記 号 処 理 58－1
（1991 1．11）

C言語の問題向き拡張システム:OPTEC

小島 泰三　杉本 明　阿部 茂
三菱電機株式会社 中央研究所

OPTEC システムは，問題向きに拡張されたC言語表現を通常のC言語表現に変換するトランスレータ構築を支援するツールである．本システムでは，データ型を導入したパタン変換技法を用いることにより，C言語のシンタクスとセマンティクスの拡張をパタン変換の単一枠組において実現している．そして，このパタン変換を用いて，C言語拡張を容易に実現できることを特長としている．本稿では，OPTEC システムの概要を説明する．そして，パタン変換へのデータ型の導入について述べ，データ型を用いたパタン変換の利点を示す．また本稿では，データ型を導入したパタン変換を高速実行可能な，型推定を伴うパタンマッチング処理についても述べる．

Overloading in Pattern Transformation for Extending C language

Taizo Kojima, Akira Sugimoto and Shigeru Abe
Central Research Laboratory, Mitsubishi Electric Corporation
8-1-1 Tsukaguchi-Honmachi, Amagasaki, Hyogo, 661, JAPAN

The OPTEC system is a tool for introducing application-specific terminology into C language. This system provides the ability to extend both syntax and semantics of the language. The novel feature of the OPTEC system is that it regards data type as one of the elements that compose the pattern. In designing the OPTEC system, we combine the pattern transformation method and the overloading mechanism. Overloading mechanism in the OPTEC system is built into the pattern transformation by introducing data type into the transformation algorithm. Thus, both extension of syntax and semantics can be handled within a single framework of the pattern transformation.

1 はじめに

従来のプログラム開発では，プログラム言語としてADA,PL/1などの汎用言語，あるいはCOBOLなどの問題向きの言語が用いられている．汎用プログラム言語を用いることの利点は言語の柔軟性にあり，様々な問題に対応できることである．一方，問題向き言語を用いる場合，各問題向きの機能及び構文を使用することにより可読性が増し，保守が容易になるという利点がある．このため両者の利点を享受するため，既存の汎用プログラム言語をベースとし，これを拡張する形で問題向きの機能を付加するものがあらわれた[1].

筆者らはC言語[2]の問題向き拡張システムOPT-ECの開発を行なった．本システムの目的は，C言語を問題向き拡張することにより，大規模応用プログラム開発におけるアプリケーションジェネレータとして利用することである．本システムは，問題向きに拡張されたC言語を用いた記述を標準のC言語による表現に変換する方式を採用した．広く普及しているC言語を問題向きに拡張することの利点は，(i) C言語は各種問題向き機能を実現するために必要なプリミティブ操作を備えている，(ii) マシン依存の情報が不要となりシステム構築が容易，また可搬性に優れる，ことである．

本システムの構築にあたり，筆者らはシンタクス及びセマンティクスの拡張がプログラム言語の問題向き拡張には重要であると考えた．以下に理由を示す．

- シンタクスの拡張：C言語は一般的なプログラミングに対するシンタクスは有しているが，特定の問題に特殊化したものは備えていない．例えば，並列処理プログラムを記述するには同期操作などを簡潔に表現できることが望まれる．ところが通常のC言語では並列処理に対する言語機能は備わっていないため，並列処理を実現するためには複雑なプログラミングが必要である．この場合，プログラムを明確且つ簡潔とするためには，並列処理に適したシンタクスの導入が必要となる．さらにまた，要求されるシンタクスは問題領域によって様々である．

- セマンティクスの拡張：既存のシンタクスの範囲で対象の問題が自然に記述できるならば，新たなシンタクスを導入せず，既存のシンタクスで対象を扱えることが望ましい．例えば演算子＋は整数，実数だけでなく，複素数あるいは行列などに対しても使用できることが望まれる．ところが通常のC言語では整数，実数に対する操作のみ定義されており，復素数などに対しては演算子＋の利用は許されていない．この場合，整数，実数以外に対しても利用できるよう，演算子＋のセマンティクスを拡張できることが必要である．

