概念データモデルで記述された仕様の関数型言語への変換

2B－3

横田 和久　小林 吉純　佐藤 正和　太田 理
ATR 通信システム研究所

1　はじめに

ソフトウェアの生産性と信頼性を向上させるため，プログラム仕様の再利用性が重要である．筆者らは仕様の再利用性を向上させるためのERモデルと従属性制約を適用し，理解性，拡張性に優れた非手続き的なプログラム仕様記述言語PSDLと，そのコンパイラをに提案した[1].

また，PSDLを用いたプログラム仕様の開発支援システムSoftReuseを作成し，ソフトウェア記述実験を行ない生産性の向上を確認した．

これらの環境を使ったプログラムの開発においては，プログラムは自動生成されるのでその段階での誤りの混入はない．しかし，プログラムの元となるPSDL仕様に誤りが存在した場合には，その修正が必要になるが，それは簡単なことではない．そのため，仕様の誤りを見つける仕組みを開発する必要がある．そこで，我々は関数型言語に着目し，仕様の誤り発見に優れた，PSDLを関数型言語へ変換する方法を開発した．その手法について報告する．

2　プログラム仕様記述言語PSDL

PSDLで記述されたプログラム仕様は，図1に示すように，情報層，データ層，アクセス層の3階層で構成される．そのうち，アクセス層には入出力ファイルのアクセス方法を記述し，データ層には入出力データのデータ構造を記述する．

一方，情報層には，ERモデルを用いて，対象世界の情報の枠組みを記述する．対象世界の中の「もの」や「こと」を実体と呼び，実体の集合を実体型と呼ぶ．実体相互の対応づけは関連と呼び，関連の集合を関連型と呼ぶ．個々の実体の性質は属性値の組で表される．

プログラム中で起きる計算は以下の3種類の従属性制約で記述する．属性値従属性制約は実体の属性値を，その実体と関連で対応づけられた他の実体の属性値から得るための計算式である．関連存在従属性制約は，実体を相互に対応づける関連を，それらの実体の属性値から得るための計算式である．実体存在従属性制約は，実体を，他の実体の属性値から得るための計算式である．この場合，それらの実体を相互に対応づける関連も同時に得られる．

3　PSDLプログラム仕様の解析

PSDLプログラム仕様を関数型言語へ変換するためには，それが表している計算内容を解析する必要がある．ここでは，仕様上のデータの依存関係を表す有向グラフ(図3)を作成し，このグラフから関数型言語の構成

Translation from specification described with conceptual data model to functional language

Kazuhisa Yokota, Yoshizumi Kobayashi, Kazumasa Sato, Tadashi Ohta
ATR Communication Systems Research Laboratories

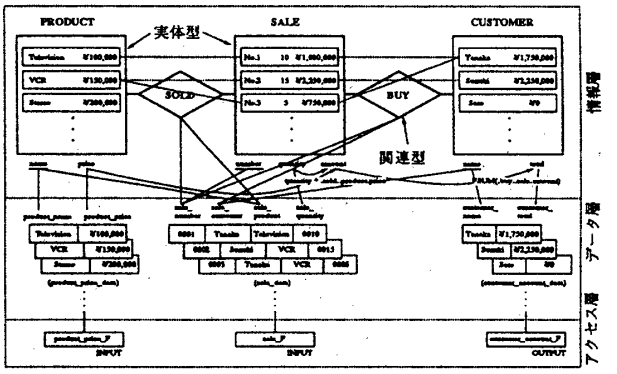

図 1: PSDL プログラム仕様の例

要素へ変換を行なう．以下，PSDLプログラム仕様の解析方法の概略を示す[1].

(1) 有向グラフ作成

グラフの節点はPSDL文に従って集める．PSDLの構成要素である実体型，関連型，従属性制約等に対応して節点を設ける．それらの節点の間には，従属性制約によって生じるデータの依存関係に従って有向枝を張る．枝の方向は，情報層では実体，属性値および関連が制約から参照される方向と，制約からデータが得られる方向に合わせる．また，データ層とアクセス層ではデータが入力ファイルから流入する方向と，出力ファイルへ流出する方向に合わせる．

また，有向枝は節点に流入するデータの流れが同一節点に流入する他のデータの流れと同期しているか否かにより同期型と非同期型に分類される．

(2) 局所的解析

実体型節点に2本以上の枝が流入していれば，同じ実体が異なる順序で流入するため，それらの枝を非同期型にする．また，それらの実体型の属性節点へ流入している枝も非同期型にする．関連型節点についても同じである．

実体型節点または属性節点と，属性値従属性制約節点または実体存在従属性制約節点との間にある枝は，関連によって相互に結ばれた実体の数量関係と対応している．そこで，1つの実体に相手の実体が2つ以上結ばれていれば，同じデータが2回以上参照されるのでその枝を非同期型にする．

(3) 大局的解析

図2(1)に示すような方向混在閉路上で，同図(2)に示すように非同期型有向枝がすべて同じ方向を持っていれば，制約の実行順序に矛盾が起きる．その理由は非同期型有向枝から流入するデータ流と，同期型有向枝から流入するデータ流を同期させる処理を行なわなければならないためである．この制約実行順序の矛盾を解消するには，図2(3)に示すように反対方向の枝を少なくとも1本は非同期型に変えればよい．

