ASN.1 ライトウェイト符号化規則用コンパイラの設計と評価

堀内浩規† 小花貞夫† 鈴木健二†

OSI（開放型システム間相互接続）のプレゼンテーション層や応用層におけるデータ要素は，ASN. 1 により定義され，符号化される．プレゼンテーション層や応用層のソフトウェアを効率的に開発するには，処理の複雑な ASN. 1 の符号化／復号を行うプログラムを自動生成する ASN. 1 コンパイラが有効であり，これまでに，基本符号化規則（BER）用のコンパイラが報告されている．近年，高速な OSI 通信を実現するため，BER より処理負荷を大幅に減少させるライトウェイト符号化規則（LWER）が注目されている．しかしながら，LWER を対象とするコンパイラは，高速処理に主眼を置いているため，従来の BER コンパイラとは符号化／復号関数などの生成方法が大きく異なる．本論文では，ASN. 1 による抽象構文定義から LWER による符号化／復号プログラムを自動生成する LWER コンパイラの設計とその評価結果について報告する．具体的には，ASN. 1 による抽象構文定義の各型に対応して，C 言語による符号化／復号関数と関数を呼び出す際に渡される引数の型定義等の生成方法を示すとともに，生成された符号化／復号プログラムの処理時間ならびにプログラム規模の評価を通じて，設計手法の有効性を示す．特に，生成する符号化／復号関数の処理では，抽象構文中の型定義に対応する変数値を符号化結果格納領域に直接書き込むことに加え，符号化時のコピー回数の削減や復号時のコピーの排除により高速化を図った．

Design and Evaluation of Compiler for ASN. 1 Light Weight Encoding Rules

Hiroki Horiuchi,† Sadao Obana† and Kenji Suzuki†

Abstract syntaxes of data elements in OSI presentation layer and application layer protocols are defined using Abstract Syntax Notation One (ASN.1), and for the encoding of those data elements in transfer syntaxes, Basic Encoding Rules (BER) has been standardized as an international standard. The ASN. 1 compiler which generates encoding/decoding programs from abstract syntaxes is very effective for developing protocol programs, and several works have been done in the design of a compiler for BER. Currently, it becomes necessary to realize high-speed ASN. 1 encoding/decoding to avoid the bottle-neck at end systems under high-speed communication network environments such as high-speed LAN and B-ISDN. For the purpose of enabling the high-speed ASN. 1 encoding/decoding, standardization of Light Weight Encoding Rules (LWER) is under way as one of the new encoding rules. However, the compiler for LWER is very different from BER in the mechanisms for generation of encoding/decoding programs. This paper discussed the design methodology of compiler for LWER and its evaluation. We clarified the mechanisms for generation of encoding/decoding programs for LWER and ensured the effectiveness of the design methodology through the evaluation of the encoding/decoding time and the size of generated programs. Especially, the decoding time for the actual protocol data units used in OSI application protocol was about 25 times faster than that in BER.

1. はじめに

OSI（開放型システム間相互接続）のプレゼンテーション層や応用層におけるデータ要素は，ASN. 1（抽象構文記法 1）[1] により定義され，符号化される．プレゼンテーション層や応用層のプロトコルを処理するソフトウェアを効率的に開発するには，処理の複雑な ASN. 1 によるデータ要素の符号化/復号を行うプログラムを生成するコンパイラ（ASN. 1 コンパイラ）が有効である．ASN. 1 コンパイラとしては，広く普及している基本符号化規則[2]（BER : Basic Encoding Rules）のためのコンパイラが報告されている[3)-5)]．これらのコンパイラでは，ASN. 1 による抽象構文定義の各型に対応して，C言語などのプログラミング言語による符号／復号関数と関数を呼び出す際に渡される引数の型定義など（以下，これらを符号化／復号プログラムと呼ぶ）を自動生成する方式をとっている．

近年，光 LAN や広帯域 ISDN 等の通信環境での

† 国際電信電話株式会社研究所
KDD R&D Laboratories

高速な OSI 通信を実現するために，BER より処理負荷を大幅に減少させるライトウェイト符号化規則 (LWER : Light Weight Encoding Rules)[7] の標準化が開始され，注目されている．LWER は計算機の内部表現に近い表現方法を符号化に採用しており，LWER コンパイラは，BER コンパイラとは符号化／復号プログラムの生成方法が大きく異なる[8]．特に，BER コンパイラの符号化／復号関数では，値の符号化形式への変換や識別子の付加を行うのに対し，LWER コンパイラの場合には値の設定された変数を直接符号化結果格納領域に書き込む必要がある．また，生成する符号化／復号関数の種類については，BER コンパイラが単一の転送構文を扱えば十分であるのに対して，自システムのワード長やバイト順序の異なる計算機との通信を可能とするために複数の転送構文への対応等が必要となる．しかしながら，これまでの LWER に関する報告[9]は，LWER の方式提案が重点で，コンパイラの符号化／復号関数の生成方法や複数の転送構文への対応等の設計手法は報告されていない．

本論文では，ASN. 1 による抽象構文定義から LWER による符号化／復号プログラムを自動生成する LWER コンパイラの設計手法とその評価について論じる．2章で，BER と LWER の概要，ならびに，従来の ASN.1 コンパイラについて述べ，3章で，LWER コンパイラの設計として，BER コンパイラとの違い，符号化／復号関数の生成方法，ならびに，複数の転送構文への対応方法を述べる．4章では，設計に基づいて実装したコンパイラが生成するプログラムの符号化／復号処理時間の評価を行い，最後に，5章で設計手法の有効性等を考察する．

2. BER/LWER の概要と従来の ASN. 1 コンパイラ

2.1 BER/LWER の概要

2.1.1 BER

基本符号化規則 (BER) では，ASN. 1 により定義されたデータ要素の型を，識別子 (ID)，データの長さ (LI)，コンテンツ (CO) の3要素からなるオクテット列として符号化する（ただし，LI が不定長の場合には，さらにコンテンツ終了 (EOC) が付加)．ID は型のクラスと番号等を示し，CO には，データ要素の値または，入れ子になったデータ要素が入る．また，プレゼンテーションコネクションの確立時に折衝される転送構文名（以下，転送構文と呼ぶ）は，BER では一種類定義されるのみである．

2.1.2 LWER

ライトウェア符号化規則 (LWER) は，符号化形式を計算機の内部表現に近づけることにより，変換を容易にし高速な符号化／復号処理を可能とするワード単位の符号化規則 (**表 1**) として，現在 ISO で作業草案が作成されており，以下の特徴を持つ．

（1） 符号化形式

①ワードを符号化の単位とする．

②型を識別するための識別子は付与しない．

③BOOLEAN や INTEGER 等の値が固定長である基本形の型や SEQUENCE および SET の構造形の型には値の長さを付与しない．

④IA 5 String 等の値が可変長となる型，SEQUENCE や SET の要素のうちオプショナルな型や SEQUENCE OF の要素数が可変な型は，ポインタ（以下ではオフセットと呼ぶ）を用いて，値や対応する要素を符号化の後部にまとめて配置する．なお，オフセットは，オフセットの位置と値の位置との相対位置を示す

表 1 主なライトウェイト符号化規則
Table 1 Summary of Light Weight Encoding Rules.