  あるいはまた，データ並びに対して繰り返し処理を記述する場合，記述にはデータ並びの実現(配列あるいはリスト)が出現せず，同一構文により様々なデータに対して適用できるほうがプログラムは明確且つ簡潔となる．

従来からプログラム言語の問題向き拡張には，さまざまな手法がとられている．例えば[1]では，Concurrent CからC言語へのトランスレータをYACCを用いて構築している．YACCなどのコンパイラ構築ツールを用いて実装する場合，デバッグ支援や最適化など様々な機能を言語システムに実現することが可能である．しかしながらYACCでは，シンタクスを実現するには文脈自由文法による完全な定義を与える必要があり，またセマンティクスはC言語を用いて複雑なプログラミングを行なうことによって実現する必要がある．このためシンタクス及びセマンティクスの拡張を実現するのは容易ではない．さらにまた，新たな拡張が必要となる度に処理系の再構築が必要である．

あるいはまた，パタン変換を用いた言語拡張システムにより言語拡張の容易化を狙ったシステムの開発も行なわれている．TXL[3, 4]はパタン変換を用いて，拡張した言語による記述をベースとなる言語表現に変換するシステムである．TXLでは専用の記述言語を用いて，言語の拡張について記述する．このときシンタクス定義では追加拡張した部分についてのみ定義する．一方，セマンティクス定義では構文パタンからベースとなる言語表現を生成するための書き換えルールを与える．変換システムは，入力されるプログラムソースに対してパタンマッチングによりセマンティクス定義で指定されたパタンを検出し，書き換えルールを実行することによりパタンの書き換えをする．従ってYACCを用いて言語拡張を実現する場合と比較し，シンタクスの拡張は容易である．しかしながらTXLのような構文形式に対するパタン変換だけでは，一つのシンタクスには一つの変換しか定義することができない．このためセマンティクスの拡張は実現されていない．

一方，オブジェクト指向の研究では，オーバーローディングによりセマンティクスを拡張する試みが行なわれている．オーバーローディングは，対象とするデータの型により操作が決定する機構である．C++[5]では，引数の型により呼び出される関数を選択するオーバーロード関数や，オペランドの型により演算子

の意味を決定するオペレータオーバロードなどを導入している．これにより，複素数やストリーム入出力などに対する問題向き表現をC言語に導入することに成功した．オーバローディングの利点は，同一形式の構文であっても，その中で出現する式の持つデータ型により表現の意味を決定できることである．

OPTEC システムの実現では，パタン変換によるシンタクスの拡張とオーバローディングによるセマンティクスの拡張に注目した．そして，データ型を構文パタンを構成する要素の一つとすることにより，オーバローディングをパタン変換として一般化した．本システムでは TXL と同様，拡張部分のみについてのみシンタクス定義を与え，セマンティクスはルールにより定義する方式を採用した．しかしながらルール選択機構は従来のパタン変換システムとは異なり，構文パタンのみでなく，そのデータ型も用いる．またシステム構成としては，(1)エンドプログラマにはトランスレータのみを提供することにより，問題向き拡張を集中的に管理できる．また，(2)問題向き拡張定義などの再利用性の高い部分の処理を前処理として分離し，エンドプログラマ使用時のオーバヘッドを低減できることから，トランスレータ生成による方式を採用した．

従来からトランスレータ構築に関する研究は行なわれている [?, 9]．これに対し本システムでは，単一のパタン変換の枠組でプログラム言語のシンタクスとセマンティクスの拡張を実現し，これをシステム自身及び生成される問題向き言語トランスレータの中核にしたことを特長としている．

以下，第2章では OPTEC システムの構成を示し，第3章で，データ型を導入したパタン変換によるオーバローディングの実現と，パタン変換の利点について示す．そして第4章では，パタン変換を高速に処理することを目指した，型推定を伴うボトムアップなパタンマッチング方法を説明する．そして第5章では簡単なC言語拡張例を示し，最後にまとめを行なう．

