ソフトウェア工学 97−14
（1994．3．1）

概念データモデルで記述された仕様の検証

横田 和久　小林 吉純
ATR 通信システム研究所

ソフトウェアの生産性と信頼性を向上させるためには，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に，ERモデルに基づき計算を記述するために従属性制約を用いた，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，そのコンパイラを提案した．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，誤りの発見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，我々は仕様の誤りの検出能力のある関数型言語に着目し，PSDLを関数型言語へ変換して検証する方法を開発した．その手法について報告する．

Validation of Specification Described with Conceptual Data Model

Kazuhisa YOKOTA　Yoshizumi KOBAYASHI

ATR Communication Systems Research Laboratories

2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan

The reusability of program specification is important to improve software productivity and software reliability. The authors have already proposed the declarative language PSDL based on the conceptual data model and the dependency constraints, and we made the compiler of PSDL. We implemented the supporting system called the SoftReuse system. The experiment using it shows the improvement of software productivity. In the program development, this environment avoid including the faults in programs because the programs are generated automatically. But it is not easy to detect the specification faults if there are faults in the specification. We need the way to find the faults of the specification. So we paid attention to functional language that had the ability to find faults. Then we propose the validation method of PSDL using functional language. The method is reported in this paper.

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[2].

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，誤りの発見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，我々は仕様の誤りの検出能力のある関数型言語に着目し，PSDLを関数型言語へ変換して検証する方法を開発した．その手法について報告する．

2 PSDL

PSDLではプログラムの入出力の性質に着目して，プログラム仕様を以下の3階層に分けて記述する．

1. プログラムの入出力データに表される対象世界の情報について，その枠組を定めた情報層．
2. 帳票のような入出力データ形式を定めたデータ層．
3. 入出力ファイルのアクセス方法を定めたアクセス層．

本稿の説明に用いられるプログラム仕様を記述した図1と，その仕様を図示した図2を用いてPSDL文を概説する．このプログラムはファイルproduct_fileとsale_fileを入力して，売上毎に計算した売上額をファイルaccount_fileへ出力する．その時，顧客毎に計算した売上合計も併記して出力するものとする．

2.1 情報層

情報層は，図1では1行目のINFORMATION文と30行目のDATA文の間に記述し，図2では上半分の点線の四角内に図示している．

2.1.1 実体型，属性，主キー，実体数

入力データは図2のTelevisionや，Sale No.1, Mr. Tanakaというような実体を表している．これらの実体の集合である実体型productや，sale, customerは太線の四角形で図示する．実体型の主キー属性と非主キー属性はその四角形の下に示している．

図1では各々の実体型を2行目のE(Entity type)文で記述する．それに続けて属性を4行目のA(Attribute)文で記述し，5行目のK(Key)で直前の属性が主キー属性であることを示す．4行目のSTR(STRing)は属性値の定義域が文字列であることを示し，6行目のNUM(NUMber)は数値であることを示す．実体型に含まれる実体数は3行目のEN(Entity Number)文で記述する．3行目の"-50"は実体型productに含まれる実体が最大50個であることを示している．

2.1.2 関連型，実体型の対応づけ，関連数

入出力データは図2の点線の直線で示した"Television are *sold* in sale No. 1."や"Mr. Tanaka *buys* in sale No. 1."というような関連も表している．これらの関連の集合である関連型soldやbuyは太線の菱形で図示している．

図1では各々の関連型を7行目のR(Relationship type)文で記述する．それに続けて，関連型で対応づけられた実体型を8行目と10行目のC(Collection)文で指定する．この文には実体型とその役割をRole.Entity typeの形で指定する．それらの実体型が相互に異なる場合は図1のように役割を省略してもよい．実体型を指定したC文の後には，その実体型の1つの実体につながる関連の個数を9行目のRN(Relationship Number)文で記述する．9行目のM(Many)は個数が不定であることを示し，11行目の1は1個であることを示している．

2.1.3 属性値従属性制約

この制約は非主キー属性の値を得るために用いる．図2の矢印で示すように実体saleの属性amountの値は，その実体へ関連soldで対応づけられた実体productについて，その属性priceの値とsaleの属性quantityの値を掛けて得られる．また，実体customerの属性totalの値は，その実体へ関連buyで対応づけられた零個以上の実体saleについて，その属性amountの値を合計して得られる．

