

基本原理からの説明機能を持つプログラム合成システム

松元貴志 上原邦昭 豊田順一

大阪大学産業科学研究所

プログラム合成システムを一般的なユーザにとって使いやすいものとすることを目的とした、自然言語仕様からの仕様獲得支援機能およびプログラム合成過程の説明機能の実現手法を提案する。本手法では、はじめに自然言語で記述された仕様を解析し、次に解析結果に対して抽象化および対象領域の常識的知識による補完を行い形式的な仕様を作成する。最後に形式的仕様を目標とする計算機言語へと詳細化しプログラムを合成する。ユーザが自然言語で記述した初期的な仕様から、最終的にプログラムを合成するまでの全過程をシステムが管理しているために、仕様記述時のユーザの漠然とした認識に溯って合成されたプログラムを説明することができる。説明時には、仕様の解析および合成の際にシステムが参照した問題領域のモデルをもとに説明を生成するようにしている。

A Program Synthesis System with Explanatory Facilities Through Models of Application Domains

Takashi MATSUMOTO, Kuniaki UEHARA, and Jun'ichi TOYODA

The Institute of Scientific and Industrial Research, Osaka University
8-1 Mihogaoka, Ibaraki, Osaka 567 Japan

A paradigm for generating explanation on program synthesis systems is proposed. In this paradigm, the course of program construction contains two stages, namely, interpretation and synthesis stages. In interpretation stage, the system parses natural language specifications first. Then formal specifications are derived from the results of parsing, by abstracting them away from the physical world, and complementing them with domain knowledge. In synthesis stage, formal specifications are refined into programs written in a programming language. After proposing synthesized programs to users, the system can explain what it did. Explanations are generated from the domain model used in the course of program construction.

1. はじめに

従来のプログラム合成システムでは、設計時にユーザとシステムのコミュニケーションの重要性が省みられず、ユーザにとって使いやすいシステムとはなっていなかった。プログラム合成システムをユーザの立場にたって眺めた場合に、不満を感じる点は、①ユーザの要求を記述する手段が制限されており、仕様記述が困難であること、②システムが合成結果のプログラムのみを提示し、どのようにして合成を行なったのかが示されないことである。

①はユーザからシステムへの要求伝達方法に関するものである。プログラム合成システムを利用するにあたって、ユーザはプログラムに対する要求をシステムに伝達する必要がある。この要求は、はじめにユーザの頭の中で曖昧なイメージとして形成され、つぎに自然言語によって記述できる程度整理された状態を経て、最終的にプログラム合成システムが受理できる形式的な仕様となる。この過程を仕様獲得と呼ぶ。従来のプログラム合成システムでは、仕様獲得の部分はユーザに任せ、要求が過不足なく明確に表現された形式的仕様を入力とするものが多かった。しかしながら、一般に仕様獲得はプログラム作成の各工程の中で最も困難なものであり[1]、これを支援しない限りプログラミングにおけるユーザの労力を大幅に軽減することは難しい。したがって、ユーザにとって比較的容易に記述できる自然言語による仕様を受け取り、仕様獲得を支援する能力を持つプログラム合成システムが望まれる。

また、②はシステムからユーザへの合成結果の伝達方法に関するものである。一般にわれわれは他人の提案や助言を受け入れるために、なぜそのような提案や助言がなされたのかという理由を知ろうとする[6]。プログラム合成システムを利用する場合も同様に、システムが提案した結論が、どのような根拠のもとに、どのような過程を経て導かれたのかをユーザは知りたがる。合成の根拠が示されて初めて、ユーザはシステムの結論を信頼して受け入れることができる。したがって、プログラム合成システムをユーザにとって使いやすいものにするためには、合成結果としてプログラムを提示するだけでなく、合成の過程や、合成の根拠を説明する必要があると考える[5]。

われわれは、以上の2点をプログラム合成システムにおいて要求されるインタフェース機能としてとらえ、これらの機能を統合したプログラム合成システムJUSTIFYを開発中である。JUSTIFYでは、はじめにユーザが自然言語で記述した要求仕様を受け取り、これを仕様記述言語にもプログラミング言語にも依存しない形で書かれた形式仕様へと変換する。次にこの形式仕様を目的プログラムへと変換する。合成過程の説明時には、システムが変換のために適用したルールを再現することにより、ユーザの要求がどのような中間表現に変換されたのか、また中間表現からどのようなプログラムが生成されたのかを述べる。さらに、システムが行なったルールの適用を再現するだけでなく、変換ルールの作成時にルール記述者が考慮した領域の基本原則を用意し、どのような基本原則のもとに変換を行なったのかも述べる。原則に基づく説明により、変換の妥当性が理解でき、ユーザがプログラムを受け入れやすくなると考える。

本稿では、他のソフトウェア開発技法との比較を容易にするため、および実世界に依存した自然言語による仕様が扱えることを示すために、システムに与える仕様として酒類販売会社の在庫管理問題[7]を取り上げている。