型　名	符号化方法
BOOLEAN	1ワードで符号化．値は FALSE (0)/TRUE (0 以外)．
INTEGER/ ENUMERATED	1ワードで符号化．値はワードサイズにより制限される．
REAL	64 bit で符号化．IEEE 標準フォーマット．
BitString/ CharacterString/ OctetString	長さ／オフセット／値の3組で表現．値はワードでアラインメントが行われる．値がない時，長さ／ポインタの値 0．
NULL	符号化されない．ただし，SET および SEQUENCE の要素で OPTIONAL の時オフセット（存在 1，存在せず 0)．
SEQUENCE/SET	SEQUENCE と SET の区別なし．抽象構文の定義順に要素を並べる．OPTIONAL, DEFAULT の要素はオフセット／値で表現．
SEQUENCE OF/ SET OF	要素数を示すカウンタ／オフセット／値列の3組で表現．
CHOICE	選択要素の識別のためのインデックス／選択された型の符号化の2組で表現．選択された型の符号化は，最も長い型にそろえてパディング．
OBJECT IDENTIFIER	Octet String として符号化．
Tagged	符号化されない．
ANY	長さ／オフセット／値の3組で表現．

(図 1).

(2) 扱うデータ構造に関する制約

①SEQUENCE 型と SET 型の区別をなくし，要素の符号化順は抽象構文定義の順とする．

②INTEGER の値の範囲，IA 5 String 等の長さ，SET OF/SEQUENCE OF の要素数等は，1ワードで表現できる範囲に限定される．

(3) 転送構文

転送構文はワード長とワードを構成するバイトの順序を組として複数定義される．現在，ワード長が 16/32/64 ビットの3種，バイトの順序が Big Endian か Little Endian かの2種の組合せによる計6組の転送構文が定義されている．

2.2 従来の ASN. 1 コンパイラ

これまで報告されている ASN. 1 コンパイラは，いずれも BER を対象としている[3)-5)]．これらの BER コンパイラは，以下のように抽象構文に依存しない汎用方式[5),6)]と抽象構文ごとに型定義を生成する専用方式[3),4)]とに大別できる[3),5)]．

① 汎用方式

抽象構文に依存せず，ASN. 1 の任意の型を表現する汎用的な型定義と符号化／復号関数をあらかじめ用意しておく方式である．抽象構文中のデータ構造は，汎用的な型定義を組み合せて表現する．符号化／復号関数はこのデータ構造を参照しながら符号化／復号処理を行う．この方式は，抽象構文に依存しない型定義と符号化／復号関数を利用するため，対象となる ASN. 1 の型定義が増えても，プログラム規模は，あまり増大しない特徴を持つ．

② 専用方式

抽象構文に依存し，各型定義ごとに直接対応したC言語の型定義と符号化／復号関数を自動生成する方式である．専用方式では，各型に対して専用の符号化／復号関数が生成されるため，汎用方式のようにデータ構造情報を参照するオーバヘッドがなく，符号化／復号処理時間が高速となる特徴を持つ．

3. LWER コンパイラの設計

3.1 基本方針

LWER コンパイラの設計に当たり，以下の基本方針を立てた．

(1) 専用方式の採用

2.2 節で述べた従来の BER コンパイラと同様に，LWER コンパイラにおいても汎用方式と専用方式のいずれも適用可能であるが，以下の理由により専用方式を採用した．

①LWER が高速な符号化／復号を目的としており，処理効率を重視する必要がある．

②専用方式は，LWER が使用する計算機の内部表現に対応したC言語の型定義を持つ変数を符号化／復号の処理に直接利用可能である．

(2) 性能

符号化／復号処理では以下の点に留意して高速化を達成する．

①値の設定された変数を直接符号化結果格納領域（以下ではバッファと呼ぶ）に書き込む．

②データのコピーを極力削減する．

(3) 複数転送構文への対応

不特定の計算機と通信するために，自システムと同じワード長とバイトの順序となる転送構文（以下，自システム転送構文と呼ぶ）に加え，自システムと異なるワード長やバイト順序を持つ転送構文（以下，他システム転送構文と呼ぶ）にも対応する．

この基本方針に基づき，以下のように LWER コンパイラを設計した．

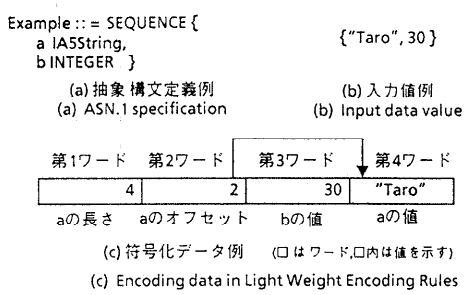


図 1 ライトウェイト符号化規則の符号化例
Fig. 1 An example of encoding data in Light Weight Encoding Rules.

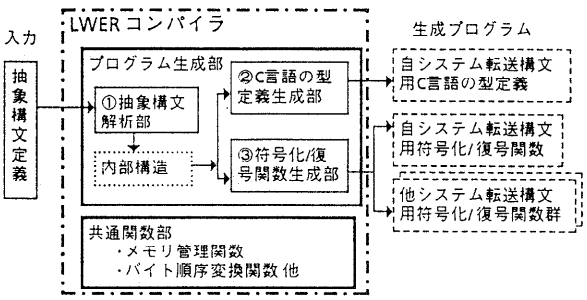


図 2 LWER コンパイラのソフトウェア構成
Fig. 2 Software structure of LWER Compiler.