図1では，値が得られる非主キー属性のA文に続けて，18行目の=(equal)文で制約を記述する．この制約から参照される属性はattributeまたはrole1.relationship Type.role2.entity Type.attributeの形で記述する．ここで前者は，値の得られる実体が持っている他の属性を参照するのに用い，一方その実体へ関連で対応づけられた他の実体の属性を参照するのに後者を用いる．なお，前者のみが現れる制約も記述してよい．また，後者を用いる場合，1つの制約に1つの関連型しか記述できない．ところで，上記のrole1は値が得られる方の実体の役割を示し，role2は他方の実体の役割を示している．

2.1.4 関連存在従属性制約

この制約は関連を得るのに用いる．この例は図1にないので以下に示す．たとえば，実体personが持っている属性skillの値が，実体sectionに適した属性skillの値に等しければ，その場合のみ両実体が関連belong-toで対応づけられるものとする．この関連存在条件はR文とC文に続けて，RC(Relationship existence Condition)

```
 1 INFORMATION
 2   E product
 3       EN -50
 4     A name  STR
 5      K
 6     A price NUM
 7   R sold
 8     C .product
 9       RN M
10     C .sale
11       RN 1
12   E sale
13       EN -100
14     A number   NUM
15      K
16     A quantity NUM
17     A amount   NUM
18       = .sold..product.price * quantity
19   R buy
20     C .customer
21       RN M
22     C .sale
23       RN 1
24   E customer
25       EN -50
26     A name  STR
27      K
28     A total NUM
29       = ASUM(.buy..sale.amount)
30 DATA
31   I product_data
32         IX product_id
33       G product_record ON EndOfFile(product_data)
34     O product_record
35       %12s product_name
36         = product.name
37       %8d  product_price
38         = product.price
39   I sale_data
40         IX sale_id
41       G sale_record ON EndOfFile(sale_data)
42     O sale_record
43       %4d  sale_number
44         = sold..sale.number
45         = buy..sale.number
46       %16s sale_customer
47         = buy..customer.name
48       %12s sale_product
49         = sold..product.name
50       %4d  sale_quantity
51         = sale.quantity
52   I account_data
53         IX account_id
54       G account_record ON PEntityNumber(sale)
55     O account_record
56       %4d  sale_number
57         = sale.number
58         = buy..sale.number
59       %8d sale_amount
60         = sale.amount
61       %16s sale_customer
62         = buy..customer.name
63       %10d customer_total
64         = buy..customer.total
65 ACCESS
66   D product_file INPUT  20 product_data
67   D sale_file    INPUT  36 sale_data
68   D account_file OUTPUT 38 account_data
```

図 1: PSDL プログラム仕様 (テキスト表現)

文で "RC (.person.skill == .section.skill)" と記述する．ここで，条件式から参照される属性は role.entity Type.attribute の形で記述している．

2.1.5 実体存在従属性制約

この制約は実体を得るのに用いる．この例も図 1にはないので以下に示す．たとえば，実体 customer の属性 total が正値であれば，その場合にのみ，その実体に関連 demand で対応づけられる実体 account が存在するものとする．この時，得られた実体の主キー属性の K に続けて，主キー属性値を決めるための計算式を属性値従属性制約と同じ = 文で "= .demand..customer.name ON (.demand..customer.total > 0)" と記述する．この例では account の主キー属性値は customer の主キー属性 name の値に等しいものとした．また，total の値が正か否かを判定するための条件式は ON 句で記述する．なお，1 つの制約に 1 つの関連型しか記述できない．

2.2 データ層とアクセス層

2.2.1 データ層

データ層は，図 1では 30 行目の DATA 文と 65 行目の ACCESS 文の間に記述し，図 2では中段の点線の四角形の中に示している．入出力データの構造は基本データ型や，連接集団データ型，繰り返し集団データ型，選択集団データ型で階層的に定める．これらのデータ型は各々 % 文や，Q(seQuence) 文，I(Iteration) 文，S(Selection) 文で記述する．