2. JUSTIFY概要

JUSTIFYの概要を図1に示す。

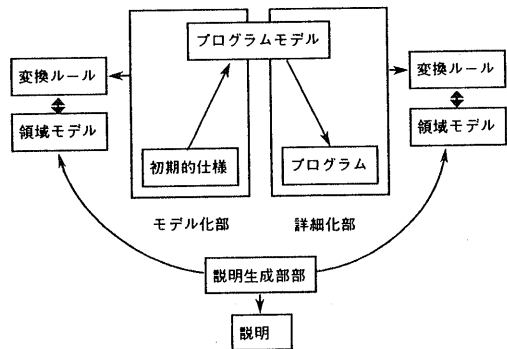


図1 JUSTIFY概要

JUSTIFYは、はじめにユーザの要求を具体的なレベルで記述した要求仕様を受け取る。この初期的な仕様は、プログラム合成の対象となる実体間の関係や、これらの実体に対する操作を記述することにより、ユーザの要求する問題を表現したものである。要求仕様を受け取ったシステムは、仕様解析部で仕様を解析し意味表現へと変換する。

計算機の内部では、実際に物理的に存在する実体をそのまま扱えるわけではなく、これらをデータとして抽象化して取り扱わなければならない。したがって、これらの具体的レベルで記述された処理をプログラム上で実現するためには、仕様内の具体的な実体に関する記述を、計算機上で操作可能なレベルにまで抽象化する必要がある。たとえば、在庫管理システムに関する仕様内に、倉庫内に物が保管されていることが記述されている場合には、倉庫を集合、保管されている物をその集合の要素として捉える必要がある。これを抽象化処理と呼ぶ。また、自然言語による仕様記述を許しているために、仕様には明示的に表現されない情報がある。これらの情報を補うことを補完処理と呼ぶ。仕様に対して抽象化処理、および補完処理を行うことによって、ユーザの要求した問題を形式的な表現(プログラムモデル)に変換するのがモデル化処理である。モデル化処理は、仕様解析部から仕様の意味表現を受け取ったモデル化部で行なわれる。

プログラムモデルは、一般的な計算機上で操作可能なように抽象化された表現であるから、そのままでは特定の処理系におけるプログラムとすることはできない。プログラムモデル内の抽象的にかかれた処理をプログラミング言語で表現するためには、データ構造や制御構造を満たすように、処理の内容の詳細を決定しなければならない。プログラミング言語上での詳細な制約を満たすために、プログラムモデル内の処理を詳しく書き換えることを、詳細化処理と呼ぶ。プログラムモデル内の記述を詳細化することによって、最終的なプロ

グラムが得られる。このように、プログラムモデルを受け取り処理を詳細化するのが詳細化部である。

合成されたプログラムに誤りが発見されたり、プログラムに不審を抱かれる場合には、システムの行なった合成過程をユーザに説明しなければならない。説明は、要求をモデル化する過程と、モデルからプログラムへと変換する過程のそれぞれに対して行なわれる。したがって、合成されたプログラム内に発見された誤りが、システムにユーザの要求が伝わらなかったために発生したのか、あるいは伝達されたがシステムのプログラム変換に誤りがあったためなのか明確に区別できるようになっている。説明の生成は、説明生成部で行なわれる。説明生成部では、モデル化及び詳細化の変換の際に適用されたルールと、ルールの基本となる原理をもとに説明を生成するようにしている。

3. 仕様解析

仕様解析の目的は、与えられた要求仕様から、プログラムの合成に必要な情報を計算機で取り扱える形で取り出すことである。仕様中に表現されている情報には図2のようなものがある。これらを図に示すような表現形式で抽出する。仕様解析部として、黒板モデルを用いたパーザ[3]を利用している。

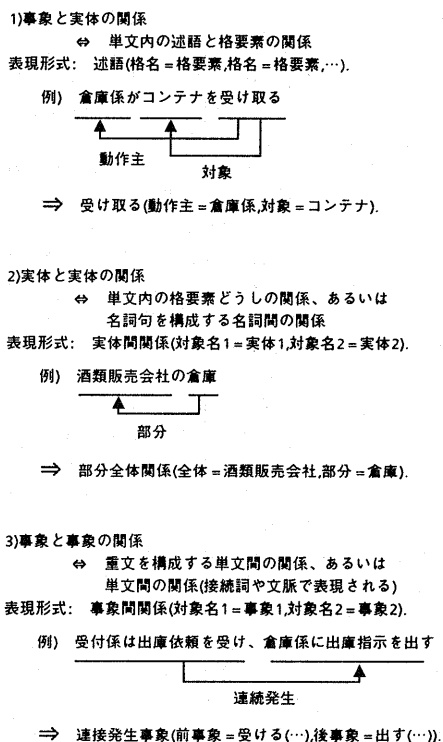


図2 仕様内の情報

仕様解析部の入力である自然言語文と、出力である意味表現の例を図3に示す。以降では、特にことわらないかぎり、仕様を解析して得られる意味表現を単に仕様と呼ぶことにする。