3.2 ソフトウェア構成

LWER コンパイラのソフトウェア構成は**図 2** に示すように，プログラム生成部と共通関数部からなる．

（1） プログラム生成部

①抽象構文解析部では，抽象構文定義中の ASN. 1 の型定義の構文解析を行い，内部構造に変換する．②C 言語の型定義生成部では，内部構造から，ASN. 1 の型定義に対応する C 言語の型定義を，3.3 節の生成方法に従って生成する．③符号化／復号関数生成部では，内部構造から，抽象構文定義中で定義された型定義ごとに 6 種類すべての転送構文に対応した，符号化／復号関数群を 3.4 節の生成方法に従って生成する．また，符号化／復号処理の高速化と転送構文ごとの切り出しを容易とするため，転送構文ごとに別の関数を生成する．

（2） 共通関数部

抽象構文における ASN. 1 の型の種類に依存せず，生成した符号化／復号関数によって共通的に使用される関数をあらかじめ用意する．

以下では，C言語の型定義と符号化／復号関数の生成方法，ならびに，共通関数について述べる．これらは，専用方式の BER コンパイラ（以下，BER コンパイラと呼ぶ）と**表 2** に示す違いを持つ．

表 2 BER コンパイラと LWER コンパイラの相違
Table 2 Differences between BER Compiler and LWER Compiler.

構成要素		BER コンパイラ	LWER コンパイラ
C 言語型定義		ASN. 1 の型定義に対応する型定義.	ASN. 1 の型定義に対応し，かつ，動作計算機のワード長／バイト順序と等しい転送構文に従った型定義.
符号化／復号関数	符号化／復号方式	LI の計算，値の符号化形式への変換や ID の付加を行う.	値の設定された変数を直接バッファに書き込む.
	転送構文	単一の転送構文のため対処の必要なし.	6 種類の転送構文に対応．使用転送構文の指定は，符号化／復号関数の引数で行う.
共通関数		基本形の符号化／復号関数.	バイト順序変換，ワードサイズ変換，文字列の符号化／復号関数.

表 3 C言語型定義および符号化／復号関数の生成規則
Table 3 Rules on generation of C type definition and encoding/decoding function.

ASN. 1 の型	生成するC言語の型	生成する符号化関数の処理	生成する復号関数の処理
BOOLEAN/INTEGER/ ENUMERATED	int	値の設定された変数を代入.	バッファの先頭アドレスを代入.
REAL	double	同　上	同　上
BitString/OctetString/ CharacterString/ Object Identifier/ANY	値の長さ (int length) と値を指すポインタ (char *field) からなる構造体 (struct field).	①値の長さと値を指すポインタをまとめてコピー．②バッファ内のポインタの値をオフセットに変換．③値をバッファの後部にコピー.	①バッファの先頭アドレスを代入．②値を指すオフセットをポインタの値に変換.
NULL	オプショナルな要素の時のみ int を生成.	変数が在る場合にはコピー.	変数が在る時，バッファの先頭アドレス代入.
SEQUENCE SET	対応する要素を持つ構造体 (struct) で，オプショナルな要素は型定義へのポインタとする.	①sizeof 演算子で求めた複数要素の領域をコピー．②オプショナルな要素と可変長の要素のポインタの値をオフセットに変換．③可変長の値やオプショナルな要素を後部にコピー.	①バッファの先頭アドレスを代入．②オプショナルな要素がある時，オフセットをポインタの値に変換．③要素は各型の復号に従う.
SEQUENCE OF SET OF	要素数を示すカウンタ (int number) と要素の型へのポインタを持つ構造体 (struct).	①要素数を示すカウンタと要素を指すポインタをまとめてコピー．②バッファ内のポインタの値をオフセットに変換．③バッファ後部にカウンタの数だけ，要素をコピー.	①バッファの先頭アドレスを代入．②要素を指すオフセットをポインタの値に変換.
CHOICE	選択要素を識別するためのインデックス (int index) と対応する要素を持つ共用体 (union) とする.	①インデックスと共用体をまとめてコピー．②選択要素にポインタが有る時は，バッファ内のポインタの値をオフセットに変換．③バッファ後部に可変長の型の値等を書き込む.	①バイト列の先頭アドレスを代入．②選択要素にオフセットがある時，ポインタの値に変換.

3.3 C言語型定義の生成方法

ASN. 1 の型定義からC言語の型定義の生成は**表 3** に示す生成規則を適用する．この際，BER コンパイラとは異なり，処理速度を高めるため，自システム転送構文に従う制約を加え，以下のように対応づけを行う．

BOOLEAN 型に対して通常 BER コンパイラでは unsigned char を使用する[4]が，LWER はワード単位の符号化によって高速化を図っているため，int を対応させた．SEQUENCE および SET には対応する要素を持つ構造体を生成する．オプショナルな要素を持つ場合には，符号化時のオフセットとの対応づけのため，要素を持つ型定義へのポインタとし，文献 3) 等の BER コンパイラで用いられるオプショナル用のフラグは使用しない．また，SEQUENCE OF については，BER コンパイラでは複数繰り返される要素に対してリスト構造の型定義をとることが多い[3),4)]が，C言語ではポインタで任意の配列を表現できるとともに，符号化時のオフセットとの対応づけが容易なため，ポインタを使用する．なお，可変長の値を持つ BitString や OctetString 等は，BER コンパイラと同様に値の長さと値を格納するポインタをメンバに持つ構造体とした．

入力される抽象構文定義例と生成されるC言語の型定義例を**図 3**（a），（b）に示す．各 ASN. 1 型定義に対応する構造体 struct Record, struct ChidInf および struct Name が生成されている．

3.4 符号化／復号関数の生成方法

BER コンパイラの符号化／復号関数の処理では，変数の値から符号化形式への変換，LI の計算や ID の付加等を行うのに対し，本コンパイラの符号化（復号）関数の処理は，3.3 節で示したC言語の型を持つ変数をバッファに（から）直接書き込む（読み込む）ことを基本とする．図3（c）には，図3（a）の抽象構文に対するユーザ作成プログラム例を示す．ここでは，生成する型定義 struct Record に値を設定した後，符号化関数 encode_lwer () を呼び出している．この符号化関数は，動的に転送構文の変更を可能とするため，第4引数に転送構文の種類を指定することとしている．以下では，抽象構文の各型ごとに生成する符号化／復号関数の生成方法を述べる．

