで,「プログラムを理解する」とはどういうことかを明確にしておかなければならない. TA では,「プログラムを理解する」ためには以下の3種類の能力が必要であるとしている.

1) エミュレーション能力 (emulation skill)

プログラムの仕様が与えられると, 作成すべきプログラムがどのような出力を生成するか予測可能なこと

2) 説明能力 (explanatory skill)

予測された出力がどのように生成されたかを1ステップごとに説明可能なこと

3) 類推能力 (analogical skill)

プログラムの仕様が与えられると, 仕様と類似しているプログラムの仕様を発見可能なこと

TA の動作概要を以下に示す.

1) 類推能力を用いて, 与えられた仕様と類似する仕様をもつプログラムをプログラム事例ベースから検索する.

2) エミュレーション能力を用いて, 与えられた仕様から予測されるプログラムの振舞いと類似したプログラムの実際の振舞いを比較する.

3) もし予測された振舞いと実際の振舞いが異なっていれば, 説明能力を用いてプログラム中のどのコードが誤っているかを指摘し, さらに類推能力を用いて正確なコードを発見する.

具体的な例を用いて上記の動作を説明する. まず, TA に「リスト中のすべての要素がアトムでないことを示すプログラム NO-ATOMS を作成せよ」という仕様

```
(∀ ?X ?L (¬ (ATOMIC ?X)))
```

が与えられたものとする. また, プログラム事例

* たとえば, Protos[7] では同一範疇での事例の典型的な度合いを示す典型度 (prototypicality), 関連する範疇や事例を直接指し示す特徴 (reminding), 類似した事例を明確に区別しうる特徴 (exemplar difference) による索引付けが行われる.

ベースには「リスト中のすべての要素がアトムからなることを示す」プログラム ISLAT のみが蓄積されているものとする（図-1（a））．TA が NO-ATOMS に類似した事例として ISLAT を検索すると，このプログラムの引数 ?l にリスト（(a)（b））を代入した見本が与えられる．TA は，エミュレーション能力を用いて，仕様から予測される出力が実際の出力と異なることを発見する．さらに，プログラムを1ステップごとに追跡しながらユーザとの対話を開始する．この例では，2行目の (atom (car ?l)) が誤っているが，ユーザから正しいコード (not (atom (car ?l))) が与えられると，正誤コード対をパッチ事例ベースに蓄える．また，正しいコードを適用した正しいプログラムをプログラム事例ベースに蓄える（図-1（b））．さらに，「cond 節の第2番目のコードが誤っている」というバグ情報をバグ事例ベースに蓄える．

同様にして，「リスト中に含まれるすべてのサブリストの第1要素がアトムであることを示す」プログラム ALL-ATOM-CAR の仕様

(∀ ?X ?L (ATOMIC (FIRST ?Y ?X)))

が与えられると，ユーザとの対話によって，旧コードが (atom (car ?l))，新コードが (atom (car (car ?l))) である新たなパッチ事例を獲得する．さらに，このパッチ事例をプログラム事例 ISLAT に適用し，新たなプログラム事例を獲得する（図-1（c））．

最終的に，「リスト中のすべてのサブリストがアトムでないことを示す」プログラム NO-ATOM-CAR の仕様

(∀ ?X ?L (¬ (ATOMIC (FIRST ?Y ?X))))

が与えられるとする．この場合，TA は仕様に類似したプログラム事例として NO-ATOMS を発見する．さらに，パッチ事例ベースを検索して，ALL-ATOM-CAR を作成するために獲得したパッチを発見する．このパッチを NO-ATOMS に適用すると，仕様を満たす新たなプログラムが作成される（図-1(d)）．

```
(cond ((null ?l) t)
      ((atom (car ?l)) (islat (cdr ?l)))
      (t nil))
```
（a） ISLAT のプログラム事例

```
(cond ((null ?l) t)
      ((not (atom (car ?l))) (islat (cdr ?l)))
      (t nil))
```
（b） NO-ATOMS のプログラム事例

```
(cond ((null ?l) t)
      ((atom (car (car ?l))) (islat (cdr ?l)))
      (t nil))
```
（c） ALL-ATOM-CAR のプログラム事例

```
(cond ((null ?l) t)
      ((not (atom (car (car ?l)))) (no-atom-car (cdr ?l)))
      (t nil))
```
（d） NO-ATOM-CAR のプログラム事例