% 文は基本データ型のデータ形式も C 言語と同様に定める．たとえば，%12s は 12 文字の文字列を定めている．I 文には続けて指標も IX 文で記述する．上記の集団データ型は他のデータ型から構成されるが，構成要素のデータ型の中に集団データ型があれば，それを G(Group) 文で指定する．繰り返し集団データ型には繰り返し終了条件が必要であり，33 行目の ON 句は End Of File 条件を示し，54 行目の ON 句は実体 sale の個数だけ繰り返すことを示している．

ところで，本稿では入出力データは実体や，その属性値，関連を表すものと見なしているので，まず実体については実体型の主キー属性を図 1の 36 行目の = 文で基本データ型と結合する．属性値については属性を 38 行目の = 文で基本データ型と結合する．関連については，関連で対応づけられた実体型の主キー属性を 44 行目と 49 行目の = 文で基本データ型と結合する．なお，実体のないところには関連もないので，入出力データには関連とともに実体も表される．上記の結合を取ることは情報層とデータ層の結合制約と呼ぶ．

2.2.2 アクセス層

図 1の 65 行目の ACCESS 文に続けて各々のファイルをデータセット型として 66 行目の D(Dataset Type) 文で記述する．この文にはファイル名や，INPUT と OUTPUT

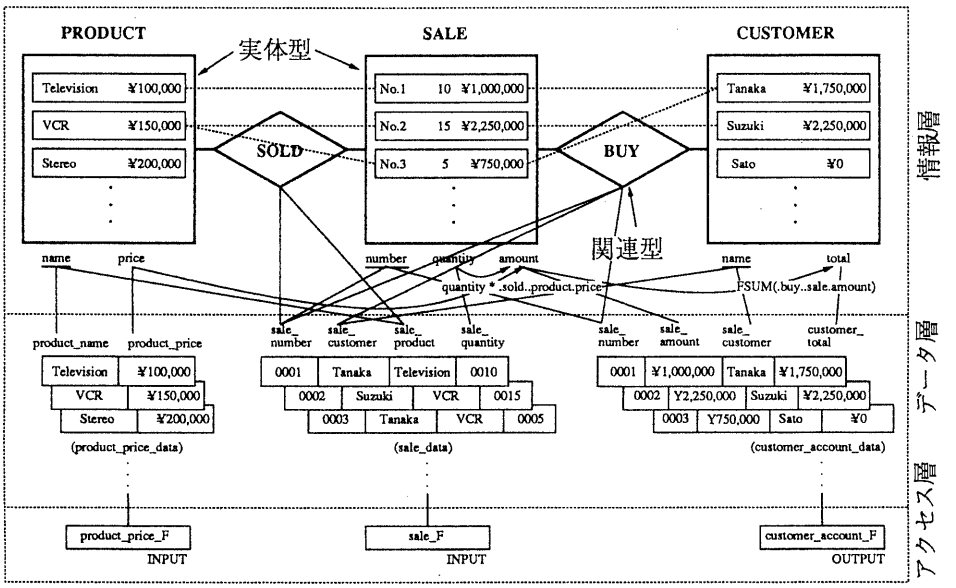

図 2: PSDL プログラム仕様 (図的表現)

の区別，レコード長を記述する．データ層とアクセス層の結合制約として product_data のようにデータ型を指定する．

3 PSDL から関数型言語への変換

PSDL から関数型言語への変換は以下の条件で行なう．

1. 変換対象は情報層のみ
2. 入力データはすでに関連型や実体型を表す変数に入力されていると仮定する
3. 出力データは出力対象となる変数へ代入されることで出力されたとみなす

変換対象を情報層に限るのは，アクセス層やデータ層はファイルやレコードの構造を定義するだけで計算を含まないので，論理的なエラーは生じないためである．以下の関数型言語の表記は「関数プログラミング」[3] に従う．

3.1 データ構造の変換

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
   実体は複数の属性値が集まったものであるので，このまとまりを関数型言語のデータ構造の組で表す．
   実体型は実体の集合であるので，実体の組をリストにして表現する．すなわち実体型は組を要素として持つリストになる．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 2 つ組で表す．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