3.4.1 符号化関数の生成

（1） 自システム転送構文の場合

一般的に，コピーの回数を減らすことは，関数呼び出しの回数を減らすこととなり高速化が図れる．しかしながら，通常の応用層のデータ要素は，固定長の要素だけでなく可変長やオプショナルな要素が混在するため，符号化される変数の値はメモリ上の連続領域にない．このような場合でも，コピーの回数を減らすため，固定長の型の領域および可変長の型とオプショナルな型の固定長部分をまとめてコピーし，可変長の値を分離して扱う以下の方式をとった．

①sizeof 演算子で求められた型の長さの分だけ，変数の先頭から関数 memcpy() を用いて，含まれる固定長の要素，可変長の値以外の長さおよびポインタの部分をまとめてコピーする．

```
Record ::= SEQUENCE {   number     INTEGER,
                        name       Name,
                        dateOfHire IA5String,
                        children   ChildInf }
ChildInf ::= SEQUENCE OF Name
Name ::= SET {   first  [1] IA5String,
                 last   [2] IA5String }
```

(a) 入力抽象構文定義例

(a) Input ASN.1 Specification

```
struct field {  int     length;
                char    *field; };
struct Name {struct  field first ;
             struct  field last; };
struct ChildInf {   int number;
                    struct Name *name ; };
struct  Record {    int  number;
                    struct Name    name;
                    struct field   dateOfHire;
                    struct ChildInf children; };
```

(b) 生成されるC言語の型定義例

(b) Generated type definition in C language

```
#include "lwer.h"
#define StrSet(ptr,data) (ptr.field = data; ptr.length = strlen(data))
main()
{
    struct Record t ; char *malloc() ; ERROR error ; String out ;
/* 値の設定 */
    t.number = 123 ;
    StrSet (t.name.first, "Hiroki") ;
    StrSet (t.name.last, "HORIUCHI") ;
    StrSet (t.dateOfHire, "199204") ;
    t.children.number = 1 ;
    t.children.name = (struct Name *)malloc(sizeof(struct Name)) ;
    StrSet (t.children.name->first, "Taro") ;
    StrSet (t.children.name->last, "HORIUCHI") ;

/** 符号化 **/
  if( encode_lwer (&t, &out, IRecord, FTLWER32B, &error) = = FALSE )
    { printf("Encode Fail") ; exit(1) ; }
}
```

(c) ユーザプログラム例

(c) An example of user program

図 3 抽象構文に対応するC言語の型定義生成例とユーザプログラム例

Fig. 3 Examples of input ASN. 1 specification, generated type definition in C language and user program.

②コピーしたバッファ内のポインタ値をオフセットに変換する.
③バッファの後部に可変長の型の値とオプショナルな要素のコピーを行う.

この方式を基本とした，各型に対する符号化関数の生成規則を表3に示す．なお，本コンパイラでは，まず，データ要素を作成するのに必要となる領域（データの長さ）の計算を行い，データの長さに応じた領域の確保を行った後，上記の手順でバッファに変数を書き込む．このデータの長さの計算においても計算回数を減らすため，固定長の型と可変長の型の値以外の部分の長さをまとめて求め，可変長の値とオプショナルな要素に対してのみ，変数をたどり符号化に必要なデータの長さを加算する.

図3(a)の抽象構文の入力に対して，型Record, ChildInf, Nameに対する符号化関数（32ビット/BigEndian）の生成例と符号化で使用されるマクロを**図4**に示す．これらの関数は，複数の要素の固定長部分をまとめてコピーした後に呼び出される.

（2） 他システム転送構文の場合

ワード長が同じで，バイト順序のみが異なる転送構文の符号化関数の処理は，（1）で述べた自システムの転送構文の処理に，バッファ内のオフセット等に対して以下の追加規則（I）を追加する．ワード長が異なる転送構文の場合には，生成するC言語の型定義とアライメントが異なるため，自システムの転送構文で使用した複数の要素をまとめてコピーする処理は適用できず，要素ごとにコピーを行いつつ以下の追加規則(II)を適用する．さらに，ワード長に加え，バイト順序と異なる転送構文の場合には追加規則(I)も適用する.

追加規則

（I） バイト順序が異なる転送構文の場合

オフセットやint型を持つINTEGERの値やSET OFのカウンタ等の要素に対して，バッファ内に書き込む際にバイト順序変換を行う．ただし，OctetString等の値は，オクテット列として意味を持ち，ワードとして扱う必要がないので，バイト順序変換は行わない.

（II） ワード長が異なる転送構文の場合

オフセットやint型を持つ要素に対して，自システムのワード長より小さい転送構文の場合には，値があふれないことを検査した後必要なバイトのみを，また，ワード長が大きい転送構文の場合は，必要なバイトを付加してバッファへ書き込む．OctetString等の値に対しては，アライメントを他システムのワード長に合わせる.

3.4.2 復号関数の生成

（1） 自システム転送構文の場合

符号化同様にコピーを減らす工夫が高速処理を行う上で重要となる．OctetString等の値の復号で，受信したバッファを復号の結果の値として利用することで，データのコピーを排除することは文献4)のBERコンパイラでも採用されている．LWERコンパイラにおいては，同様な手法を用いて，OctetStringだけ

```
#define     StringCopy( iDATA, oFIXF, oVARF ) ¥
    { if( oFIXF.length ) ¥
      { int POFF, PSIZE; ¥
        POFF = (int)oVARF - (int)&(oFIXF.field); ¥
        POFF >> = 2; oFIXF.field = (char *) POFF; ¥
        memcpy( oVARF, iDATA.field, oFIXF.length ); ¥
        PSIZE = StringLength( iDATA ); if( (PSIZE - iDATA.length) ) ¥
          for( POFF = iDATA.length; POFF < PSIZE; POFF + + ) ¥
            oVARF[POFF] = '¥0'; oVARF + = PSIZE; } ¥
      else  oFIXF.field = (char *) NULL; }
#define     OfCopy( iDATA, oFIXF, oVARF, bOX ) ¥
    {   int POFF; ¥
        POFF = (int)oVARF - (int)&(((OfTypeDef *)oFIXF)->child); ¥
        POFF >> = 2; ¥
        ((OfTypeDef *)oFIXF)->child = (char *) POFF; ¥
        memcpy( oVARF, iDATA, sizeof(bOX) * ((OfTypeDef *)oFIXF)->OfCnt ); ¥
        oVARF + = (sizeof(bOX) * ((OfTypeDef *)oFIXF)->OfCnt); }
```