(4) 連結部分グラフ分割

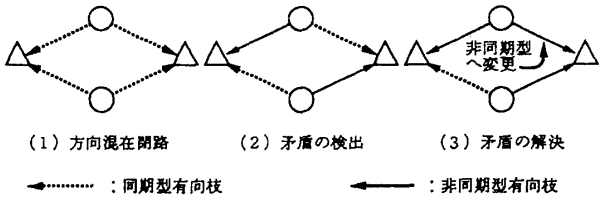

図 2: 閉路解析による矛盾の検出と解決

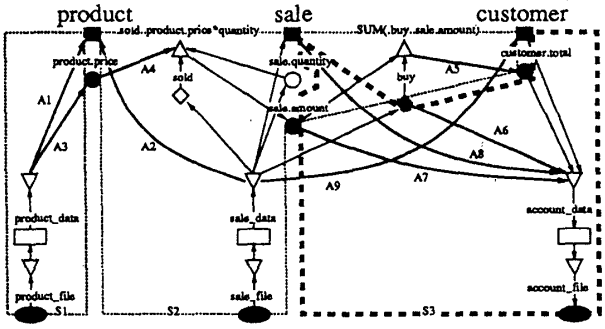

図 3: 図 1 の仕様の解析後の有向グラフ

グラフを，以下のような同期型連結部分グラフに分割する．まず，非同期型有向枝につながっている実体型と属性と関連型の節点を非同期型節点として，その他の節点は同期型節点とする．次に，非同期型節点を部分グラフの境界点に置き，同期型節点は部分グラフの内部に置く．節点を介してつながっている同期型枝よりのデータの流れは相互に同期しているので，同じ部分グラフの中に置く．そうすると，部分グラフの中ではデータの流れがすべて同期する．

4 関数型言語の生成

上記の解析で得られた，有向グラフから関数型言語を生成する概略を以下に示す．

(1) 構造節点の変換

PSDL 上でのデータの表現である実体型や関連型は，集合の性質を持っているので，関数型言語上では複数の要素を持つことができるリストで表現する．

1. 実体型
   実体は複数の属性が集まったものであるので，このまとまりをリストで表す．実体型は実体の集合であるので，実体のリストをさらにリストにして表現する．すなわち実体型はリストを要素として持つリストになる．また，PSDL の言語仕様により，実体型は属性の中の主キー属性の値によって一意に区別できる必要がある．このことを実現するため，実体型に実体を追加する操作は，主キー属性の重複を検査する関数を通して行なう．実体型の構造は下記の式のようになる．

$$E = [[\, key_1, attr_1, attr_2, \ldots, attr_n \,], \ldots]$$

$key_n \cdots$ 主キー属性

$attr_m \cdots$ 属性

2. 関連型
   関連は，2 つ実体間のつながりを表す．関連型はこのつながりの集合である．そこで関連を，関係付ける 2 つの実体の主キー属性を要素とする長さ 2 のリストで表す．関連の集合である関連型は，関連のリストで表す．関連型も集合としての性質を満たすために，関連の追加を行なう場合は重複を排除する関数を通して行なう．関連型の構造は下記の式のようになる．

$$R = [[\, key_{11}, key_{21} \,], [\, key_{12}, key_{22} \,], \ldots]$$

$key_{1n} \cdots$ 実体型 1 の主キー属性

$key_{2n} \cdots$ 実体型 2 の主キー属性

(2) 制約節点の変換

制約節点は計算を表す節点である．そこで，流入枝に対応するデータを引数とする関数へ変換する．関数の内容は，制約節点に対応する制約式によって決まる．PSDL には，3 つの制約があるが属性値従属性制約の変換についてのみ示す．

1. 属性値従属性制約
   属性値従属性制約は，実体の中の属性の値を決める制約である．この制約は関連と関連が結び付ける二つの実体型を引数とする．関数の戻り値は，属性の値が決まる方の実体である．関数の引数は以下のようになる．

$$e = F_{ad}\ r\ E_1\ E_2$$

   ここで，E_1, E_2 は入力となる実体型，r は E_1, E_2 中の要素を指定する関連である．F_{ad} は r が指し示す要素を E_1, E_2 中から探し，属性値従属性制約で指示される計算を行ない，新しい属性値を持った実体 e を生成する．F_{ad} の形は制約の内容により決まる．

(3) 部分グラフの関数への変換

解析の結果得られた有向グラフは，非同期型節点を境界とした連結部分グラフに分かれている．この非同期型節点を上記に示した方法でリストで表し，各連結部分グラフ内の制約節点も上記の方法で，関数で表す．この関数は，関連を一つしか処理できないので，関連のリストを処理するために再帰呼び出しを用いて，関連のリストである関連型を処理できるようにする．以上により，連結部分グラフを関数に変換できる．ところで，各連結部分グラフ間には非同期節点を共有することで依存関係ができている．そのため，変換してできた関数全体も相互に依存関係を持ち，PSDL プログラム仕様の処理内容を表している．すなわち PSDL から関数型言語への変換ができたことになる．

5 まとめ

非手続き的な言語である PSDL プログラム仕様を関数型言語へ変換する方法を提案した．関数型言語へ変換された仕様は，参照透過性などの関数型言語の特有の性質を備えており，もとの PSDL プログラム仕様に比較して，仕様の性質が調べやすくなっている．今後は，関数型言語上での仕様の検証手法の研究を行なう予定である．

参考文献

[1] M. Hashimoto, K. Okamoto: A Set and Mapping-based Detection and Solution Method for Structure Clash between Program Input and Output Data, *Proc. COMPSAC'90*, pp. 629-638, 1990.