2 システム構成

OPTEC システムは，問題向きに拡張されたC言語表現から通常のC言語表現へのパタン変換を行なう変換システムの生成系と，変換実行をサポートするランタイムライブラリから構成されるC言語拡張システムである．図 1に OPTEC システムの構成を示す．本システムにおける処理は，以下の3つのステップから構成される．

1. パーサジェネレータ (図 1(a)) を用いて構文解析部の生成を行なう．このときルールコンパイラ (図 1(b)) 用及び変換システム (図 1(c)) 用の2種類を生成する．このステップでは，ベースであるC言語のシンタクス定義 (YACC プログラムのテンプレート) と追加拡張するシンタクス定義を用い，YACC ソースファイルを生成する．

2. 変換システムを生成する．まず，第1ステップで生成した構文解析部を組み込んだルールコンパイラを用いて，変換ルール定義をC言語プログラムに変換する．また，高速なパタンマッチングを可能とするため，ルールで指定されるパタンも予め処理する．次に，変換システム用の構文解析部と本ステップで生成したC言語プログラムをコンパイルリンクし，変換システムを生成する．

3. 第2ステップで生成した変換システムを用いて，問題向きに拡張されたC言語ソースから通常のC言語表現への変換を行なう．この処理では，問題向きに拡張された記述の構文ツリーと，変換ルールの構文ツリーとの間でパタンマッチングを行い適用可能ルールを求める．ルールが検出されるとマッチングしたツリーに対してルールを適用し，ツリー書き換えを行なう．なおマッチングするルールがない場合は，無変換で出力する．

本システムの実用化では，多数のプログラマによる大規模システムの開発支援を目指した．このため，(1)エンドプログラマには変換システムのみを提供し，問題向き拡張を集中的に管理できる，(2)変換ルールの読み込みやパタンマッチングの準備を前処理として分離でき変換処理時の高速化が可能であることから，上記のように変換システム生成系と変換実行を分離して構成した．なおルールコンパイラ自身，本システムを用いた実装となっており，ルール記述において拡張したC言語による記述が行なえる．ルール例は後述する．

3 パタン変換へのデータ型の導入

本システムでは，データ型をパタン変換に導入することにより，型によるオーバローディング処理を実現している．このため，シンタクスとセマンティクスの拡張を一つの枠組で実現することが可能となった．以下では，データ型のパタン変換への導入について説明し，次にデータ型を導入したパタン変換の利点について示す．

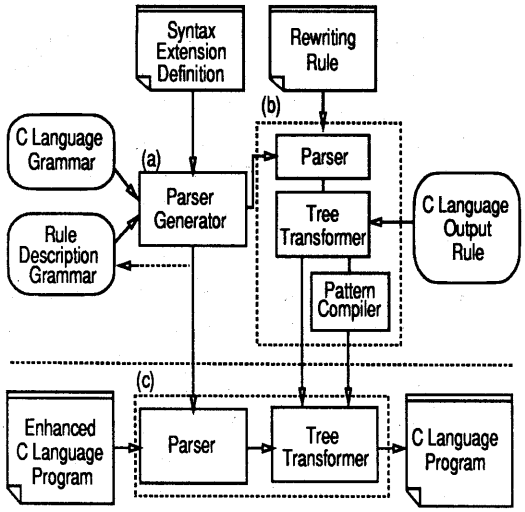

図 1: Organization of OPTEC System

3.1 パタン変換によるオーバローディング

本システムではオーバローディングをパタン変換の枠組で実現している．その仕組みについて，2項演算子 @ におけるオーバローディングを例として説明する．

```
String a,b; Vector A,B;
a @ b   /* 式(1) */
A @ B   /* 式(2) */
```

ここで演算子 @ についてオーバローディングを行ない，オペランドの型がString 型の場合と Vector 型の場合では異なる処理を実現するものとする．ところが従来の構文ツリー表現では，式(1),(2)は同じパタンを持つツリー表現となる．(図 2(a)) このため OPTEC では，データ型をパタンに導入した．図 2(b),(c) に OPTEC による式(1),(2) のツリー表現を示す．本システムでは，式ノードにデータ型を付加しており，図 2 のように，データ型はパタンを構成する要素の一つとして扱われる．