(a) 符号化に使用される共通関数の一部
(a) Common functions for encoding

```
mERecord32No( Inp, str1, str2 )   /* 型Recordの符号化関数 (32Bit / BigEndian) */
    struct Record   *Inp;   /* 入力変数 */
    struct Record   *str1;  /* 符号化結果の格納位置*/
    unsigned char  **str2; /* 符号化後部の位置*/
{
    if( mEName32No( &(Inp->name), &(str1->name), str2) = = FALSE )
        /* name符号化*/   return FALSE;
    StringCopy( Inp->dateOfHire, str1->dateOfHire, (*str2) );
        /* dateOfHire 符号化 **/
    if( mEChildInf32No( &(Inp->children), &(str1->children), str2) = = FALSE )
        /* childrenの符号化 */  return FALSE;
    return TRUE;
}
mEChildInf32No( Inp, str1, str2 )   /* 型Childの符号化関数 (32Bit / BigEndian) */
    struct ChildInf  *Inp;   /* 入力変数 */
    struct ChildInf  *str1;  /* 符号化結果の格納位置*/
    unsigned char  **str2; /* 符号化後部の位置*/
{   if( Inp->number ){
        int      cnt;          struct Name     *tstr1;
        tstr1 = (struct Name *) (*str2);
        OfCopy( Inp->name, str1, (*str2), struct Name );
        for( cnt = 0; cnt < Inp->number; cnt + + )
            if( mEName32No(&((Inp->name)[cnt]), &(tstr1[cnt]), str2) = = FALSE )
                /* Nameの符号化 */     return FALSE;
    }else   str1->name = NULL;
    return TRUE;
}
mEName32No( Inp, str1, str2 )  /* 型Nameの符号化関数 (32Bit / BigEndian ) */
    struct Name     *Inp;   /* 入力変数 */
    struct Name     *str1;  /* 符号化結果の格納位置*/
    unsigned char  **str2; /* 符号化後部の位置*/
{   StringCopy( Inp->first, str1->first, (*str2) );     /* firstの符号化 */
    StringCopy( Inp->last, str1->last, (*str2) );      /* lastの符号化 */
    return TRUE;
}
```

(b) 生成された符号化関数
(b) Generated encoding functions

注) コメントは関数の内容説明のため，コンパイラ出力に追加している.

図4 生成された符号化関数例
Fig. 4 An example of generated encoding functions.

でなく，固定長の値が単独の場合や固定長の要素を複数持つ SEQUENCE OF 等についてもバッファを復号の結果として直接利用するようにした．さらに，可変長が含まれる場合にも，復号時にバッファの値が一部変更されることを許容すれば，コピーを排除してオフセットからポインタへの変換のみで復号を可能とする以下の方式をとった．表3には ASN.1 の型ごとの復号処理を示す．

①復号結果となる変数のアドレスとして，バッファの先頭アドレスを使用する．

②オフセットがある場合には，オフセットをワード長からバイト長を示すように変換し，バッファの先頭アドレスの値を加えて，ポインタ値として使用する．

図3(a)の抽象構文の入力に対して生成する復号関数ならびに復号で使用されるマクロを**図5**に示す．図5の復号関数では，可変長の型とオプショナルな型に対して，それぞれ，オフセットからポインタの値への変換を行うマクロ StringRecover(), PointRecover() が呼び出される．

(2) 他システム転送構文の場合

ワード長が同じで，バイト順序のみが異なる転送構文の場合には，(1)で述べた自システム転送構文のコピーを排除した復号処理に，バッファ内で，オフセットや int 型に対応する値のバイト順序変換を行う処理を，②のオフセットをポインタの値に変換する処理の前に追加する．ワード長が異なる転送構文の場合には，自システム転送構文の場合と異なり要素ごとにバッファから変数へコピーを行い，この際に3.4.1節(2)の追加規則(II)の逆の規則を適用してワード長変換を行う．さらに，バイト順序も異なる転送構文の場合にはバイト順序変換も追加される．

```
#define   StringRecover( sTR ) ¥
{   int PADR;  PADR = (int) sTR.field;  PADR <<= 2; ¥
    PADR + = (int)&(sTR.field); sTR.field = (char *) PADR;  }
#define   PointRecover( pOINT, bOX ) ¥
{   int PADR;  PADR = (int) pOINT;  PADR <<= 2; ¥
    PADR + = (int)&(pOINT);  pOINT = (bOX) PADR;  }
```

(a) 復号に使用される共通関数の一部
(a) Common functions for decoding

```
/* 型Recordの復号関数 (32Bit/BigEndian) */
DRecord32No( Outp )
    struct Record   *Outp;
{   if( DName32No( &(Outp->name) ) = = FALSE ) return FALSE;
    StringRecover( Outp->dateOfHire );
    if( DChildInf32No(&(Outp->children)) = = FALSE ) return FALSE;
    return TRUE;
}
/* 型Childの復号関数 (32Bit/BigEndian) */
DChildInf32No( Outp )
    struct ChildInf  *Outp;
{   if( Outp->number ){   int      cnt;
        PointRecover( Outp->name, struct Name * );
        for( cnt = 0; cnt < Outp->number; cnt + + ){
            if( DName32No( &(Outp->name[cnt]) ) = = FALSE )
                return FALSE;  }
    }else   Outp->name = NULL;
    return TRUE;
}
/* 型Nameの復号関数 (32Bit/BigEndian) */
DName32No( Outp )
    struct Name   *Outp;
{   StringRecover( Outp->first );  StringRecover( Outp->last );
    return TRUE;
}
```

(b) 生成された復号関数
(b) Generated decoding functions

注) コメントは関数の内容説明のため，コンパイラ出力に追加している．

図5 生成された復号関数例
Fig. 5 An example of generated decoding functions.