ある酒類販売会社に毎日数個のコンテナが搬入されてくる。受付係は毎日数10件の出庫依頼を受け、倉庫係へ出庫指示書を出す。在庫が出庫依頼の数量より少ない場合には、依頼者に在庫不足連絡を出し、在庫不足情報を在庫不足リストに記入する。受付係が依頼者に在庫不足連絡を出す処理をプログラム化せよ。

自然言語仕様

1. ある(対象=酒類販売会社A).
2. 部分全体関係(全体=酒類販売会社A, 部分=倉庫B).
3. 搬入する(終点=倉庫B, 対象=コンテナC*).
4. 受ける(動作主=受付係D, 対象=出庫依頼E*).
5. 出す(動作主=受付係D, 終点=倉庫係F, 対象=出庫指示書G*).
6. 連続発生事象(前事象=事象4, 後事象=事象5).
7. より少ない(対象=在庫H*, 比較対象=数量I*).
8. 部分全体関係(全体=出庫依頼E*, 部分=数量I*).
9. 出す(動作主=受付係D, 終点=依頼者J, 対象=在庫不足連絡K*).
10. 選択発生事象(発生条件=事象7, 事象=事象9).
11. 記入する(動作主=受付係D, 対象=在庫不足情報L*, 場所=在庫不足リストM).
12. 連続発生事象(前事象=事象9, 後事象=事象11).
13. プログラム(出す(動作主=受付係D, 終点=依頼者J, 対象=在庫不足連絡K*)).

意味表現

図3 自然言語仕様とその解析結果

4. モデル化处理

4.1 仕様の抽象化

モデル化处理では初めに、仕様内の各事象の意味を抽象的に捉えるために、仕様内の動詞をその意味によって分類し、概念表現へ変換する。この変換により、文の表層構造は異なるが、概念的には同一の事象を表現した文を同一視できるようになる。たとえば、「受付係は倉庫係に出庫指示を渡す」という文と「倉庫係は受付係から出庫指示を受け取る」という文は、動作の主体が受付係にあるか倉庫係にあるかが異なるだけで、対象の移動に関しては全く同一の事象を表している。これらの仕様を

「授受(動作主=受付係, 始点=受付係, 終点=倉庫係, 対象=出庫指示)」

および

「授受(動作主=倉庫係, 始点=受付係, 終点=倉庫係, 対象=出庫指示)」

と表現すれば、これらの仕様が対象の移動に関しては同じ意味を持つことがわかる。

抽象化のために必要な動詞の分類は、解析の際に利用する動詞辞書内に記述しておくのが自然である。実際、黒板モデルを用いたパーザで利用している動詞辞書IPAL[4]では、意味的分類を記載項目の1つとして取り上げている。このような辞書を利用することによって、抽象化处理は解析と同時に行うことができる。図3の仕様を抽象化した結果を図5に示す。

表4 仕様の分類

分類		格構造	対応する動詞の例
大分類	小分類		
状態	存在	対象,場所	ある
	属性	対象,属性,属性値	である
動作	移動	動作主,対象,始点,終点	搬入する,搬出する
	授受	動作主,対象,始点,終点	受け取る,渡す
	保管	動作主,対象,場所,キー	保管する,登録する
	検索	動作主,対象,場所,キー	探す,検索する
	生成	動作主,対象,場所	作る,作成する
	削除	動作主,対象,場所	取り除く,消去する
	比較	動作主,対象1,比較条件,対象2	比べる,比較する
	演算	動作主,対象1,演算子,対象2	加える,かける

- 1.存在(対象=酒類販売会社A).
- 2.部分全体関係(全体=酒類販売会社A,部分=倉庫B).
- 3.移動(終点=倉庫B,対象=コンテナC*).
- 4.授受(動作主=受付係D,終点=受付係D,対象=出庫依頼E*).
- 5.授受(動作主=受付係D,始点=受付係D,終点=倉庫係F,対象=出庫指示書G*).
- 6.連続発生事象(前事象=事象4,後事象=事象5).
- 7.比較(対象=在庫H*,比較条件=<,比較対象=数量I*).
- 8.部分全体関係(全体=出庫依頼E*,部分=数量I*).
- 9.授受(動作主=受付係D,始点=受付係D,終点=依頼者J,対象=在庫不足連絡K*).
- 10.選択発生事象(発生条件=事象7,事象=事象9).
- 11.保管(動作主=受付係D,対象=在庫不足情報L*,場所=在庫不足リストM).
- 12.連続発生事象(前事象=事象9,後事象=事象11).
- 13.プログラム(授受(動作主=受付係D,始点=受付係D,終点=依頼者J,対象=在庫不足連絡K*)..