図-1 TA のプログラム合成過程

4. 知的インタビュを用いたアプローチ

4.1 インタビュの概要

プログラミングに必要な設計知識を知識ベースに貯え，それらの知識を推論エンジンによって適用しながらプログラムを合成しようというアプローチは，見方を変えると設計型のエキスパートシステムの一種であると考えることもできる．エキスパートシステムの開発においては，知識，すなわち問題解決に必要な情報を専門家やテキストなどの知識源から収集・定式化し，それを推論エンジンが取り扱える形式に変換してシステム内に取り込む作業が必要である．また，開発されたエキスパートシステムが稼動可能な状況になってからも，知識の誤りや矛盾，あるいは知識の増大に対応して，常に修正・変更しなければならない．このような知識の収集・変換・管理という作業を知識獲得と呼ぶ．知識獲得は，知識工学者が対象分野の専門家にインタビュしながら進められるが，一般に，人間は自分の知っている知識を誤りなく，明文化して伝えることが困難である．このため，効率的かつ体系的にインタビュを行う知識獲得支援ツールの開発が盛んに行われている．このようなシステムとして I²S-LD[9] がある．I²S-LD は，インタビュ形式でユーザと対話しながらデータベースの論理設計を支援するシステムである．

4.2 I²S-LD

I²S-LD は，まず，ユーザが与えた検索要求文の集合を解析する．検索要求文から抽出される情報は，ドメインモデルと呼ばれる内部モデルとして蓄積される．次に，ユーザにインタビュを実施し，必要と思われる情報を引き出し，ドメインモデルを洗練化する．最後に，論理設計によってドメインモデルをデータベースの概念スキーマに変換して出力する．I²S-LD を要求（仕様）獲得

にあてはめると，検索要求文はプログラム仕様，ドメインモデルは内部仕様に相当したものと考えることができる．

ドメインモデルはプラン構造と呼ぶ形式で表現される．プラン構造は，問題領域における個々の活動や実態（対象物）を，それぞれ動詞フレーム，名詞フレームによって表現したものである．動詞フレームは，goal, action および precondition からなるスロットをもっている．各スロットは，それぞれ，その動詞が表す行為を実行したときに達成する可能性がある状態，動詞が表す行為，動詞が表す行為を実行するために前もって達成していなければならない状態を示している．

名詞フレームには，その名詞の属性および上位概念が記述される．なお，I²S-LD の特徴として，具体的な名詞辞書はいっさいもたず，問題領域に独立な 17 個の抽象概念に関する知識（抽象概念フレーム）のみを用意していることがあげられる．これは，I²S-LD が多様な問題領域でも論理設計を支援できるように，問題領域に依存した特定の知識（名詞辞書）はもたずに汎用性を高めることを意図したものである．このため，検索要求文に未知の名詞が現れる場合は，インタビュを通じてその名詞がどのような抽象概念の下位概念であるかをユーザに質問しながら，対応する名詞フレームを作成するようにしている．

このようなインタビュを実現するために，動詞辞書フレームには候補概念スロットが用意され，動詞フレームの各スロットに入る概念の候補が示されている．たとえば，動詞フレーム construct の goal スロットには候補概念として physicalObject が記述されている（図-2(a)）．これは，「construct という行為を実行すれば physicalObject が生成される可能性がある」という goal を示している．未知の名詞が出現した場合は，この候補概念を参照してユーザに質問が行われる．