3.2 制約の変換

PSDL の制約には，属性値従属性制約，関連存在従属性制約，実体存在従属性制約の 3 種類がある．これらの制約は，リストを引数としてリストを返す関数へ変換する．

3.2.1 補助関数

リストをPSDLの実体型や関連型とみなして操作を行なうために，いくつかの補助関数を用意する．

1. リストの先頭を取り出す関数

$$hd\ ([x] +\!+ xs) \quad = \quad x$$

2. リストの先頭を取り除いた残りのリストを返す関数

$$tl\ ([x] +\!+ xs) \quad = \quad xs$$

3. 実体の各属性値を取り出す関数

$$\begin{aligned} attr1\ (a,b,c) &= a \\ attr2\ (a,b,c) &= b \\ attr3\ (a,b,c) &= c \\ &\vdots \end{aligned}$$

関数$attr$nは，引数の型が違うと適用できないが(2つ組用の関数は3つ組には適用できない)，以下の例では引数の型に対応した関数を$attr$nで代表させている．

4. 関連の各主キー属性値を取り出す関数

$$\begin{aligned} fst(a,b) &= a \\ snd(a,b) &= b \end{aligned}$$

5. 主キー属性値k1を与えたときに，関連型Rを介してつながっている実体型E2の実体e2を返す関数

$$\begin{aligned} e2 &= lookup\ k1\ E2\ R \\ lookup\ k1\ [\,]\ R &= [\,] \\ lookup\ k1\ (x:xs)\ R &= x, \textbf{ if } fst\ x = k2 \\ &= lookup\ k1\ xs\ R, \textbf{ otherwise} \\ &\quad \textbf{where } relation\ k1\ R. \end{aligned}$$

ここで関数relationは，主キー属性k1と関連付けられたもう一方の実体型の主キー属性を関連型Rから見つけて返す関数である．定義は以下の通り．

$$\begin{aligned} relation\ k1\ R &= k2 \\ relation\ k1\ [\,] &= [\,] \\ relation\ k1\ (x:xs) &= snd\ x, \textbf{ if } fst\ x = k1 \\ &= relation\ k1\ xs, \\ &\quad \textbf{otherwise.} \end{aligned}$$

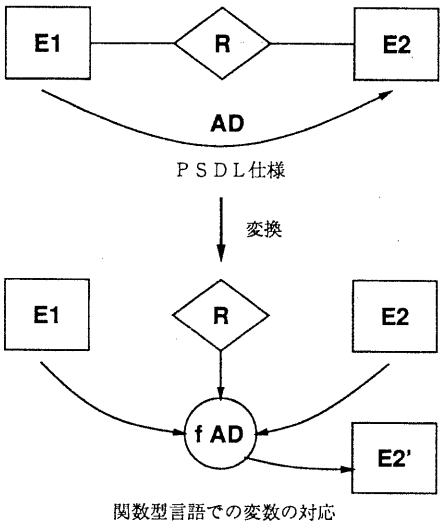

図 3: ADの変換

3.2.2 属性値従属性制約(AD)

ADは，実体型の属性値を計算する制約である．そこで，以下のようなアルゴリズムを実行する関数型言語の関数へ変換する．

実体型E1,E2と関連型RからADの計算を行なう．関数の形は以下のようになる．

$$E1' \quad = \quad f_{AD}\ E1\ E2\ R$$

このf_{AD}の計算内容は以下のようになる．

for e1 in (実体型E1の各要素について)
begin

- e1の主キー属性値k1をもつ関連型Rの要素rを探す
- rから実体型E2の主キー属性値k2を取り出す
- k2を主キー属性値とする実体e2を実体型E2から探す
- 計算に必要なe1,e2がそろったので計算を行なう
- e1の属性値を計算によって得た属性値で置き換えたe1'を作る