3.2 パタン変換の利点

データ型を導入したパタン変換は，C++ など演算子に対してオーバロード定義を行なう従来システムと比較して，より多くの情報を利用できるという利点を持っている．

従来のシステムは演算子に対してオーバローディングを指定するため，局所的な情報のみで処理する必要があった．例えば2項演算子の場合は，2つの引数の型のみでオーバローディング処理が行なわれる．これに対してデータ型を導入したパタン変換で

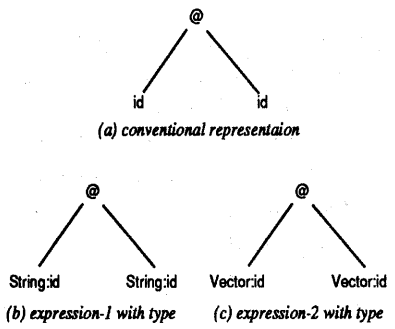

図 2: Pattern for Binary Operator

は，ある特定のパタンに対して処理を定義する．用いるパタンには複数の演算子が用いられても良く，また特定の式の包含関係を指定することもできるため，より広域な情報を利用することができる．この点について，代入演算を例として説明する．

以下の例では，演算子 '=' の左右で式が異なる意味を持つ場合について述べる．

```
String x; int y;
y = x @ 2; /* 式(1) */
x @ 2 = y; /* 式(2) */
```

一般にC言語において，代入演算の左式は値が格納される場所，右式は値を示す．上記例において，'x @ 2' は，式(1) では評価結果の値を，式(2) では右辺の値を格納すべき場所を示している．従って，演算子 '@' に関して，それを用いた式の出現する場所に応じて，例えば以下のように，異なる操作が要求される．

```
y = get(x,2); /* 式(1)' */
set(x,2,y);   /* 式(2)' */
```

しかしながら従来のオペレータオーバローディングの場合，該当する式パタンが現れたのが代入演算の左辺あるいは右辺であるかの区別ができない．このため，以下のように単一の変換しか扱うことができない．

```
y = at(x,2);   /* 式(1)" */
at(x,2) = y;   /* 式(2)" */
```

これに対して，データ型を導入したパタン変換の場合，図 3 に示すような複数の演算子を含むパタンを用意することにより，この問題を解決することができる．本システムにおけるパタンマッチングでは，より上位のノードでマッチングするものが優先する．式(1),(2) について図3の (a),(b) のパタンを適用すると，式(1) ではパタン(a) が選択される．一方，式(2) では パタン(a),(b) ともにマッチングするが，@ より上位の = でパタン(b) がマッチングするので，パタン(b) が選択される．なお図中，

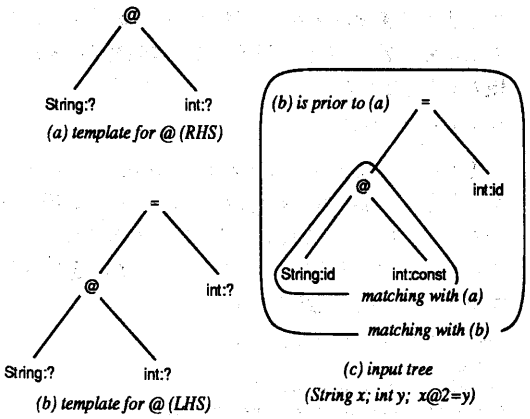

図 3: Pattern for LHS and RHS

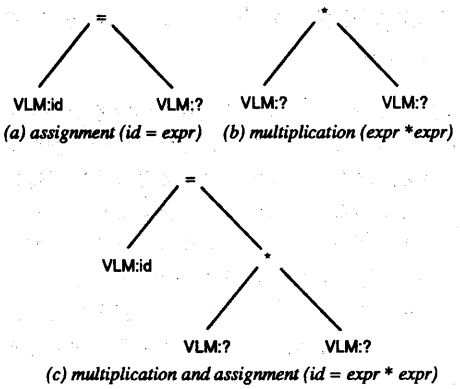

図 4: Template Pattern for Matrix Operation

'?' は任意の式種別をマッチングするワイルドカードである.