名詞の属する抽象概念フレームが決定されると，その名詞に対応する名詞フレームが生成される．このとき，抽象概念フレームの候補属性スロットには，下位概念となる名詞がもつべき属性の候補が記述されている．たとえば，抽象概念 physicalObject の候補属性スロットには size, weight, cost, price という属性が記述されている（図-2(b)）．これらの候補属性を参照して，名詞フレームに付随する各種の属性がユーザに質問される．

```
construct
  super_class_is : [verbs]
  value_class :
   goal :
    exist :
     what : [physicalObject]
   precondition :
    exist :
     what : [physicalObject]
```

(a) 動詞辞書フレーム construct

```
physicalObject
  super_class_is : [nouns]
  attr_cands : [size, weight, const, price]
```

(b) 抽象概念フレーム physicalObject

図-2 I²S-LD の知識表現

検索要求文を解析した結果，プラン構造による「粗い」ドメインモデルが構築されるが，このドメインモデルはまだ情報が欠落していたり，曖昧な情報が存在しており，データベースの概念フレームに変換することはできない．このようなドメインモデルを洗練するために，さらにユーザへのインタビュが行われる．たとえば，動詞フレーム間の接続関係を調べて，ユーザがまだ言及していない行為に関する検索要求文を示唆したり，すでに検索要求文によって言及されている行為との接続を図るインタビュがある．このほかにも，プランの goal と precondition がもつ情報の詳細度を等しくするためのインタビュ，時間に関する質問を行うためのインタビュ，属性名詞を手掛かりとして新しい活動（動詞）を示唆するためのインタビュなどが用意されている．

5. EBL と類推の統合によるアプローチ

5.1 EBL の概要

近年，機械学習の新しいパラダイムとして説明に基づく学習（EBL）が注目されている．EBL では，学習すべき概念（目標概念）と領域理論が事前に定義され，ある特定の例（訓練例）が与えられると，訓練例が目標概念の一例であることを領域理論を用いて証明（説明）する．次に，証明された結果（説明）を正確さを維持しながら最大限に一般化する．最終的に，一般化された説明から目標概念に対する定義を抽出するものである[10),11)]．

EBLは，ただ一つの訓練例から目標概念を学習することができるという利点があるが，これは領域理論が完全な形で与えられている場合に限られる．言い換えると，領域理論が不足している場合には，訓練例を説明することができない．たとえば，領域理論として

F0⇒F1　F2　F3

F2⇒y

F3⇒z

目標概念としてF0，訓練例としてx，y，zが与えられる場合，F1とxを結ぶためのルールが欠落しているために，訓練例を説明することができない．なお，**図-3**に示したような，完全ではない説明を部分的な説明（partial explanation）と呼ぶ．

このような欠落したルールを推測し，さらになんらかの形で推測したルールが妥当であることを確かめるために，EBLをさらに発展した学習アルゴリズムがいくつか提案されている*．たとえば，Hammondの CBD[12]，Rajamoneyの ADEPT[13]，Genestの LISE[14]，Pazzaniの OCCAM[15]などがある．文献10）の分類に従うと，これらの学習アルゴリズムは分析手法（analytical method）と実験手法（empirical method）の2種類に分類できる．分析手法は，領域理論とは別の「補助的な理論」を用いて，推測されるルールが妥当であるかを評価する手法である[12)~14)]．逆に，実験手法は他の理論に頼らず，複数の訓練例を用いて推測されるルールを実験的に評価する手法である[15]．

以上の考え方をプログラム合成に当てはめると，EBLの訓練例はユーザの要求例に，領域理論は設計知識に，説明は詳細化されたプログラムの内部仕様に相当する．また，不完全な領域理論は設計知識の欠落に，部分的な説明は詳細化された内部仕様が不完全なことに相当する．したがって，最後に問題となるのは，欠落した設計知識をどのようにして推測すればよいかということである．上記の分析手法と実験手法のうち，分析手法