end
e1'を集めてリストにする．

上記のアルゴリズムの関数型言語での表現は以下のようになる．

$$
\begin{aligned}
E1' &= f_{AD}\ E1\ E2\ R \\
f_{AD}\ [\,]\ E2\ R &= [\,] \\
f_{AD}\ E1\ E2\ R &= f_{ADC}\ e1\ lookup\ fst\ e1\ E2\ R\ : \\
&\quad\ f_{AD}\ tl\ E1\ E2\ R \\
&\quad\ \textbf{where}\ e1 = hd\ E1.
\end{aligned}
$$

f_{ADC} は実体 e1,e2 を引数にして AD を計算する関数で，属性値の求まった実体 e1' を関数の値として返す．f_{ADC} は，PSDL 中の AD と実体 e1,e2 の構造によって形が決まる．例えば，

$$
sale.amount = .sold..product.price * quantity
$$

では

$$
\begin{aligned}
product &= (name, price) \\
sale &= (No, quantity, amount) \\
sold &= (name, No)
\end{aligned}
$$

とすると，

$$
\begin{aligned}
sale' &= f_{ADC}\ sale\ product \\
f_{ADC}\ sale\ product &= (attr1\ sale, attr2\ sale, \\
&\quad\ attr2\ sale * attr2'\ product).
\end{aligned}
$$

となる．ここで関数 f_{AD} はリストを処理するために再帰呼び出しをしているが，再帰呼び出しの回数はリストの長さに等しいので，リストの長さが有限であれば f_{AD} は必ず停止する．PSDL の AD と関数型言語の対応を図 3に示す．

3.2.3 関連存在従属性制約 (RD)

RD は，生成される関連型が結び付ける 2 つの実体型 E1,E2 を引数にとり，関連型 R を生成します．

$$
R = f_{RD}\ E1\ E2
$$

この f_{RD} の計算内容は以下のようになる．

for (e1,e2) **in** (E1,E2 の直積)
begin
　　if e1,e2 が関連存在条件を満たす
　　then
　　　　R に e1,e2 の主キー属性の組を追加
end

RD はリストの内包表記を使って以下のように書ける．

$$
\begin{aligned}
f_{RD}\ E1\ E2 &= [(fst\ e1, fst\ e2) | e1 \leftarrow E1; e2 \leftarrow E2; \\
&\quad\ f_{RDC}\ e1\ e2].
\end{aligned}
$$

ここで，f_{RDC} は関連存在条件を検査する論理式を表す関数である．f_{RD} は，E1,E2 が有限であれば停止する．PSDL の RD と関数型言語の対応を図 4に示す．

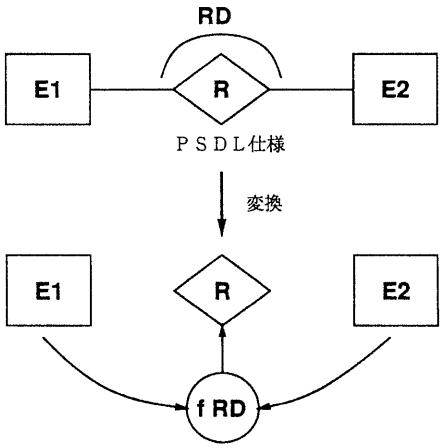

図 4: RD の変換

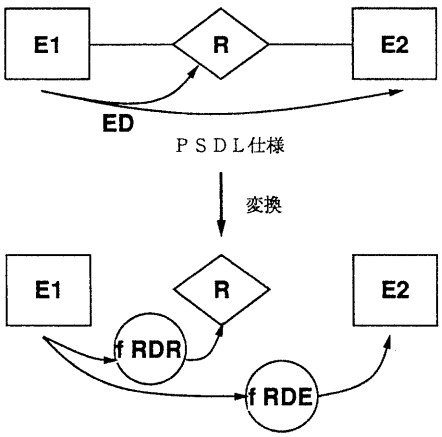

図 5: ED の変換

3.2.4 実体存在従属性制約 (ED)

