プログラムと仕様からの設計上の決定の抽出

1G−6

福田 剛志, 深沢 良彰, 門倉 敏夫

早稲田大学 理工学部

1 始めに

通常の人手によるプログラム開発は、仕様に対して設計者やプログラマが行なう設計上の決定を与えることにより行なわれる。この設計上の決定とは、仕様に対するデータ構造やアルゴリズムの選択である。このことから、“**プログラム＝仕様＋設計上の決定**”と言える。

このように考えると、プログラム中に存在する設計上の決定とはプログラムと仕様の差(設計上の決定＝プログラム－仕様)である。もし、プログラムから設計上の決定を分離することができれば、これは次の様なことに利用可能である。

- 設計上の決定を変更して、元の仕様に対して適用すれば、容易に別の決定を持つプログラムを合成することができる。
- 設計上の決定は仕様とプログラムのギャップであるから、これを人間に提示することにより、プログラムと仕様の可読性を高めることができる。

そこで本発表では、プログラムとその仕様から、設計上の決定を抽出する方法を提案する。

2 本研究の概要

設計上の決定とは抽象的な仕様を具体化する方針である。そこで、本研究では仕様とプログラムとの対応をとりながら、クリシェ(*cliche*)[1]を利用してプログラムの抽象化を行なうことにより、設計上の決定の抽出する。

人間がプログラムを記述するときは、慣用句的な記述を多用しているので、クリシェの当てはめによるプログラムの抽象化には、可能性があると考える。

本システムの構成を図1に示す。本システムは、仕様とプログラムを入力とし、システム内に用意されているクリシェライブラリとそれに関するメタ知識ベースを利用して、プログラム設計上の決定を出力する。

本研究で述べる設計上の決定とは次のようなものである。

- クリシェ同士の接続関係

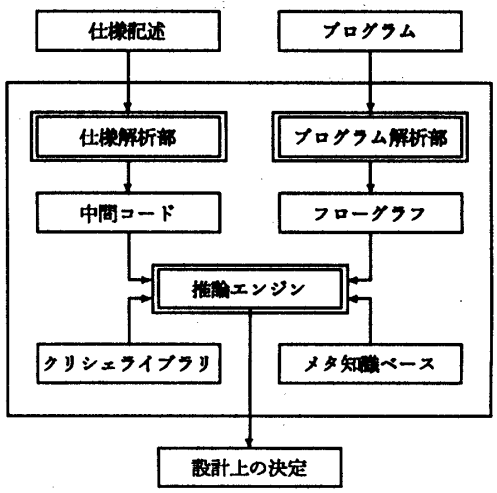

図 1: 本システムの構成

- 抽象的クリシェからの具体的クリシェの選択過程
- 仕様中のデータに対するプログラムでの表現

仕様に含まれる情報を利用する為には、仕様とプログラムの対応をとる必要がある。現在のところ仕様記述中の操作名と同名の関数がプログラム中に存在することを仮定し、これにより対応をとっている。

本システムは、仕様記述言語に我々の研究室で開発中のSOLARIS、プログラム言語にCを採用するが、本手法自体が言語に依存するものではない。

3 知識

3.1 クリシェライブラリ

不必要な情報による混乱を避けるために、クリシェを本質的な部分だけのフローグラフとしてライブラリ中に格納する(図2参照)。

クリシェの集合は、抽象的なクリシェを導入することにより、それが含む設計上の決定の抽象度により階層をなし、決定の種類で枝分れするような構造を作る(図3参照)。また、この構造を利用して関連するクリシェの検索を効率化することができる。さらに、設計上の決定の変更を行なう際、このクリシェライブラリの構造を利用することができる。