図5 仕様の概念表現

4.2 プログラムモデル生成

プログラムモデルの生成時には、仕様を補完しながら仕様内の事象を連結する必要がある。本節では、はじめにプログラムモデルとは何かについて述べ、つぎに仕様の補完に必要な知識を説明し、最後に実際のプログラムモデルの生成方法を述べる。

4.2.1 プログラムモデル

プログラムモデルとは、プログラム内の各プロセスどうしがどのようなデータのやりとりによって通信を行なっているか、また各プロセスではどのような処理を行うのかを表現するプログラムの形式仕様である。実体と実体の関係および実体の状態を表現した仕様を、プログラムの扱うデータやプログラムを実行するときの環境として捉え、さらに動作を表現した仕様の動作主をプログラム内のプロセス、動作主の行なう操作をプロセス内の処理として捉えることにより、仕様をプログラムモデルと見ることができる。

プログラム内のプロセス間の通信に利用されるデータには、流動型データと蓄積型データの2種類がある。流動型データとは、1つのプロセスから1つのプロセスへ渡されるデータで、後者のプロセスのトリガとなるものである。流動型データによる通信は、移動、および授受の動的記述で表わされる。蓄積型データとは、1回の通信で消え去るのではなく、任意の時間に各プロセスが参照できるように、プログラム内部に保存しておかなければならないデータのことである。蓄積型データによる通信は、検索、保管、および削除の動的記述で表わされるように、蓄積型データへの書き込み・読み取りによって表現されている。これらのデータのやりとりをもとに、各プロセスとデータの動きの関係をグラフで表現したものを、各プロセス間の外部仕様とする。また各プロセス内の処理は、事象間の関係をもとに接続、選択、反復として並べることによって表現できる。この並びをプロセスの内部仕様とする。

具体的なプログラムモデル、およびそのの生成手法は4.2.3項で述べる。

4.2.2 仕様補完知識

ユーザの記述した仕様には、自然言語で記述されていること、問題の整理が行なわれていない初期的なものであることなどの理由により、断片的であったり、情報が欠落している場合がある。これらの情報の欠落は、問題領域に関する常識や事象の連結性に関する知識を前提として簡略的に記述されたために発生したと考えられる。常識や事象の連結性を前提とした情報の欠落は、システムがそれらの常識や事象の連結性に関する知識をユーザと共有することにより補完することができる。JUSTIFYでは、問題領域に対する常識的知識をネットワークとして表現し、これを問題領域の基本原理(領域モデル)として持っている。領域モデルは、対象とする問題領域内にはどのような実体が存在するのか、実体の間にはどのような依存関係があるのか、実体に対してどのような操作が行われるのか、操作によって問題世界がどのように変化するかを記述したものである。

また、ある動作の発生によってどのような状態が引き起こされるのか、ある状態が発生する前にどのような動作が行われたのかなどの事象の連結性を推論するための知識を、一般的な事象のシーケンスで表し事象列モデルとして用意している。これらの知識を参照することにより、仕様内の欠落している情報を補うことが可能となる。在庫管理に関する領域モデルを図6に、事象列モデルを図7に示す。

しかしながら、情報の欠落を補う際に、領域モデルや事象列モデルを直接参照すると、処理が複雑になり、システム化が困難となるばかりか、実行速度の低下も免れない。これらの問題を解決するために、仕様記述時に発生する可能性のある情報の欠落を想定し、それぞれの欠落に対して、システムが行なうべき補完処理をルールとして用意する。これらのルールはシステムを動作させるための知識と見ることができ、理解に必要な詳細な情報が失われてしまっており、ルールだけではその意味(ルール内で言及された事象や実体を持つ意味やそれらの間に存在する関係など)を理解するこ

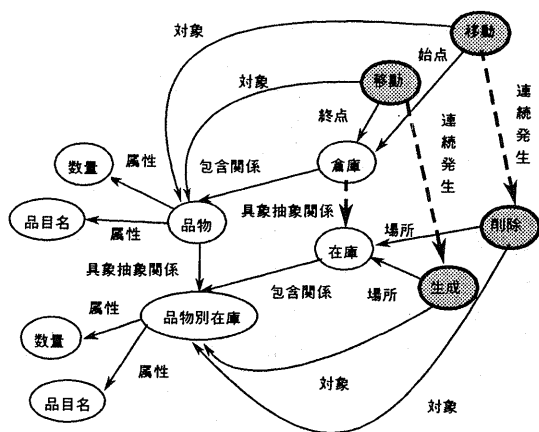


図6 在庫管理の領域モデル

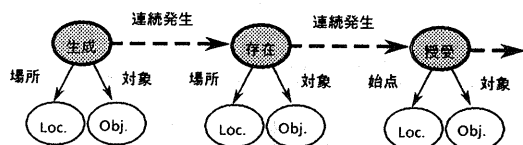


図7 事象列モデル

とはできない。したがって、説明を生成する際にはルールだけでなくこれらのモデルも参照する必要がある。JUSTIFYではルールを用意するとともに、説明のためにモデルもシステムに持たせている。領域モデルに基づくルールを領域知識補完ルール、事象列モデルに基づくルールを前提付加ルールおよび帰結付加ルールと呼ぶ。これらのルールを、次項のプログラムモデル生成の過程で適用することにより、仕様の補完が行われる。領域知識補完ルールの例を図8a)に、前提付加ルールの例を図8b)に示す。