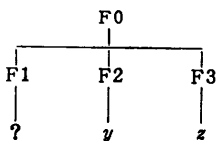


図-3　部分的な説明

* 従来のEBLが記号レベルの学習と呼ばれるのに対し，知識レベルの学習と呼ばれる．

（たとえば，類推による問題解決と融合した学習アルゴリズム）は，単一の訓練例を用いて部分的な説明を妥当な説明（正当性は保証されない）に変換する（妥当な内部仕様を生成する）ことができるために，プログラム合成における欠落した設計知識の獲得には有効な手法である．

5.2 LISE

Genestらの開発したLISE(Learning In Software Engineering)は，ユーザの与える非形式的かつ非操作的な銀行業務システムに関する要求（仕様）を，形式的かつ操作的な内部仕様に変換することを目的としている．LISEの領域知識は，通常のEBLで用いられるルール形式とは異なり，階層的に構成されたフレーム形式によって表現される．各フレームはオブジェクト（object）あるいはオブジェクトに適用可能な操作（operation）からなる．このうち，オブジェクトは問題領域である銀行業務システムの対象や構造を表現するために用いられる．また，操作は銀行業務システムの振舞いを表現するために用いられる．**図-4**にLISEの領域知識の一部を示す．

操作を表現しているフレームは ACTION 型と

```
client
isa : person
  account
  goal (account, identify_client)
  credit_margin
  goal (credit_margin, protect_bank_interest)
  safety_box
  necessary_condition :
    (client (Person) <- account (Person, Account))
    or
    (client (Person) <- safety_box (Person, Safety_box))
withdraw (Person, Amount)
isa : TRANSACTION
  precondition :
    account (Person, Account)
    goal (account, identify_client)
  procedure :
    debit (Account, Amount)
    issue_money (Person, Amount)
    goal (issue_money, inc_client_liquid)
debit (Account, Amount)
isa : ACTION
  precondition :
    balance (Account, Balance)
    goal (balance, protect_bank_interest)
    Balance>Amount
  procedure :
    sub_fm_balance (Account, Amount)
    goal (sub_fm_balance, record_transaction)
```

図-4　銀行業務システムについての領域知識

TRANSACTION 型に分類される．いずれのフレームも前提条件 (precondition) と手続き (procedure) からなる．前提条件は操作を適用する際の条件を記述したもの，手続きは一連の操作系列である．ACTION 型の操作は，前提条件とただ1個のプリミティブな操作からなる手続きによって定義される．なお，プリミティブな操作は，これ以上分解できない基本的な操作単位である．同様に，TRANSACTION 型の操作も前提条件と手続きからなるが，手続きの操作系列にはプリミティブとプリミティブでない操作が混在している．また，フレームの前提条件や手続きのなかに含まれるゴール (goal) は，対応する操作が達成すべき制約条件を示しており，類推的推論 (analogical reasoning) を実現する際に用いられる*．以上のような枠組みで定義される領域知識は，EBL の実行時に前もって通常のルール形式に変換される．

すでに述べたように，LISE は銀行業務システムの内部仕様に対応した領域知識を用いて，ユーザの要求（仕様）に対応した訓練例を説明しようとする．しかしながら，領域知識が欠落している場合には，説明に必要なルールが抜けており，一つないしは複数の部分的な説明を生成する可能性がある．このため，以下の3ステップを用意して不完全な領域知識を取り扱っている．

ステップ1

まず，生成された複数の部分的な説明のうち，未証明の部分が最も少ない説明を選択する．次に，発想的推論 (abductive reasoning)[16] を用いて部分的な説明を完全にし，妥当な説明を生成しようと試みる．発想的推論とは，規則(ルール)Q←Pと観測事象（ファクト）Q が与えられたときに，仮説としてPを生成する操作である．たとえば，図-4の領域知識を用いて withdraw (bob, 100) を説明づけようとした場合，もし訓練例にファクト account (bob, Account) が欠落しているならば，完全な説明を生成することはできない．しかしながら，訓練例にファクト client (bob) が与えられているとすれば，領域知識にあるルール

client (Person)←account (Person, Account) から account (bob, Account) を仮説として生成し，妥当な説明を作り上げることができる（図-5）．