3.2 メタ知識

クリシェライブラリ中から、プログラム中に使用されているクリシェを全ての可能性について検索するこ

Extraction of design decision from program and specification

T. Fukuda, Y. Fukazawa, T. Kadokura

Waseda University

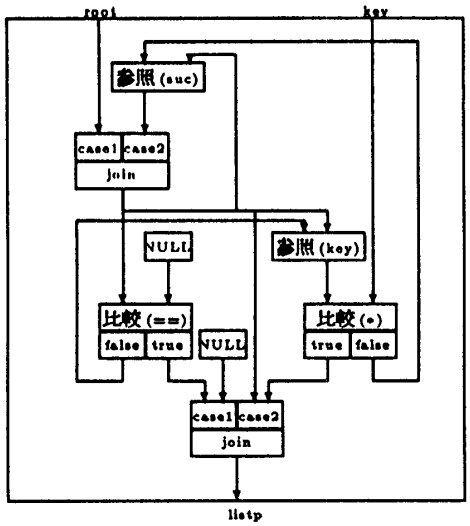

図 2: クリシェの例: リストの数え上げサーチループ

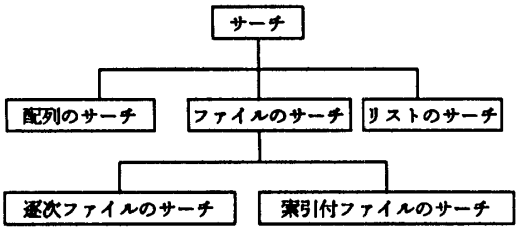

図 3: クリシェライブラリの階層構造の例

とは現実的ではない。

しかし、本システムはプログラムと仕様の対応をとっているので、プログラム中の操作に対する入出力条件やデータ表現から、使用されているクリシェを絞ることができる。

プログラムと仕様の状態から各クリシェが使用されている可能性を確信度として計算するメタ知識を、クリシェライブラリのノード毎に用意しておく。推論エンジンはこれを利用して、可能性の高い順に、対応するプログラムに対してクリシェの適用を試みる。

4 本システムの動作

図1において、仕様解析部とプログラム解析部は推論の為の前処理として、仕様を構文解析し、プログラムをクリシェと同様のフローグラフに変換する。

推論エンジンは、入力されたプログラムに対して用意されたクリシェライブラリの中から適切なクリシェの当てはめを行ない、クリシェをノードとするフローグラフを得る。さらに、その中の個々のクリシェを、クリシェライブラリの構造を利用して、抽象的なクリシェに置き換える。

本推論エンジンは専用のプロダクションシステムで、その大まかな動作は次の様である。

1. 仕様とプログラムの対応をとりながら、プログラムと仕様の状態を生成する。
2. これに対してメタ知識を用いて、クリシェライブラリのノードに確信度をつける。
3. 適用できるクリシェがなくなるまで、確信度の高いクリシェから順にプログラムへの適用を繰り返す。

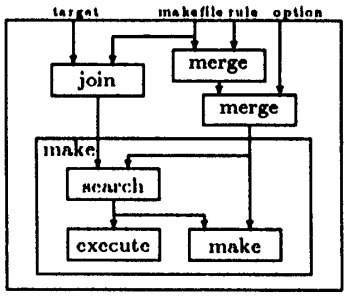

図 4: makeプログラムに対する抽象クリシェの接続図

以上の過程により、2章で列挙した設計上の決定を得る。

例として、本システムをUNIXのmakeプログラムとその仕様に対して適用した結果のクリシェの接続関係を図4に示す。これと、具体的クリシェの選択とデータ表現の対応を合わせたものが、本システムの出力する設計上の決定である。

これらは、仕様で与えられた入出力関係に対するプログラムよりも、抽象度が高い操作の接続関係であるため、仕様とプログラムの差を埋めるものとなり得る。また、図1の例にある再帰構造を繰り返し構造に変更するなどの、制御構造の変更を与えることも容易である。データ表現を隠蔽しているため、データ表現の変更により、比較的単純に新たな設計を考えることもできる。

5 終りに

今後の課題として

- クリシェに依るプログラムの理解の有効性と限界を確かめる
- クリシェライブラリの整備と共に、不足しているクリシェの有効な獲得方法を考案する

などが考えられる。

さらに、本システムにより出力される設計上の決定を、1章で述べたように利用するシステムを構築する予定である。

参考文献

[1] Richard C. Waters, *"Program Translation via Abstraction and Reimplementation"* IEEE Trans. Software Eng. vol. 14, No.8, Aug. 1988