EDは，実体型と関連型の両方を生成するので二つの関数に変換する．計算に必要な引数は両方の関数に共通である．生成される実体型をE2，生成される関連型をR，もとになる実体型をE1とする．E2を生成する関数をf_{EDE}，Rを生成する関数をf_{EDR}とすると引数と関数の値の関係は次のようになる．

$$\begin{aligned} E2 &= f_{EDE}\ E1 \\ R &= f_{EDR}\ E1 \end{aligned}$$

f_{EDE}は，実体型E1からEDの生成条件に従って実体型E2を生成する関数で具体的な形は次のようになる．

$$\begin{aligned} f_{EDE}\ [\,] &= [\,] \\ f_{EDE}\ E1 &= f_{EDK}\ hd\ E1\ :\ f_{EDE}\ tl\ E1, \\ &\quad \textbf{if}\ f_{EDC}\ hd\ E1 \\ &= f_{EDE}\ tl\ E1,\ \textbf{otherwise}. \end{aligned}$$

f_{EDR}は，実体型E1からEDの生成条件に従って関連型Rを生成する関数で具体的な形は次のようになる．

$$\begin{aligned} f_{EDR}\ [\,] &= [\,] \\ f_{EDR}\ E1 &= (fst\ f_{EDK}\ hd\ E1, fst\ E1) : \\ &\quad f_{EDR}\ tl\ E1,\ \textbf{if}\ f_{EDC}\ hd\ E1 \\ &= f_{EDR}\ tl\ E1,\ \textbf{otherwise}. \end{aligned}$$

ここで，f_{EDE}, f_{EDR}ともに再帰呼び出しをしているが，再気呼び出しの回数はリストE1の長さに等しいので，E1が有限であればf_{EDE}, f_{EDR}は停止する．PSDLのEDと関数型言語の対応を図5に示す．

3.3 関数の組み合わせ方

上記の方法により生成された各関数は，まだ相互に関連付けられていない．関数型言語では単一代入の制限により，変数への代入は一度しかできない．そのためある実体型を生成する関数が複数ある場合は，同じ実体を表す変数が複数できる．この複数ある変数と，関数の入力に現れる変数をどう対応付けるかが問題になる．ここでは，以下の方法により対応付けを行なう．

1. 実体型を表す各変数について，属性ごとににフラグを作る．
2. 入力側の変数については，値が確定している必要がある属性に相当するフラグを立てる．
3. 出力側の変数については，入力側の変数の属性も考慮して値が確定している属性のフラグを立てる．
4. 同じ実体型を表す入力側の変数と出力側の変数を対応付けるが，このとき入力側のフラグが立っている属性と同じ属性のフラグが立っている出力側の変数を対応付ける．出力側の変数の他の属性のフラグはどちらでも良い．

上記の方法で対応付けを行なった場合，対応付けが複数可能なことがある．現段階では，対応付けが複数可能になるということは元のPSDLで書かれた仕様に曖昧さが含まれているということで，仕様に不備があるということにしている．また対応付けが不可能な場合も，仕様に何らかの誤りがあると考えられる．

図1を例に説明する．図1には2つのADが含まれるので，関数型言語へ変換すると2つ関数ができる．それぞれのADをAD1,AD2として属性値の依存関係を含めて変数の関係を図示すると図6のようになる．属性を囲む楕円はその属性のフラグが立っていることを表す．

ここでは実体型saleを表す変数がsale,sale',sale2の3つある．saleは入力側の変数であるがあらかじめデータが入力されている変数なので，出力側の変数と対応付ける必要はない．sale2が対応のついてない入力側の変数である．sale2は，属性Noとamountのフラグが立っている変数と対応付ける必要がある．そこで，saleとsale'を調べると属性Noとamountのフラグが立っているのはsale'なので，sale2はsale'が対応付けられるので同じ変数を割り当てる．

4 PSDLの停止性の判定

前節の変換方法をPSDL仕様に適用することにより，PSDL仕様を関数型言語へ変換することができる．このとき各制約(AD,RD,ED)を変換してできる関数は，入力となるリストの長さが有限であれば停止することが保証できる．またPSDL言語の性質より入力は必ず有限である．このことより，停止性を保証するにはある関数の出力が入力へ戻ってこないことを確認すれば良い．これにより，仕様の停止性の判定は，以下の手順により行なうことができる．