次に特殊なパタンを用意することにより, 従来のシステムと比較して, 実行効率の良い変換結果を得る例を示す. 以下のような行列に対する式を考える. なおここで, 型 VLM (VeryLargeMatrix) は行列を表すデータ型でとする.

```
VLM a,b,c;
a = b * c;
```

行列の乗算を扱う数値演算ライブラリは, 通常2つの入力行列と結果を格納する行列を引数として呼ばれる. 演算子に対してオーバーローディングを指定する従来の方式の場合 (この場合 演算子=, *), 図4 の (a),(b) の2つのパタンを用いて変換することと等しい. このパタンによる変換を概念的に示すと以下のような操作となる.

```
VLM tmp;
mul(&b,&c,&tmp);
set(&a,&tmp);
```

ところが, このような演算が頻繁に行なわれ且つ行列が非常に大きい場合, 一時領域 tmp の確保及びコピー操作のオーバヘッドは無視できなくなる. これに対してデータ型を導入したパタン変換の場合, 図4(c) に示すパタンを持つルールを追加することにより1つの乗算操作, すなわち

```
mul(&b,&c,&a);
```

で置き換えることが可能となる. このため従来と比較して, 実行効率のより良い変換結果を得ることができるようになる.

4 変換処理

本システムのパタン変換の特徴は, 型推定とパタンマッチングを同時に行なうことである. そして, この手法によりデータ型を導入したパタン変換の高速化を達成している. 以下では, ルール適用と型推定について説明し, 次に型推定を伴うボトムアップなパタンマッチングによるツリー書き換え処理について説明する.

4.1 ツリー書き換えとデータ型

従来からパタンマッチングを用いたツリー書き換えは, コンパイラのコード生成, 数式システムのインタプリタの実現などで用いられている [10, 11, 12].

パタンマッチングによる書き換え処理では, ある書き換えの後の次の書き換え対象の選択が処理速度に影響する [13]. このため書き換えの性格を把握することが重要である. 本システムの場合, データ型をパタン変換に導入したことが書き換え処理の特徴となっている. 例えば, 書き換えにより式の型が決定し, そのため上位レベルでの書き換え処理が発生する場合がある. また, C言語には型検査の機能が備わっているため, もし書き換えによりツリーがC言語基本型間の演算式に変換される場合にはボトムアップな型の付加が必要である. 一方, 同じパタンであっても位置により異なる書き換えが要求されることがある. 例えば代入演算子の左右では式の持つ意味は異なり, ルール適用はトップダウンで行なう必要がある. 以上のように, 本システムの書き換え処理では, トップダウンな処理とボトムアップな処理が同様に要求される. 試作システムを用いた実験では, パタンマッチングと型の決定の組み合わせ方が変換速度に大きく影響を及ぼすという結果が得られた [16, 17]. このため, 次に述べる変換方法を採用した.

4.2 型推定を伴うボトムアップなパタンマッチング

変換方法の特徴は，C言語既存の型検査やルール適用により生じるボトムアップな型の付加を型推定として捉え，パタンマッチング時に型推定を同時に行なうことである．そして本システムでは，一パスのボトムアップなトラバースにより，ツリーに対するすべての適用可能ルールの検出を行ない，次にトップダウンなトラバースによりルールを適用し変換結果を出力する方式を採用した．以下に書き換え方式を示す．

- 変換ルールに変換後の型を定義し，型推定に用いる．C言語組み込みの型の付加も同一の枠組の中で処理する．
- ボトムアップなトラバースによりパタンマッチングと型推定を同時に行なう．あるノードがあるルールのパタンのルートとマッチングする場合には，そのルールを記録し，定義されている型をそのノードに付加する．複数のルールが適用可能な場合，最も特定的なルールを適用可能ルールとして選択する．なおこの決定は，パタンを構成するノード数，型あるいはノード種別の指定，ルール定義順により行なう．
- トップダウンなトラバースによりルール適用を行なう．ルール適用可能なノードに出会った場合，そのノードをルートとするツリーに対してルールを適用し，そのサブツリーに対するトラバースは行なわない．
- ルール実行は出力操作により実現する．またサブツリーに対するルール適用は，出力操作の中から再帰的に呼び出す．