ステップ2

ステップ1が失敗した場合，部分的な説明における未証明のルールの条件部と訓練例の事実の関連を調べるために，類推的推論を適用して，部分的な説明から妥当な説明を作り出そうと試みる．もしステップ2で妥当な説明が生成されると，その説明から新しいルールが抽出され，領域知識に追加される．

ステップ3

ステップ2も失敗した場合，さらにCBR を用いて事例ベースから過去の事例を検索し，その事例を訓練例に適用して妥当な説明を生成しようと試みる．ステップ3は，領域知識に含まれる一般的なルールでは対処することができない，例外的な訓練例を取り扱うために用意されたものである．

以下では，LISE の中心的なアプローチであるステップ2についてさらに詳しく説明する．なお，訓練例として TRANSACTION 型の borrow に関する例が与えられるものとする．

```
borrow (bob, 1000).
        account (bob, acc_1),
        credit_margin (bob, 3500),
        record_loan (bob, 1000),
        issue_money (bob, 1000),
        car (bob, car_of_bob),
        value (car_of_bob, 12000).
```

LISE の領域知識に borrow に関する知識が含まれていないものとすると，訓練例からはいくつかの部分的な説明が生成される可能性がある．図-6（a）に TRANSACTION 型の withdraw と deposit を用いて生成される部分的な説明を示す．このうち，いずれかの説明を選択して妥当な説明を生成しなければならないが，LISE には部分的

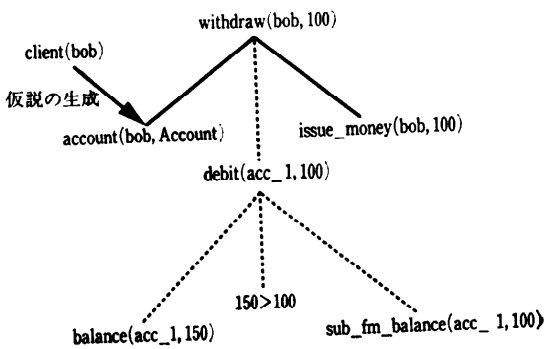
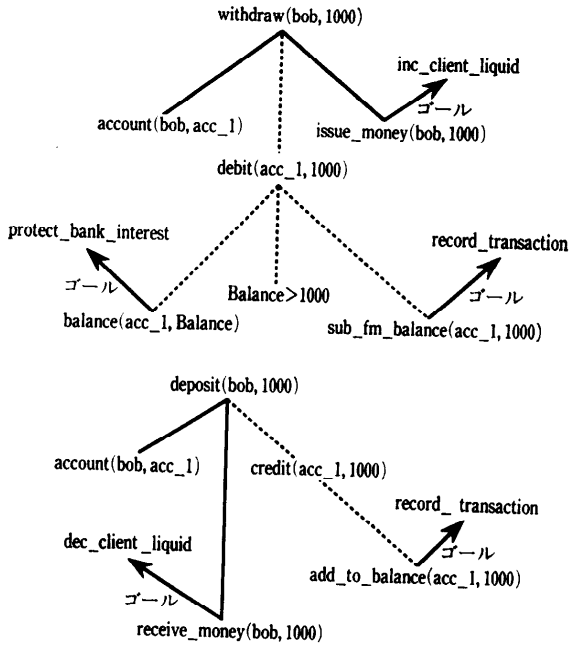


図-5 発想的推論による妥当な説明の生成

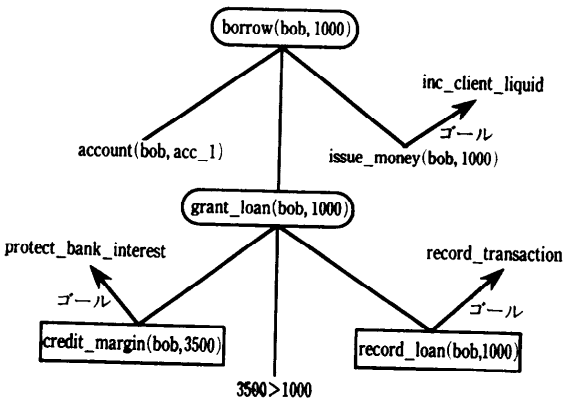
* オブジェクトを表現しているフレームは ENTITY 型と呼ばれるが，以降の議論には登場しないのでここでは説明しない．