1. 前節の方法により，PSDLを関数型言語へ変換する．
2. 関数，実体型，関連型を節点，関数の引数の依存関係を有向枝とした有向グラフを作る．
3. 2の有向グラフを解析し，閉路を見つける．閉路がなければ仕様は停止する．

図6の例を有向グラフへ変換すると図7になる．このグラフは閉路を含まないため，図1のPSDLの仕様は停止する．

5 まとめ

非手続き的な言語であるPSDLプログラム仕様の停止性を，以下の条件の元で保証する方法を提案した．

1. PSDLから関数型言語への変換において，関数の組み合わせ方が一意に定まる．
2. 関数型言語での表現において，変数の依存関係に閉路が生じない．

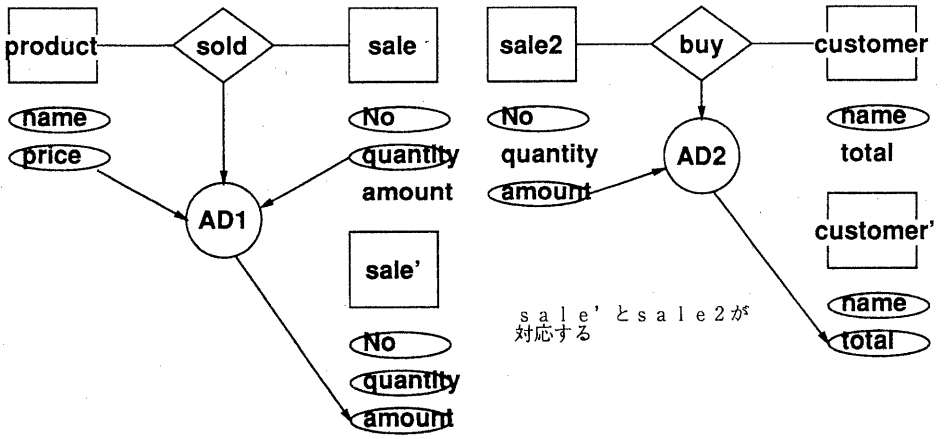

図 6: 関数の組み合わせ方

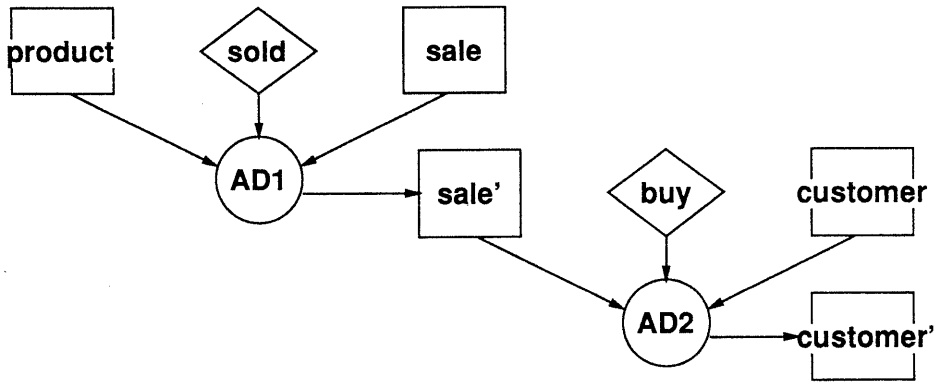

図 7: 有向グラフ

今回の報告では，非常に限定された範囲でしか停止性の保証ができていない．特に前節の手順で，有向グラフに閉路が含まれる場合についても条件を設けることによって停止性を保証できる場合があると考えている．今後は，停止性を保証できる範囲を広げると共に，関数型言語上での仕様の実行や，停止性以外の検証についても研究を進めるつもりである．

謝辞 — 日頃ご指導いただく葉原会長，寺島社長，太田室長に深く感謝いたします．また，ご討論いただいた研究室の諸氏に感謝いたします．

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. of IEEE COMPSAC'90,* pp. 629-638, 1990.

[2] 横田, 橋本, 佐藤, 概念データモデルと従属性制約を用いた言語による仕様再利用の実験, 情報処理学会研究報告 92-SE-89, pp. 169-176, 1992.

[3] R. バード, P. ワドラー, 関数プログラミング, 近代科学社, , 1991.