3.5 共通関数部

BER コンパイラの共通関数は，抽象構文の ASN.1の型に依存しない INTEGER や IA5 String 等の ASN.1 の基本形の型の符号化/復号関数等が用意されるのに対して，本コンパイラの共通関数部では，複数転送構文をサポートするために，バイト順序を Big Endian から Little Endian に変換する関数やワード長の変換を行う関数，String 系の符号化／復号を行う関数等が含まれる（**表4**）．

4. 生成される符号化／復号プログラムの処理時間の評価

3章の設計に基づき，SUN および VAX (OS は VMS) 上に LWER コンパイラを実装し，典型的なデータ構造と実応用層プロトコルのデータ構造を用いた評価を行った．

4.1 測定条件

測定には SUN 3/260（ワード長 32 ビット/バイト順序 BigEndian）を使用し，符号化／復号処理時間は1000回ループさせた平均を示した．また，比較のため，文献3) のコンパイラを用いた BER の符号／復号処理時間も同じ計算機環境で測定した．

4.2 典型的なデータ構造を用いた場合の測定結果

典型的なデータ構造として，1)符号化が構造体のコ

表4 共通関数部に含まれる関数例
Table 4 Functions in common function part.

関数	機能
バイト順序変換	BigEndian と LittleEndian 間の変換.
ワードサイズ変換	各ワード長 (16/32/64) 間の変換.
型 SEQUENCE OF/SET OF 符号化／復号	要素数を示すカウンタとオフセットの符号化／復号.
String 系の型の符号化／復号	OctetString や BitString 型の符号化／復号.
メモリ管理	メモリの割り付け／解放を管理する.

ピーだけで済む固定長の要素のみの場合，2)可変長の要素を含み，符号化が構造体のコピーに加えて，ポインタ値のオフセットへの変換と後部への値のコピーが必要となる場合について，6種類の転送構文に対し符号化／復号処理時間を測定した．また，BER, LWERの符号化データ長の結果も併記した．

（1） 固定長の要素のみの場合

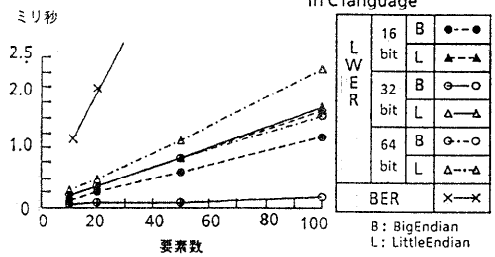
図6（a）に示すSEQUENCEでINTEGERの要素を複数持つ抽象構文定義（対応するC言語型定義を図6（b）に示す）に対して，要素数を変化させた場合の符号化/復号処理時間を図6（c），（d）に示す．

```
ClassA ::= SEQUENCE {
  elm1  INTEGER,
  elm2  INTEGER,
  elm3  INTEGER,
    .
    .
  elmn  INTEGER  }
```

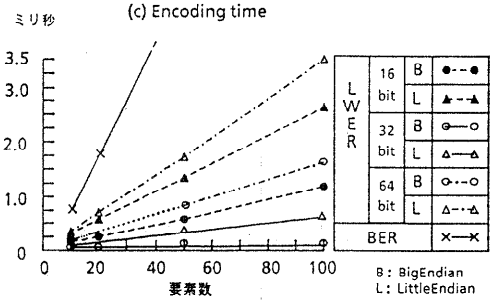
(a) 抽象構文定義
(a) ASN.1 specification

```
struct ClassA {
    int  elm1;
    int  elm2;
    int  elm3;
      .
      .
    int  elmn; };
```

(b) 生成されるC言語の構造体
(b) Generated type definition in C language



(c) 符号化処理時間
(c) Encoding time



(d) 復号処理時間
(d) Decoding time

符号化規則＼要素数		10	20	50	100
LWER	16bit/word	20	40	100	200
	32bit/word	40	80	200	400
	64bit/word	80	160	400	800
BER		32	62	153	304

(e) 符号化データ長
(e) Length of encoding data

図6 固定長の要素のみの場合の符号化／復合処理時間
Fig. 6 Encoding/decoding time in fixed length type.

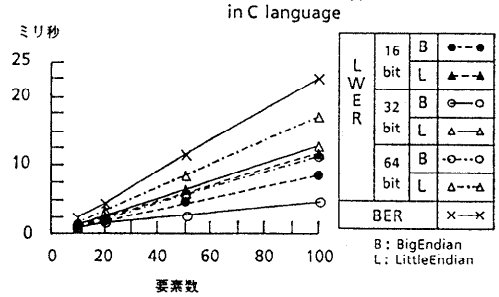
図6（c），（d）より，自システム転送構文（32ビット/BigEndianの場合）の符号化／復号処理時間は，バイト順序変換とワードサイズの変換が不要であることと，符号化処理時に複数の要素をまとめてバッファに書き込む方法と復号処理時のコピーを排除する方法により，BERより20倍以上高速となった．また，他システム転送構文の場合には，符号化処理では，16ビット/BigEndianの場合がコピーするデータ長が短く，バイト順序変換も不要のため最も高速となり，復号処理では，32ビット/LittleEndianの場合がコピーを排除した復号処理により最も高速となった．

```
Group ::= SEQUENCE OF
            Person
Person ::= SEQUENCE {
  number  INTEGER,
  nameIA5String }
```

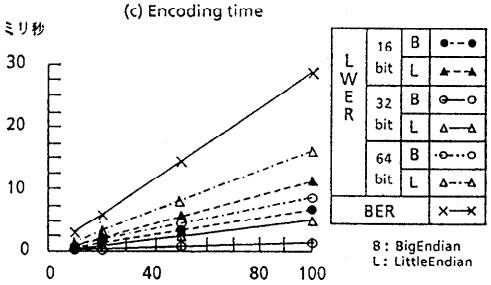
(a) 抽象構文
(a) ASN.1 specification

```
struct Person { int number;
    struct { int length;
        char *field; } name;};
struct Group { int count;
    struct Person *person;};
```

(b) 対応するC言語の構造体
(b) Generated type definition in C language



(c) 符号化処理時間
(c) Encoding time



(d) 復号処理時間
(d) Decoding time

符号化規則＼要素数		10	20	50	100
LWER	16bit/word	124	244	604	1204
	32bit/word	208	408	1008	2008
	64bit/word	336	656	1616	3216
BER		122	248	604	1204

(e) 符号化データ長
(e) Length of encoding data

図7 可変長の要素を含む場合の符号化／復号処理時間
Fig. 7 Encoding/decoding time in variable length type.