な説明を選択するために，いくつかの経験則が用意されている．この例では，「証明された条件部をより多く含む説明のほうに高い得点を与える」という経験則を用いて，withdraw による部分的な説明が選択され，borrow の妥当な説明を生成する際に利用される．ただし，図-6(a)では証明された条件部を実線で，説明されていない部分を破線で表している．

次に，withdraw による部分的な説明のうち，未証明の条件部を訓練例と置き換える．たとえば，balance (Account, Balance) と credit_margin (bob, 3500) はともに同一のゴール protect_bank_interest をもつために，balance は credit_margin に置き換えられる．同様に sub_fm_balance (Balance, Amount) は record_loan (bob, 1000) に置き換えられる．以上の操作が類推的推論に相当している．なお，debit (Account, Amount) に対応する記述は訓練例に含まれていないが，ユーザに新たにつける名前を問い合わすことによって，grant_loan と名付けられる（新規述語の命名）．以上のようにして borrow の妥当な説明が生成されると（**図-6**(b)），最終的に，borrow は withdraw のフレーム，grant_loan は debit のフレームを雛型として，欠落していた知識が抽出される．



(a) withdraw と deposit から生成される部分的な説明



(b) withdraw を用いた borrow の妥当な説明

図-6 LISE による類推的推論の例

6. 要求（仕様）獲得の問題点

本章では，上記のシステムを叩き台として，要求（仕様）獲得の問題点について検討する．

6.1 事例の類似性について

Carbonell は文献 5) で類推による問題解決における検討事項として，以下の4点をあげている．

1) 与えられた問題状況（仕様）と類似した事例（プログラム事例）の間で重要な共通点は何か

2) 過去の事例のなかからどのような知識を変換するか

3) どのようにしてその変換プロセスを実現するか

4) どのようにして膨大な知識(事例ベース)から類似した事例を選択するか

これらの検討事項は，CBR や類推を用いた要求（仕様）獲得においても同様にあてはまる．すなわち，検討事項のうち 1)，4) は事例の索引付け問題と関係している．たとえば，TA では与えられたプログラム仕様から，エミュレーションをするために必要な内部形式仕様が得られる．この表現は，TA がプログラム仕様をどのように「理解しているか」を表したものである．この内部形式仕様から検索のキーとなる索引が生成される．一方，LISE では操作にゴールという制約条件を付加し，ゴールが等しい操作とファクトを置き換えるという手法で類推的推論を実現している．言い換えると，ゴールが索引の役割を果たしていることになる．また，TA では与えられるプログラム仕様がリストの

サーチに関するもの，LISEでは銀行の業務に関するものであると仮定されており，索引となる特徴も妥当な数に押さえられている．しかしながら，一般のプログラム開発においては，適切な特徴を用いて必要かつ十分な索引付けを行わなければ，類似した事例を発見することは困難になる．さらに，事例の索引付けで用いる特徴を汎用性があるように一般化しておかなければ，索引が有効に機能しなくなるという問題がある．

6.2 事例ベースの組織化について

類似した過去のプログラム事例をどのようにして検索するかということも大きな問題である．たとえば，LISEでは事例の検索に要するコストが大きいので，システムの推論過程を3ステップに分けてCBRを最後に適用している．さらに，銀行業務のうちで一般的な領域知識にあてはまらない，例外的な訓練例のみにCBRを適用している．一方，TAでは事例ベースの構築については言及されていない．このように，LISEやTAでは事例ベースの構築についてほとんど考慮されていないが，できるだけ効率良く系統的に検索を行うためには，複数の事例に共通する特徴を一般化し，事例集合を階層化して事例ベースを構築するなど，事例ベースの組織化についても検討しなければならない．また，Carbonellは類似した事例を正の例集合，関係ない事例を負の例集合として形成し，これらの集合からなんらかの帰納エンジンによって，事例に頻出する共通点を一般化プランとして抽象化した上で抽出することを提案している[5]．このように，事例から汎用的な知識を獲得するメカニズムを要求（仕様）獲得に導入することについても検討しなければならない問題だと思われる．