前述のシステム[10, 11]では，トップダウンなパタンマッチングを採用している．また[13]では，パタンマッチングと書き換えによるマッチング再実行に対してはボトムアップな方法が効率が良いものの，必要な前処理や必要記憶域などを考慮するとトップダウンな方法が優れていることを述べている．これに対して本システムでは，(1)変換の前処理と実行は分離し，変換処理の高速実行に重点を置く，また(2)型推定とパタンマッチングを同時に行なえることからボトムアップなパタンマッチングを選択した．

4.3 変換過程

以下では，前述の変換処理の過程を例により説明する．次の記述は，2項演算子 '@' の定義と，型 T に対する変換を示すルール例である．まずシンタクス定義を行なう．これは以下のように演算子の種別，優先度を記述する．

```
$operator (left,13) @;
```

一方ルール記述では，以下のように検出すべきパタン('[..]')，結果の型($type)，及びルール適用時のアクション('{..}')を記述する．以下の例ではアクションとしては 出力操作 emit のみであるが，通常のC言語プログラムを記述できる．

```
$rule r1 [ $(T:p1) @ $(T:p2) ]
 $type T
 { emit("at2(#a,#a)",p1,p2);}
$rule r2 [ $(T:p1) @ $(T:p2) @ $(T:p3) ]
 $type T
 { emit("at3(#a,#a,#a)",p1,p2,p3);}
```

上記記述では，2つのルール r1, r2 を定義している．記述中 $(*type*:*id*) 形式の表現が使用されている．これはデータ型 *type* を持つ任意の式と一致し，識別子 *id* に一致した式が束縛される．

図5に上記ルールによる変換処理を示す．第1ステップはボトムアップなパタンマッチングと型推定である．まずノード n3 において r1 がマッチするので，T型と推定する．次に n2 において r1, r2 両方とマッチングする．r1, r2 を比較すると r2 の方が優先する．そこで r2 を記録し，T型と推定する．最後に n1 においても n2 と同様に r1, r2 とマッチングし，r2 を記録し T型と推定する．以上でパタンマッチング処理は終了する．

第2ステップはトップダウンなルール適用過程である．まず n1 に対してルール r2 を適用し，出力操作 emit 命令を実行する．emit 命令の第一引数は書式制御文字であり，後続の引数の処理を示す．書式制御に '#a' が現れると，該当する引数に対して再帰的にルール適用過程が呼び出される．上記例では，パラメタ p1 には n3 をルートとするツリーが束縛されており，このツリーに対してルール r1 が適用される．そして最終的に，'at3(at2(a, b), c, d)' が得られる．

なお型推定をパタンマッチング時に行なわない場合，まずノード n3 において r1，ノード n2 において r2 がマッチングするので，ノード n2 における変換が優先される．しかしながら，この違いは実用上問題とならなかった．以上のように本システムでは，1パスでパタンマッチングと型推定を行ない，続く1パスで書き換え処理を完了する高速な変換方式を実現した．