4.2.3 プログラムモデルの生成手順

図5の仕様に例に、プログラムモデルの生成手順を説明する。

- ① システムは最初に仕様の中から特別な述語である「プログラム(授受(動作主=受付係D,始点=受付係D,終点=依頼者J,対象=在庫不足連絡K*))」という記述を探し出す。プログラム述語の引数は、プログラム化すべき事象のリストである。この例では簡単のために事象は1つとしている。プログラム述語が発見できない場合は、「プログラム(「仕様の内のすべての事象のリスト」)」をシステムが生成し処理を続ける。
- ② プログラム述語内の事象のリストから1つの事象を取り出し、仕様内からこの事象について記述されている部分

```

if current-task = 事象列生成(存在(対象 = 在庫))
then
  create-event(存在(対象 = 在庫)),
  create-event(連続発生事象(前事象 = 存在(対象 = 在庫),
    後事象 = 存在(対象 = 在庫))),
  create-event(移動(終点 = 在庫,対象 = 品物)),
  create-event(生成(対象 = (品物,数量),場所 = 在庫)),
  create-event(連続発生事象(前事象 = 移動(終点 = 在庫,対象 = 品物),
    後事象 = 生成(対象 = (品物,数量),場所 = 在庫))),
  create-event(移動(始点 = 在庫,対象 = 品物)),
  create-event(削除(対象 = (品物,数量),場所 = 在庫)),
  create-event(連続発生事象(前事象 = 移動(始点 = 在庫,対象 = 品物),
    後事象 = 削除(対象 = (品物,数量),場所 = 在庫))).

```

a)領域知識補完ルール

```

if current-task = 事象列生成(授受(対象 = X))
then
  create-event(存在(対象 = X)),
  create-event(連続発生事象(前事象 = 存在(対象 = X),
    後事象 = 授受(対象 = X))).

```

b)前提付加ルール

図8 仕様補完ルール

を探し出す。この検索は、プログラム述語内の事象と、仕様内の事象とのマッチングにより行われる。例では、事象9とのマッチングに成功する。

- ③ 事象9に関する事象間関係により、仕様に明に表現された事象列を生成する(図9)。仕様に完全に事象列が記述されている場合には、事象列は他のプロセスや外部からのデータの授受・移動からはじまり、他のプロセスや外部へのデータの授受・移動あるいは蓄積型データへのデータの保管・削除で終わっているはずである。図9の事象列がこれらの条件を満たしているかどうか、つまりプロセスの内部仕様が完全なものであるかどうかを判定する。この場合、事象列の終了条件は満たしているが、開始条件が満たされていない。このような不完全な仕様に対して、④において前提付加ルール・帰結付加ルールを適用し補完を試みる。
- ④ 開始条件が満たされていない事象列に対しては、先頭の事象に前提付加ルールを適用することにより、仕様に暗に記述されている事象間の関係を推論する。この場合図8b)のルールが適用され、図10の新しい事象列が生成される。この事象列に対して、③の処理を繰り返し、再び仕様に明に記述されている事象列の探索を行う。
- ⑤ ③,④の処理は、事象が完全に抽出されたときに打ち切られる。前提付加ルール・帰結付加ルールの適用によっても完全な事象列が生成できなかった場合は、⑥において領域知識補完ルールを適用する。図10の事象列は前提付加ルールを適用したにもかかわらず完全なものではないので⑥の処理を受ける。
- ⑥ 図10の事象列に対して図8a)の領域補完知識を適用し、図11の事象列を得る。このルール適用により、仕様内では常識として言及されなかった在庫に関する知識が補完される。図11の事象列にさらに③,④の処理を行い出庫処理に関する完全な事象列を生成する。最後に、事象の

連結性を推論するために生成した状態を表す事象を取り除き、出庫処理プロセスの内部仕様とする(図12a)。

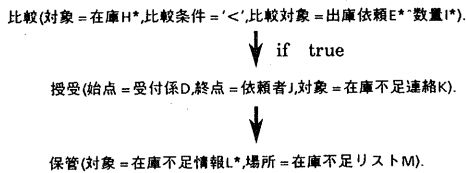


図9 プログラムモデル生成過程(1)