6.3 検索された事例の修正・適合について

検討事項のうち2)，3)は事例の修正問題と関連したものである．Carbonellは事例を修正する際の手法として解の変形類推（transformational analogy）[17]と誘導類推（derivational analogy）[5]を提案している．このうち変形類推は，過去の事例から密接に関係する解を検索し，新しい問題状況の必要条件を満たすように変換オペレータを適用し，解を徐々に変形するものである．また，誘導類推は，事例における解の誘導過程を変換して問題の解を再誘導するものである．言い換えると，変形類推では，問題解決によって得られる解そのものに有用な類似性があると考えられていたが，誘導類推では，解を導くために使われる推論プロセス，あるいは推論プロセスで行われる意志決定など，表面的な事例の特徴には現れない隠れた判断過程の中に有用な類似性があるという考え方に基づいている．Carbonellの考え方に照らし合わせてみると，TAでは事例の修正に単純な変形類推を用いているのみであり，LISEでもゴールのマッチングによるファクトと操作の置き換えが行われているに過ぎない．今後，広範なプログラム合成を考慮する場合には，誘導類推などの「深い類推」を事例の修正に導入することについても検討しなければならない[18]．

6.4 修正された事例の検証について

以上の問題以外にも，CBRや類推によって獲得された要求（仕様）を検証・修正するという問題が残されている．たとえば，TAではプログラム事例をユーザとの対話によって修正するというアプローチを採用している．またLISEでは，類推的推論によって得られた知識を疑うことなく領域知識に追加するというアプローチを採っている．しかしながら，システムが誤った類推を行った場合には，得られる結果もユーザの要求とは異なったものになる可能性がある．このような場合には，Falkenhainerの行った定性モデルのシミュレーションと基本原理（first principle）による理論修正（theory revision）[19]，あるいはRajamoneyの行った実験に基づく理論修正[13]などの考え方を利用することについても検討する必要がある．

6.5 領域知識の汎用性と特殊性について

インタビュによるアプローチでは，汎用性と特殊性という問題が残されている．たとえば，I²S-LDでは多様な問題領域でも論理設計を支援できるように，領域に依存した特定の知識（名詞辞書）はもたずに，汎用性を高めることに重点を置いている．しかしながら，このことが検索要求文に含まれる名詞に関する質問を非常に多く生成する原因となっている．また，人間にとっては自明な事柄もユーザに質問してしまう原因となっている．このような問題を解決するために，文献9)ではI²S-LDに学習機能を導入し，インタビュによって獲得された知識を蓄積しておき，以降のインタビュではそれを利用することによって，より

効率的，効果的に領域知識を獲得する手法が提案されている．しかしながら，このような学習機能の導入によってドメインモデルが拡大され，複数の領域におけるドメインモデルが混在するようになると，それらのうちからどのようにして類似なドメインモデルを選択するか，新たな知識をどのようにして既存のドメインモデルに統合するか(Knowledge integration)，混在したドメインモデルをどのようにして管理するかなど，さらに複雑な問題の解決が必要になる．

7. おわりに

本稿では，「知識工学を代表とする人工知能の研究で提案された数々の手法はソフトウェア開発においても有用である」という命題のもとに，要求（仕様）獲得をいくつかの部分問題に分類し，それぞれの問題に対する代表的なアプローチを示した．初めにも述べたように，本稿で示した研究事例はいずれも要求（仕様）獲得そのものを扱ったものではない．したがって，各システムで用いられた例題も単純であり，現場のそれとは大きくかけ離れている．しかしながら，これらの萌芽的な研究を出発点として，将来，上記の命題が真であるという結論が得られることが期待される．