64ビット/LittleEndian の場合はバイト順序変換とデータ長の増加のため低速となったが，BER よりは高速であった．

（2） 可変長の要素を含む場合

図7（a）に示す可変長の IA 5 String を含む要素の並びとなる抽象構文定義（対応するC言語型定義を図7（b）に示す）に対して，要素数を変化させた場合の符号化／復号処理時間を図7（c），（d）に示す．

図7（c），（d）より，自システム転送構文の場合の符号化／復号処理時間は，それぞれ BER の約4.6倍/約 33.3 倍高速となり，復号処理時間が大幅に短縮されている．他システム転送構文の場合には，（1）同様，符号処理では16ビット/BigEndian の場合が，復号処理では32ビット/LittleEndian の場合が最も高速となった．また，ワード長が32ビットの転送構文の場合の符号化データ長は BER の約1.8倍と大きくなった．

4.3 実際の応用層プロトコルのデータ要素のヘッダを使用した場合の測定結果

実際の応用層プロトコルでよく使用されるデータ要素のヘッダの構造は，以下の3種類に大別できる（**表5**）．

[構造a] 一般的に構造形の要素は少なく，しかも基本形の長さが小さい場合．（ディレクトリ，OSI 管理，RDA プロトコル等の操作要求を行うデータ要素を想定）

[構造b] 一般に基本形の長さは小さいが，構造形の要素が多い場合．（ディレクトリ，OSI 管理，RDA プロトコル等の操作応答が数 10 個以上の場合や FTAM におけるシーケンシャルフラットファイルの転送のデータ要素を想定）

[構造c] 特定の基本形の長さの大きな値を持つ場合．（MHS で運ばれるテキスト電文や FTAM における無構造ファイルの転送のデータ要素を想定）

上記の構造の分類に従って，実際の応用層プロトコルのデータ要素における符号化／復号処理時間とデータ長を表5に示す．ここでは，構造a，b，cの例として，それぞれ，CMIP (Common Management Information Protocol) の m-Get 操作要求，CMIP の m-Get 操作結果，84年版 MHS の IM-UAPDU を用いた．

5. 考 察

5.1 生成プログラムの符号化／復号処理性能

5.1.1 符号化／復号処理時間

LWER コンパイラにより生成された符号化／復号プログラムの処理時間は，自システムと同一の転送構文を使用する際には，固定長の要素のみの場合 BER と比較して20倍以上高速となり，FTAM や RPC 等で転送されるユーザデータが整数の一次配列となる場

表5 実際の応用層プロトコルのデータ要素における符号化／復号処理時間とデータ長
Table 5 Encoding/decoding time and length of encoding data in actual OSI Application Protocol data units.

構造	データ要素	符号化規則		符号化時間（ミリ秒）	復号時間（ミリ秒）	データ長（オクテット）
a	m-Get 操作要求（注1）	LWER（注4）	B	0.51	0.10	216
			L	1.05	0.37	
		BER		1.89	2.32	147
b	m-Get 操作結果（注2）	LWER	B	3.09	0.46	1,424
			L	6.60	1.74	
		BER		10.89	12.27	658
c	IM-UAPDU（注3）	LWER	B	0.56	0.09	1,252
			L	0.80	0.14	
		BER		2.01	2.30	1,120

（注1） m-Get 操作要求は，相対識別名を5個持ち，同期のモードおよびスコープの指定をして，全属性の読み出し要求をした場合．
（注2） m-Get 操作結果は，長さ8オクテット以下の50個の属性値の検索結果を示す場合．
（注3） IM-UAPDU は，1K オクテットのボディパートを持つ場合．
（注4） LWER において，B：ワード長32ビット，バイト順序 BigEndian の転送構文，L：ワード長32ビット，バイト順序 LittleEndian の転送構文を示す．

合の符号化にきわめて強力であることがわかった．また，実際の応用層プロトコルのデータ要素のヘッダで，自システムの転送構文を使用した場合には，4.3節の3つの構造のいずれに対しても符号化で約3～4倍程度高速，復号で約25倍程度高速となり，復号時間がコピーを排除した方法により大幅に向上している．また，符号化／復号処理時間の合計では，約7倍程度高速となる．自システムと異なる転送構文の場合においても，一方の計算機は自システムの転送構文を使用することとなり，例えば，4.3節の m-Get 操作結果では，通信時の符号化および復号時間の合計が，LWER で約4.8ミリ秒，BER で23.2ミリ秒となり約4.8倍の高速となる．また，符号化規則の提案に重点をおき，好条件の場合の値が文献 8) で報告されているが，測定条件は明確でないものの，BER との相対値における符号化／復号処理速度は同等以上であった．

比較のため使用した BER コンパイラは 4 MIPS 程度の計算機において，広域網 (64 Kbps) からイーサーネットの LAN (10 Mbps) 程度のネットワーク上で有効な符号化／復号処理時間を達成していると報告されている[3]．今回実装した LWER コンパイラが生成する符号化／復号プログラムは，この BER コンパイラより一桁近く高速な符号化／復号を達成しており，高速通信環境においてもエンドシステムの処理がボトルネックとならない性能を達成し得ると考えられる．

5.1.2 符号化／復号処理におけるコピー削減の効果

符号化処理におけるデータの長さ計算の回数削減とコピー回数の削減，ならびに，復号処理におけるコピーの排除の効果を評価した．比較を行った符号化処理では，固定長の要素のみから構成される型は，複数要素をまとめてコピーするが，その他の要素は，構成要素ごとに変数からバッファへのコピーを行う．また，比較を行った復号処理では，固定長の要素のみから構成される場合と OctetString 等の値の復号はオフセットからポインタへの変換で行うが，その他の要素については，構成要素ごとにバッファから変数へのコピーを行う．

この結果，4.3 節で示した CMIP の m-GET 操作の場合で，採用した方法は符号化で約1.3倍，復号で約7.6倍に速度が向上しており，本手法の有効性を確認した．

5.2 生成プログラムの規模

LWER コンパイラを使用した場合の FTAM と CMIP の符号化／復号プログラムの規模を**表 6** に示す．この符号化／復号関数のステップ数はワード長が 32 ビットで，2つの転送構文 (BigEndian/LittleEndian) をサポートした場合である．高速化と転送構文ごとの切り出しを容易とするため，転送構文ごとに専用プログラムとしている点により，サポートする転送構文の数に比例してステップ数が増加する傾向にある．しかしながら，現在のワークステーション等では，ワード長が一種類 (32ビット) でバイト順序が2種類 (Big Endian/Little Endian) のものが一般的であるので，2種類の転送構文のサポートで十分であることを考慮すれば，FTAM で19.5kステップ，CMIP で12.7k ステップとなり，さほど大規模とはならない．また，自システム転送構文のみをサポートした場合のプログラム規模は，評価で利用した BER コンパイラより若干小さくなる．

表 6 符号化／復号プログラム規模 (C言語ステップ数)
Table 6 Size of encoding/decoding programs.