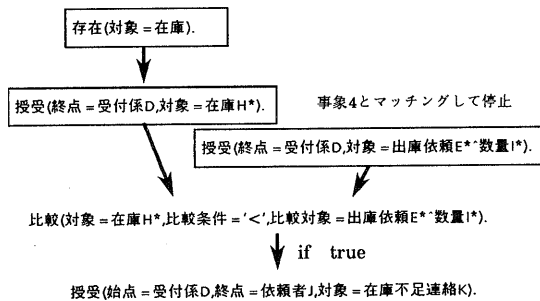


図10 プログラムモデル生成過程(2)

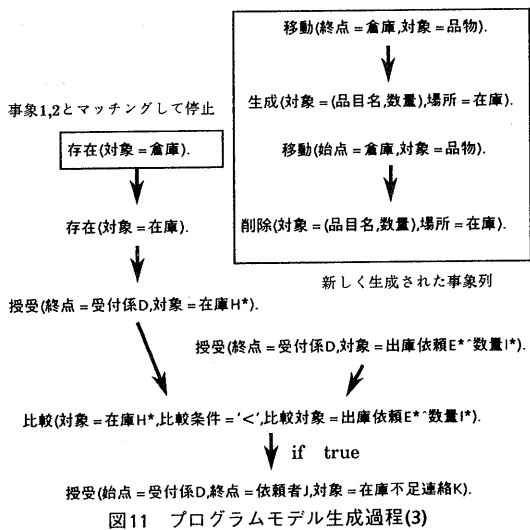
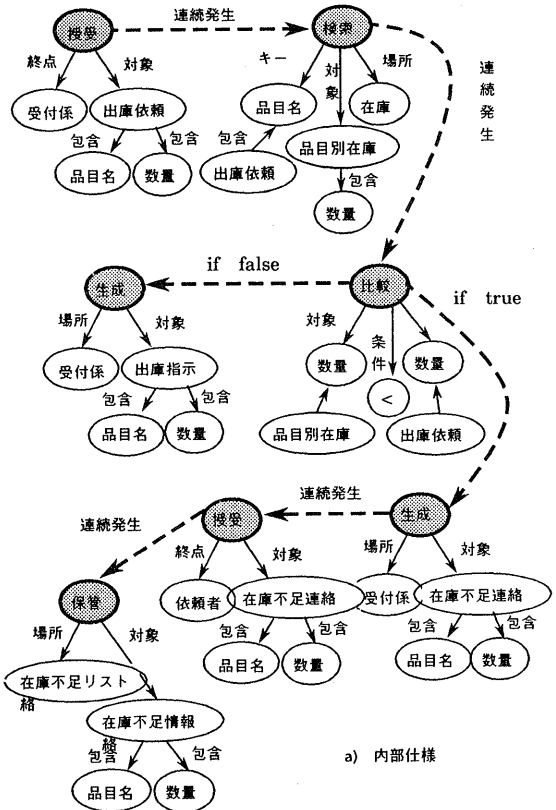


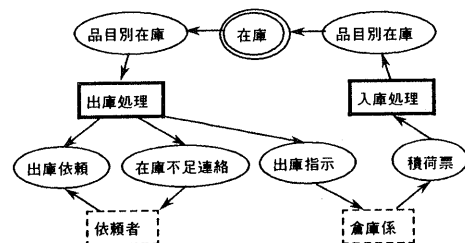
図11 プログラムモデル生成過程(3)

図5の仕様では省略したが、共通問題では入庫処理に関するプロセスもあり、上と同様の処理を行って入庫処理の内部仕様を作成する。このようにして作成された内部仕様を、データのやりとりをもとに連結することによって、外部仕様を作成する。外部仕様の例を図13b)に示す。図中の一重の楕円は流動型データを表わし、二重の楕円は蓄積型データを表わしている。また、プロセスは矩形で表現している。流動型データによる通信は、送信側のプロセスから受信側のプロセスに矢印を引き、その間に流動型データを置くことで表現している。蓄積型データによる通信は、蓄積型データへの挿入や削除などの書き込みを行なう場合には、プロセスから蓄積型データに向かって矢印が引かれ、検索などの読み取りを行なう場合には、蓄積型データからプロセスに向かって矢印が引かれている。言い換えると、すべての矢印の向きは、データが移動する向きに対応していると言える。

スに矢印を引き、その間に流動型データを置くことで表現している。蓄積型データによる通信は、蓄積型データへの挿入や削除などの書き込みを行なう場合には、プロセスから蓄積型データに向かって矢印が引かれ、検索などの読み取りを行なう場合には、蓄積型データからプロセスに向かって矢印が引かれている。言い換えると、すべての矢印の向きは、データが移動する向きに対応していると言える。



a) 内部仕様



b) 外部仕様

図12 プログラムモデル

5. 詳細化処理

プログラムモデルはユーザの要求をまとめたものであるが、これを具体的なプログラミング言語へと変換しなければ計算機上で実行することは不可能である。このために、プログラムモデル内の各処理に対して、プログラミング言語で表現するために必要な事項を決定していく。