参考文献

1) Partsch, H. and Steinbrüggen : Program Transformation Systems, ACM Computing Surveys, Vol. 15, No. 3, pp. 199-236 (1983).
2) 折原，荒木，西村：仕様獲得 vs. 知識獲得，情報処理 Vol. 33, No. 6, pp. 605-611 (1992).
3) 鈴木，腰原，木津谷：SE の仕様獲得の実際，情報処理 Vol. 33, No. 6, pp. 612-616 (1992).
4) 上原邦昭：要求獲得の知的支援，人工知能学会誌，Vol. 6, No. 2, pp. 170-172 (1991).
5) Carbonell, J.G.: Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition, in Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (eds.) Machine Learning, An Artificial Intelligence Approach, Vol. II, pp. 371-392, Morgan Kaufman (1986).
6) Schank, R.C.: Dynamic Memory, Cambridge University Press (1982).
7) Bareiss, R., Porter, B.W. and Murray, K.S.: Supporting Start-to-finish Development of Knowledge Bases, Machine Learning, Vol. 4, No. 3/4, pp. 259-284 (1989).
8) Williams, R.S.: Learning to Program by Examining and Modifying Cases, Proc. of the 5th International Conference on Machine Learning, pp. 318-324 (1988).
9) 川口敦生：知的インタビューシステムに関する研究，大阪大学大学院工学研究科博士論文 (1989).
10) Ellman, T.: Explanation-Based Learning: A Survey of Programs and Perspectives, ACM Computing Surveys, Vol. 21, No. 2, pp. 163-221 (1989).
11) Danyluk, A.P.: A Survey of Machine Learning Systems Integrating Explanation-Based and Similarity-Based Methods, Technical Report, Columbia University (1989).
12) Hammond, K.J. and Hurwitz, N.: Extracting Diagnostic Features from Explanations, Proc. of Case-Based Reasoning Workshop, pp. 169-178 (1988).
13) Rajamoney, S.A.: Experimentation-Based Theory Revision, Proc. of AAAI Spring Symposium on Explanation-Based Learning, pp. 7-11 (1988).
14) Genest, J., Matwin, S. and Plante, B.: Explanation-Based Learning with Incomplete Theories: A Three-Step Approach, Proc. of the 7th International Conference on Machine Learning, pp. 286-294 (1990).
15) Pazzani, M.J.: Integrated Learning with Incorrect and Incomplete Theories, Proc. of the 5th International Conference on Machine Learning, pp. 291-297 (1988).
16) Pople, H.E.: On the Mechanization of Abductive Logic, Proc. of the 3rd IJCAI, pp. 147-152 (1973).
17) Carbonell, J.G.: Learning by Analogy: Formulating and Generalizing Plans from Past Experience, in Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (eds.) Machine Learning, An Artificial Intelligence Approach, Vol. I, pp. 371-392, Morgan Kaufmann (1983).
18) Carbonell, J.G.: Integrating Derivational Analogy into a General Problem Solving Achitecture, Proc. of the Case-Based Reasoning Workshop, pp. 104-124 (1988).
19) Falkenhainer, B.C.: Learning from Phisical Analogies: A Study in Analogy and the Explanation Process, Ph. D thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign (1989).

(平成3年10月9日受付)

上原 邦昭（正会員）

1978 年大阪大学基礎工学部情報工学科卒業．1983 年同大学院基礎工学研究科博士後期課程単位取得退学．大阪大学産業科学研究所助手，講師を経て，1990 年より神戸大学工学部情報知能工学科助教授．工学博士．人工知能，特に機械学習，自然言語理解，プログラム合成の研究に従事．1990 年度人工知能学会研究奨励賞受賞．人工知能学会，電子情報通信学会，計量国語学会，日本ソフトウェア科学会，システム制御情報学会各会員．