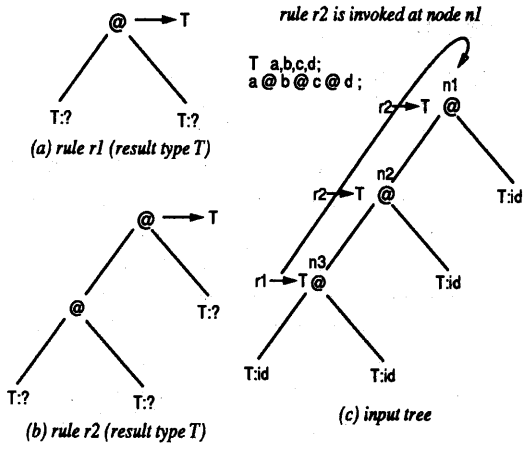

図 5: Bottom-up Pattern Matching with Type Inference

5 例

本章ではOPTECシステムによる簡単なC言語拡張について説明する．以下では，簡単なLindaのin操作の付加例を示す．なおLinda操作は[14]を参考にした．

5.1 Linda in 操作

Lindaは，並列動作を実現する機構であり，タプルスペースに対するin，rd，out，evalの4つの操作により構成される．以下にC-Lindaにおけるin操作の記述例を示す．

```
dict aMatrix;
int  row;
float data[SIZE];
row = 1;
in (aMatrix, row, ? data);  /* 式(1) */
in (aMatrix,? row, ? data); /* 式(2) */
```

in操作はデータ空間(タプルスペース)からの入力操作である．上記例では，aMatrixは行列を示すdict型のデータ(タプルスペース)，rowは行列の列を示すint型変数，またdataは列データを格納するfloat型配列である．

in文に現れるデータ並び(上記例の場合，aMatrix，row など)は，データ型を有するマッチング用テンプレートである．テンプレートに値を用いる場合，タプルの該当するアイテムとデータ型及び値の両方がマッチングしなければならない．一方，テンプレートには一種のワイルドカードであるフォーマルを用いることができる．フォーマルではデータ型のみのマッチングが行なわれ，マッチング後に，マッチしたアイテム値が指定した変数に代入される．フォーマルの指定は‘? f’のように‘? 変数名’のように記述する．

上記例 式(1) では，aMatrix，row は値として，data はフォーマルとしてマッチングに使用され，row で指定した列(第1列)のデータをタプルスペース aMatrix より取り出すことを指示している．一方 式(2) では，第2引数に‘?’を付加することにより任意の列のデータをタプルスペースから取り出すことを指示している．

5.2 記述例

以下にin操作の実現を示す．なお本例では簡単化のため，引数はdict型，行列の列を示すint型，列データを格納するfloat型の配列(float型へのポインタ)の3つに限定する．まず通常のC言語でサポートされていないフォーマルを扱うため，前置演算子‘?’を定義し，次にin操作の処理を定義する．

```
$operator(prefix,3) ? ;

$rule r1
 [in($(dict:ts),$(int:r),?$(&float:m))]
 {emit("in_sr(%a,%a,%a)",ts,r,m);}
$rule r2
 [in($(dict:ts),?$(int:r),?$(&float:m))]
 {emit("in_ar(%a,&%a,%a)",ts,r,m);}
$rule r3 [in($any)]
 {warn("not implemented");}
```

ルール r1 は列が指定された場合のin操作を示している．本ルールにマッチングするパタンは，emit文により関数呼びだし in_sr() に展開される．関数 in_sr() は，指定した列データを第一引数で指定したタプルスペースから取り出す実行支援ライブラリ関数であり，第2引数には int 型の値，第3引数には float 型へのポインタを与える．

一方，ルール r2 は任意の列に対するin操作を示しており，関数呼びだし in_ar() に展開する．ルール r1 と r2 の違いは，in 文の第2引数での‘?’の有無であり，ルール r2 は第2引数がフォーマルである場合を処理する．このため関数 in_ar() には，第2引数として int 型へのポインタを与える．またルール r3 はデフォルトの in 文の扱いを定義する．本例では，上記2つのルールにマッチングしない in 文があると警告文を出力する．そして前述の記述例の場合，式(1) はルール r1 と，また 式(2) はルール r2 とマッチし，以下の表現に変換される．

```
in_sr(aMatrix,row,data); /*式(1)変換後*/
in_ar(aMatrix,&row,data);/*式(2)変換後*/
```

6 おわりに

本稿では，C言語の問題向き拡張システムOPTECについて，その概要と言語拡張機構について述べた．本システムは，問題向きに拡張されたC言語表現を通常のC言語表現に変換するトランスレータ構築を支援するツールである．

本システムの特長は，データ型をパタン変換に導入することにより，単一のパタン変換の枠組でプログラム言語のシンタクスとセマンティクスの拡張を実現し，これをシステム自身及び生成される問題向き言語トランスレータの中核にしたことである．このため，

