

ASN.1 のための高能率圧縮符号化規則 (EPER) の提案と評価

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) の上位層や ISDN (統合サービスデジタル網) のユーザパート等におけるデータ要素は, ASN.1 (抽象構文記法 1) を用いた抽象構文として定義され, 符号化される. この ASN.1 の符号化/規則としては従来から基本符号化規則 (BER) が広く使用されているが, BER は値に対し常に値の長さを示すオクテット列 (LI) や値の型を識別するためのオクテット列 (ID) が付加される等の冗長性があるため, 符号化/復号処理時間の低下や符号化データ長の増大を招いていた. この BER の問題点を解決するため, BER 以外の符号化規則を使用して通信の効率化を図る試みが幾つか行われている. その中でも, 特に, ISO では BER より符号化データ長を短くして効率的な通信を可能とする観点から圧縮符号化規則 (PER) の標準化が進捗している. しかしながら PER は, 1) ビットのシフト演算を頻繁に行って符号化/復号処理時間を増大させる場合や, 2) オクテット境界とするためのパディングにより符号化データ長が長くなる場合があるという問題点等がある. 本論文では, これらの問題点を解決するため, ビット・データとオクテット・データを分離して符号化する等の特徴を持つ新たな高能率圧縮符号化規則 (EPER) を提案する. また, EPER を評価するため, PER および EPER の符号化/復号処理プログラムを抽象構文から自動生成するコンパイラを作成し, このコンパイラで生成した符号化/復号処理プログラムを用いて符号化/復号処理時間および符号化データ長を評価した. その結果, EPER は PER と比較して, 符号化と復号処理時間において, それぞれ, 1.2~3.0 倍, 1.2~5.7 倍高速化でき, 符号化データ長も PER の 41%~96% 程度に圧縮できることを実証した.

Efficient Packed Encoding Rules (EPER) for ASN.1 and Its Evaluation

HIROKI HORIUCHI,[†] SADA OOBANA[†] and KENJI SUZUKI[†]

Data elements in OSI (Open Systems Interconnection) upper layer protocols and user parts of ISDN (Integrated Services for Digital Network) are defined and encoded using ASN.1 (Abstract Syntax Notation One). Although BER (Basic Encoding Rules) were widely used as encoding rules for ASN.1, they have defects of decreasing encoding/decoding time and of increasing length of encoded data due to redundant identifier and length octets. The standardization of Packed Encoding Rules (PER) is under way to realize more efficient ASN.1 encoding/decoding than BER by means of minimizing the length of encoded data. However, PER have problems which cause decrease of encoding/decoding time due to many bit-shift operations and increase of data length due to paddings for octet-alignment. In order to resolve these problems, this paper proposed new encoding rules (EPER: Efficient Packed Encoding Rules) which treat bit aligned data and octet aligned data separately. Furthermore, we developed an ASN.1 compiler for EPER and PER which generates encoding/decoding programs from abstract syntax, in order to evaluate the EPER. We evaluated EPER using generated encoding/decoding programs from the viewpoint of encoding/decoding time and the length of encoded data, and ensured the effectiveness of EPER. The encoding/decoding time was 1.2~3.0/1.2~5.7 times less than that in PER, and the length of encoded data was reduced to 41%~96% of that in PER.

1. はじめに

OSI (開放型システム間相互接続) の上位層や ISDN (統合サービスデジタル網) のユーザパート等におけるデータ要素は, ASN.1 (抽象構文記法 1)¹⁾ を用いた抽象構文として定義され, 符号化される. この ASN.1 の符号化規則としては, 現在, 基本符号化規則

則 (BER: Basic Encoding Rules)²⁾ が広く使用されているが, BER は値に対し, 常に, 値の長さを示すオクテット列 (LI) と値の型を識別するためのオクテット列 (ID) が付加される等の冗長性を持つため, 符号化/復号処理時間の低下や符号化データ長の増大を招いている^{3), 4)}.

この BER の問題点を解決するため, ハードウェアを用いた並列処理により BER の高速化を行う試み⁵⁾と, BER 以外の新たな符号化規則を使用して効率化を図

[†] 国際電信電話株式会社 研究所
KDD R&D Laboratories

る試み^{6)~8)}が行われている。後者の新たな符号化規則としては、冗長性を排除した圧縮符号化規則 (PER: Packed Encoding Rules)⁶⁾、特定のアプリケーション向きの符号化規則⁷⁾、計算機の内部表現に基づく符号化規則 (LWER)⁸⁾ 等がある。中でも、特定のアプリケーションに依存せず、汎用的に符号化データ長を短くして効率的な通信を可能とするという観点から、ISO 等では PER⁶⁾ の標準化が進捗しており、注目されている。

しかしながら、PER は、1) ビットのシフト演算を頻繁に行って符号化/復号処理時間を増大させる場合や、2) オクテット境界とするためのパディングにより符号化データ長が長くなる場合があるという問題点がある^{9), 10)}。

本論文では、PER をベースに、上記問題点を解決する新たな高能率圧縮符号化規則: Efficient Packed Encoding Rules (EPER) を提案する。また、EPER を評価するため、PER および EPER の符号化/復号処理プログラムを抽象構文から自動生成するコンパイラを作成し、コンパイラにより生成した符号化/復号処理プログラムを用いて符号化/復号処理時間および符号化データ長の観点より PER、BER ならびに LWER と比較し、EPER の有効性を示す。以下、第2章で PER の概要、第3章で PER の問題点を述べる。第4章では、PER の問題点を解決する EPER の提案を行い、第5章では、EPER を符号化データ長と符号化/復号処理時間の観点から評価を行う。最後に、第6章で EPER の有効性等を考察する。