はじめに、プログラムモデル内の各プロセスを詳細化するために、1つのプロセスに対する内部仕様を取り出す。次にこの内部仕様の中から、1つの処理を表わすノードを取り出す。この処理を表わすノードには、処理に関係するデータが、処理との関係を付加したリンクでつながれている。たとえば、検索処理では、検索を行う場所、検索のためのキー、検索の結果である対象が、検索処理を表わしたノードにつながれている。このような処理を表現したグラフに対して、詳細化ルール、プログラム化ルールを適用することによりグラフを詳細化し、処理内容を詳細化していく[2]。詳細化ルールはプログラミング言語に依存しないものであり、プログラム化ルールは特定のプログラミング言語に依存したものである。たとえば、「データを格納するためには集合の概念を用いる」などは言語に依存しない詳細化ルールであり、「集合はリスト記法で表わす」などは言語に依存したプログラム化ルールである。詳細化処理の前半では、詳細化ルールを適用し、計算機上で処理を実現するのに一般的に必要な事項を決定する。このようなルールの適用が終わった段階で、最後に目標とするプログラミング言語に依存したプログラム化ルールを適用し、プログラムモジュールとする。

詳細化の前半で適用される詳細化ルールは、プログラム構造に関する基本原理に基づいて記述されている。また、後半で適用されるプログラム化ルールは、プログラミング言語の基本原理から導かれるものである。プログラム構造に関する基本原理と詳細化ルールの関係、およびプログラム言語に関する基本原理とプログラム化ルールの関係は、問題領域に関する基本原理と仕様補完ルールの関係と同等である。すなわち、プログラムとして処理を実現する場合に、その処理に関して言及されるべき情報と、それらの情報間の関係を記述したものがプログラム構造の基本原理であり、この基本原理をもとにシステムが行なうべき処理をルール化したものが詳細化ルールである。また、プログラミング言語上での表現に関する制約条件などを記述したものが、プログラミング言語に関する基本原理であり、これをシステムで扱えるようにルール化したものが、プログラム化ルールである。

6. 質問と説明

6.1 質問

システムとユーザの対話例をもとに、ユーザがシステム利用時に発する質問がどのようなものであるかを説明する。

ユーザ： 要求仕様(図3)をシステムに渡す。

システム： 合成したプログラムをユーザに提示する。

ユーザ： 「合成過程の概要を示せ。(質問①)」

システム： 「要求仕様(図3)をプログラムモデル(図12)に変換しプログラムを合成した。」

ユーザ： 「プログラムモデル内の在庫は何を意味するか。(質問②)」

システム： 「在庫は倉庫を抽象化したものである。

在庫内の品目別在庫は倉庫内の品物を抽象化したものである。品目別在庫は品目名と数量という属性を持つ。

倉庫に品物が搬入されると在庫に品目別在庫が生成される。

倉庫から品物が搬出されると在庫から品目別在庫が削除される。」

ユーザ： 「受付係は出庫依頼を受けると何をするか。(質問③)」

システム： 「在庫から品目別在庫を出庫依頼の品目名で検索する。もし品目別在庫の数量が出庫依頼の数量より少ないならば、在庫不足連絡を生成し、在庫不足連絡を依頼者に渡し、在庫不足リストに在庫不足情報を保管する。でなければ出庫指示を生成する。」

ユーザ： 「プログラムモデル内の検索はプログラムではどのように書かれているか。(質問④)」

システム： 「検索(Key,[Head|Tail],Head):-……」

ユーザ： 「検索(Key,[Head|Tail],Head) の第2引数はプログラムモデル内の何に対応するのか。(質問⑤)」

システム： 「在庫である。」

ユーザ： 「プログラムモデル内の在庫はプログラムではどのように書かれているか。(質問⑥)」

システム： 「在庫([品目別在庫(Hinmoku,Suuryou)]).」

上記の対話内の質問をまとめると以下ようになる。

- ① システムが行った処理の概要を求める質問
- ② システムの持つ常識的知識の説明を求める質問
- ③ 入力事象に対する処理の流れを求める質問
- ④ プログラムモデル内の事象とプログラムの対応を求める質問
- ⑤ プログラム内の変数とプログラムモデル内の実体の対応を求める質問
- ⑥ プログラムモデル内の実体とプログラム内のデータ構造の対応を求める質問

これらの質問に対する説明生成の方法を次節で述べる。

6.2 説明

前節における①~⑥の質問に対し、以下の方針で説明を生成する。

- ① システムが行った処理概要の説明
ユーザの記述した要求仕様と、システムが生成したプログラムモデルおよびプログラムを提示する。
- ② システムの持つ常識的知識の説明
領域モデルを自然言語に変換する。
- ③ 入力事象に対する処理の流れの説明
プログラムモデルから質問文内の事象に続く事象列を抽出し、これを自然言語に変換する。