抽象構文	C言語型定義等	符号化／復号関数	共通関数	計	BER
FTAM	0.7k	17.6k (54.1k)*	1.2k	19.5k (55.0k)*	11.6k
CMIP	0.5k	11.0k (29.6k)*		12.7k (31.3k)*	10.4k

* ( )内の数字は6種類の全転送構文を実装した場合のステップ数

6. おわりに

本論文では，ASN.1 ライトウェイト符号化規則 (LWER) を扱う OSI のプレゼンテーション層と応用層のプロトコル処理プログラムを効率的に開発可能とするため，ASN.1 による抽象構文定義から LWER 用の符号化／復号プログラムを自動生成する LWER コンパイラの設計手法とその評価について論じた．

具体的には，ASN.1 による抽象構文定義の各型に対応して，C言語による符号化／復号関数と関数を呼び出す際に必要となる引数の型定義などの符号化／復号プログラムの生成規則を示した．特に，符号化／復号関数の生成では，従来の基本符号化規則 (BER) のためのコンパイラとは異なり，抽象構文定義に対応するプログラミング言語の型を持つ変数値を符号化結果格納領域に直接書き込むとともに，コピー回数を削減した符号化処理やコピーを排除した復号処理の実現方法，ならびに，複数転送構文への対処方法を示した．また，生成された符号化／復号プログラムの処理時間

およびプログラム規模を測定し，提案したLWERコンパイラの設計手法の有効性を実証した．実際の応用層プロトコルのヘッダの符号化／復号処理においては，BER コンパイラの生成する符号化／復号プログラムと比べて，符号化／復号ともに高速となり，特に，実際の応用層プロトコルのヘッダの復号では約25倍程度高速となった．

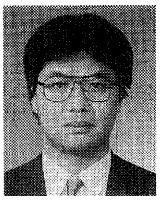
今後，高速通信環境で OSI 通信を行う際にエンドシステムの処理がボトルネックとなることを回避するために，ASN. 1 の符号化規則として LWER を使用する機会が増えることが予想され，LWER コンパイラが通信処理プログラムの開発に重要なツールとなると考えられる．

謝辞 日頃御指導頂く国際電信電話(株)研究所 浦野義頼 所長，眞家健次 次長，ならびに，御討論頂いた同研究所通信網支援ソフトウェアグループ 浅見徹リーダに感謝します．また，本論文作成にあたり，様様な御助言，御指導を賜った文部省学術情報センター 小野欽司 教授に深く感謝いたします．

参 考 文 献

1) ISO 8824: Information Processing Systems—Open System Interconnection—Specification of Abstract Syntax Notation One (ASN. 1) (1988).
2) ISO 8825: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN. 1) (1988).
3) Hasegawa, T., Nomura, S. and Kato, T.: Implementation and Evaluation of ASN. 1 Compiler, *J. Info. Process.*, Vol. 15, No. 2, pp. 157-167 (1992).
4) Neufield, G. and Yang, Y.: The Design and Implementation of ASN. 1—C Compiler, *IEEE Trans. Softw. Eng.*, Vol. 16, No. 10, pp. 1209-1220 (1990).
5) 中川路，勝山，宮内，水野：OSI 抽象構文記法支援ソフトウェア APRICOT の開発と評価，電子情報通信学会論文誌，Vol. J 73-D-I, No. 2, pp. 225-234 (1990).
6) Ohara, Y., Suganuma, T. and Senda, S.: ASN. 1 Tools for Semiautomatic Implementation of OSI Application Layer Protocols, *Proc. of The Second International Symposium on Interoperable Information Systems*, pp. 63-70 (1988).
7) ISO/IEC JTC1/SC21 N 6131: Working Draft for Light Weight Encoding Rules (1991).
8) 堀内，小花，鈴木：OSI 応用層プロトコル用 ASN. 1 ライトウェイト符号化規則のための符号化／復号処理系の実装と評価，情報処理学会マルチメディア通信と分散処研究会資料，DPS-55-2 (1992).
9) Huitema, C. and Doghri, A.: Defining Faster Transfer Syntaxes for the OSI Presentation Protocol, *ACM SIGCOMM Computer Communication Review*, Vol. 19, No. 5, pp. 44-55 (1989).

(平成4年11月9日受付)
(平成5年 4 月8日採録)



堀内 浩規（正会員）

昭和35年生．昭和58年名古屋大学工学部電気工学科卒業．昭和60年同大大学院情報工学専攻修士課程修了．同年国際電信電話(株)入社．現在，同社研究所通信網支援ソフトウェアグループ主査．この間，ネットワークアーキテクチャ，OSI プロトコルの実装方式，通信プロトコルの形式記述技法，ネットワーク管理の研究に従事．平成4年度電子情報通信学会学術奨励賞受賞．電子情報通信学会会員．

小花 貞夫（正会員）

昭和28年生．昭和51年慶応大学工学部電気工学科卒業．昭和53年同大学院修士課程修了．同年国際電信電話(株)入社．現在，同社研究所交換グループ主任研究員．工学博士．この間，パケット交換方式，ネットワークアーキテクチャ，OSI プロトコルの実装方式，データベース，国際ビデオテックス通信，分散処理技術，インテリジェントネットワークの研究に従事．電子情報通信学会会員．

鈴木 健二（正会員）

昭和20年生．昭和44年早稲田大学理工学部電気通信学科卒業．昭和44年から45年までオランダのフィリップス国際工科大学に招待留学．昭和51年早稲田大学大学院博士課程修了．同年国際電信電話(株)入社．現在，同社研究所 OSI 通信グループリーダ．工学博士．この間，磁気記録，パケット交換方式，ネットワークアーキテクチャ，分散処理の研究に従事．昭和62年度前島賞，平成4年度電子情報通信学会業績賞を各受賞．平成5年度より電気通信大学大学院情報システム学研究科客員教授．電子情報通信学会，IEEE 各会員．