- パタン変換により容易にシンタクス拡張が可能である．
- オーバローディングにより同一形式の構文に対してセマンティクスの拡張が可能となった．このため従来のパタン変換を用いた言語拡張システムと比較し，より柔軟な言語拡張が可能となった．
- データ型をパタン変換に導入したことにより，式だけではなく文に対しても同じ枠組でオーバローディングを処理できる．このため，従来の演算子に対して定義するオーバローディングと比較し，より柔軟なオーバローディングが可能となった．

また実装においては，型推定を構文ツリーのパタンマッチング過程に導入することにより，高速な変換処理を実現した．現在のシステムでは，例えば1000ステップの入力を1秒以内で変換し，実用に耐える変換速度を達成している．表 1 に本システムのソフトウェア量を示す．

現在，本システムにより系統制御システムのユーザインタフェース構築用言語を実装し，応用システムの開発に適用中である．今後の課題としては，問題向きの記述に対するデバッギング手法の問題が残されている．また，本システムで実現したプログラム変換技術と視覚的プログラム技術とを組み合わせ，より生産性の高いプログラム開発支援の検討を考えている．

表 1: OPTEC のソフトウェア量(行)

	C ソース	Yacc
Parser Generator	2,500	400
Rule Compiler	3,500	2,200
Common Library	11,000	1,800
合計	17,000	4,400

参考文献

[1] N. Gehani and W.D. Roome. *The Concurrent C Programming Language.* Silicon Press, 1989.

[2] B.W. Kernighan and D.W. Ritchie. *The C Programming Language.* Prentice-Hall, 1987.

[3] J.R. Cordy, C.D. Halpern, and E. Promislow. TXL:A Rapid Prototyping System for Programing Language Dialects. In *Proc. 1988 Int.Conf.Comput.Lang.*, 280–285, 1988.

[4] J.R. Cordy and E. Promislow. Specification and Automatic Prototype Implementation of Polymorphic Objects in TURING Using the TXL Dialect Processor. In *Proc. 1990 Int. Conf. Comput. Lang.*, 145–154, 1990.

[5] B. Stroustrup. *The C++ programing language.* ADDISON-WESLAY, 1986.

[6] J. Katzenelson. Introduction to Enhanced C(EC). *Softw.Pract.Exp.*, 13(7): 551–576, 1983.

[7] B.W. Kernighan and R. Pike. *The UNIX Programming Environment.* Prentice-Hall, 1984.

[8] J.C. Cleaveland. Building Application Generators. *IEEE Software*, July, 25–33, 1988.

[9] J.J. Purtilo and J.R. Callahan. Parse-Tree Annotations. *CACM*, 32(12): 1467–1477, 1989.

[10] A.V. Aho and M. Ganapathi. Efficient Tree Pattern Matching: an Aid to Code Generation. In *Proc. 12th Ann. Symp. on POPL*, 334–340, 1985.

[11] C.M. Hoffmann and M.J. O'DONNELL. An Interpreter Generator using Tree Pattern Matching. In *Proc. 6th Ann. Symp. on POPL*, 169–179, 1979.

[12] A.V. Aho, R. Sethi, and J.D Ullman. *Compilers - Principles, Techniques, and Tools.* ADDISON-WESLAY, 1986.

[13] C.M. Hoffmann and M.J. O'DONNELL. Pattern Matching in Trees. *J.ACM*, 29(1): 68–95, 1982.

[14] W. Leler. Linda Meets Unix. *IEEE Computer*, 23(2): 43–54, 1990.

[15] 杉本 他, オブジェクト指向方式による“オブジェクト指向方式によるC言語拡張システム：OPTEC (1,2), 情処 38 全, 879–882, 1988.

[16] 小島 他, C言語拡張システム：OPTEC, 構文ツリーの書換え, 情処 39 全, 1321–1322, 1989.

[17] 小島 他, C言語の問題向き拡張用言語変換システム：OPTEC, トラバースによる書き換え, 情処 41 全, 5, 74–75, 1990.