- ④ プログラムモデル内の事象とプログラムの対応の説明
詳細化の過程をたどり、質問文内の事象に対応するプログラムコードを示す。
- ⑤ プログラム内の変数とプログラムモデル内の実体の対応の説明
詳細化の過程を逆にたどり、質問文内の変数に対応する実体を示す。
- ⑥ プログラムモデル内の実体とプログラム内のデータ構造の対応の説明
詳細化の過程をたどり、質問文内の実体に対応するデータ構造を示す。

②を例として説明がどのようにして生成されるかを述べる。
ユーザがプログラムモデル内の在庫を指定し、それが何を意味するかと質問した状況を考える。

ユーザ:「プログラムモデル内の在庫は何を意味するか」

- 「在庫は倉庫を抽象化したものである。 ①
在庫内の品目別在庫は倉庫内の品物を抽象化したものである。 ②
品目別在庫は品目名と数量という属性を持つ。 ③
倉庫に品物が搬入されると在庫に品目別在庫が生成される。 ④
倉庫から品物が搬出されると在庫から品目別在庫が削除される。 ⑤」

①は図6内の倉庫と在庫の関係を自然言語に変換したものである。②は倉庫と品物の関係、在庫と品目別在庫の関係、および品目と品目別在庫の関係から生成された文である。③は品目別在庫と品目名および数量という属性の関係に対応する。④、⑤は在庫を対象とする事象を自然言語にしたものである。このように、システムが在庫に関して持っている常識的知識を説明することができる。

7. おわりに

7.1 まとめ

プログラム合成システムにおける、基本原理に基づく説明手法を提案した。本手法の特徴を以下に挙げる。

① 原理に基づいた説明

システムの行なった合成を説明する際に、システムの行なった処理を示すだけでなく、その処理の根拠となった基本原理をもとに説明をしている、したがって、説明を受けたユーザが、システムの行なった処理の意図を理解でき、処理の妥当性が判断しやすくなるという利点がある。JUSTIFYでは、基本原理として、プログラムが対象とする問題領域の基本原理、プログラム構造に関する基本原理、およびプログラミング言語に関する基本原理の3種類を用意している。

② プログラムモデルを用いた要求定義

JUSTIFYでは、自然言語で記述されたユーザの要求を、プログラミング言語にも自然言語にも依存しないプログラムモデルにまとめあげている。プログラムモデルはユーザにとって馴染みやすいグラフとして表現している。このよう

に、ユーザの要求を曖昧性がなく理解しやすいプログラムモデルにまとめることにより、ユーザは自身の要求をはっきりと認識でき、システムはユーザの要求を正確に受理することができるという利点がある。また、合成過程の説明を、要求を抽象的なモデルとして定義する部分と、モデルから具体的なプログラムへと変換する部分に分けて行なうことができ、合成されたプログラム内に誤りが発見されたときに、誤りの原因がユーザの仕様にあったのか、あるいはシステムの行なった合成にあったのかを区別できるという利点がある。

7.2 今後

ユーザがシステムに対して発する質問の1つに、具体的な入力に対してプログラムがどのように動作するのかという質問がある。たとえば、

「倉庫にoldが10本あったとする。出庫依頼(品目=old,数量=15)を受付係に渡すと、受付係はどうするか」

というような質問が考えられる。このような質問に対して、合成したプログラムを実行し、そのトレースを提示すれば説明とすることができる。しかしながら具体的なプログラムのトレースでは、問題の本質的な部分以外の詳細な処理まで説明に現れてしまい理解しづらい。これに対してプログラムモデル上でプログラムの動作をシミュレートすれば、本質的な動作のみからなる説明を生成することが可能である。処理の本質を捉えた説明を生成することにより、合成されたプログラムがユーザの意図にあったものかどうかの判断が容易になると思われる。今後、プログラムモデル上でシミュレーションを行うための専用の実行系を設計する予定である。

参考文献

- [1] 有沢誠: ソフトウェアプロトタイピング, 近代科学社, (1986).
- [2] Barstrow, D.: A Knowledge-Based System for Automatic Program Construction, *Proc. of 5th IJCAI*, pp.382-388, (1979).
- [3] 一本木 他: 黒板モデルの導入により戻り処理を伴わないべた書き日本語文の解析, 情報処理学会第34回全国大会講演論文集, 3w-1, (1987).
- [4] 情報処理振興事業協会技術センター: 計算機用日本語基本動詞辞書IPAL(Basic Verbs), 情報処理振興事業協会, (1987).
- [5] 松元, 上原, 豊田: プログラム合成システムにおける知識の適用状況を用いた説明, 第1回人工知能学会全国大会論文集, 9-2, (1987).
- [6] Neches, R., Swartout, W.R., and Moore, J.D.: Enhanced Maintenance and Explanation of Expert Systems Through Explicit Models of Their Development, *IEEE Trans. on Software Engineering*, Vol.SE-11, No.11, pp.1337-1350, (1985).
- [7] 山崎利治: 共通問題によるプログラム設計技法解説, 情報処理, Vol.25, No.9, pp.934, (1984).