2. PER の概要⁶⁾

PER は、BER における冗長な識別子や長さ情報の排除等を行い、符号化データ長を極力短くする符号化規則であり、以下の特徴を持つ。

- ①型に対する識別子 (ID) は付与しない。
- ②値が可変長の場合のみ長さ (LI) を付加する。
- ③論理 (Boolean) 型、ビット列 (BitString) 型および列挙 (Enumerated) 型の符号化はビット単位 (以下、ビット・データと呼ぶ) で行う。
- ④順序 (Sequence) 型/集合 (Set) 型におけるオプションな要素の使用の有無、および、選択 (Choice) 型で選択された要素を示す情報はビット・データとして符号化する (それぞれ、プリアンブル (Preamble)、インデックス (Index) と呼ぶビット列)。
- ⑤集合 (Set) 型を構成する各要素は、タグの番号が小

さなもののから順番に符号化 (Canonical Order) する。

- ⑥整数 (Integer) 型の値の範囲、順序列 (Sequence Of) 型の繰り返し要素の数等に制約が付加される場合は、制約を利用して符号化データ長を圧縮する。
- ⑦転送構文には、1) ビット単位の符号化データにパディングを行って、オクテット境界とする転送構文 (Aligned) と、2) パディングを行わない転送構文 (Unaligned) の2種類がある。

図1に PER の符号化例を示す。同図(a)(b)では、抽象構文定義例および入力値例を示し、同図(c)は対応する符号化データ例を示す。同図(c)では、第1オクテットの4ビットでオプションな要素の使用を示す Preamble を符号化し、第4オクテットと第7オクテットの先頭ビットで Boolean 型の値を、その他のオクテットで Integer 型の値を符号化している。

```
A ::= SEQUENCE {
  a [1] INTEGER OPTIONAL, b [2] BOOLEAN OPTIONAL,
  c [3] INTEGER OPTIONAL, d [4] BOOLEAN OPTIONAL }
```

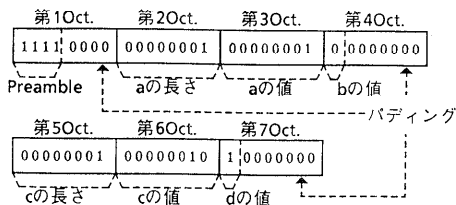
(a) 抽象構文定義例

(a) ASN.1 specification

{ a 1, b TRUE, c 2, d FALSE }

(b) 入力値例

(b) Input data value



(c) 符号化データ例 (転送構文 Aligned の場合)

(c) Encoding data in Aligned transfer syntax

図1 PER の符号化例

Fig. 1 An example of encoding data in PER.

3. PER の問題点

PER には以下の問題点がある。

- ① Unaligned の符号化/復号処理時間の増大
転送構文として Unaligned を使用した場合の符号化データは、型によりビット・データとオクテット単位のデータ (以下、オクテット・データと呼ぶ) が混在する。従って、ビット・データに後続するオクテット・データは、オクテット境界に揃わない場合が多くなる。このため、符号化/復号処理を行うプログラムでは、符号化データに対するシフトや論理積 (AND)

などのビット演算を頻繁に行う必要が生じ、処理速度が低下する。

② Aligned の符号化データ長の増加

通常的应用層のデータ要素では、Sequence 型におけるオプションな要素や Choice 型等がかなり含まれ、PER による符号化データには Preamble や Index 等のビット・データが多発する。Aligned を使用した場合の符号化データでは、ビット・データに対しパディングを行うため、ビット・データとなる要素を多く含む場合には、符号化データ長が増加する。例えば、図 1(c) の符号化データ例では Preamble および Boolean 型に対応する各オクテット (第 1, 4, 5 オクテット) のパディングは合計 18 ビットとなり、全符号化データ長 (56 ビット) の約 3 分の 1 となる。

③ Integer 型の符号化における冗長な LI

Integer 型の符号化では、常に値の長さを示すための LI がオクテット・データとして付加される。しかしながら、应用層プロトコルにおける ROS (Remote Operations) の操作値やエラーコード等は、通常小さな値が割当てられ、常に 1 オクテット以上の長さを持つ LI が付加されるのは冗長である。

④ 複数の転送構文の使い分けの煩雑さ

Aligned と Unaligned の 2 種類の転送構文が定義されているが、通信相手によって使い分けをすることを不要とするためには、単一の転送構文が望ましい。

4. 新しい符号化規則 (EPER) の提案

以下の方針に基づき、上記の PER の問題点等を解決する新たな符号化規則 (EPER: Efficient Packed Encoding Rules) を提案する。

- (1) シフト演算の頻発による処理速度の低下 (問題点①)、および、パディングによる符号化データ長の増大 (問題点②) を回避するため、ASN.1 の型に対応する符号化データがビット・データである要素とオクテット・データである要素を分離して設定する。
- (2) Integer 型の LI の符号化における冗長性 (問題点③) を排除するため、LI を常に 1 オクテット以上としない。
- (3) 通信相手によって転送構文を使い分けを不要とするため、単一の転送構文とする (問題点④)。

4.1 符号化データの構造

符号化データは、オフセット、ビットおよびオクテ

ットの 3 つのフィールドから構成する (図 2)。ビット・フィールドはビット・データを設定し、オクテット・フィールドはオクテット・データを設定する。また、オフセット・フィールドはビット・フィールドの長さを示し、抽象構文定義から長さを予め求められない場合にのみ付加する。

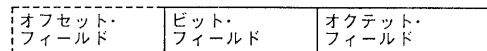


図 2 EPER における符号化の構造
Fig. 2 Structure of encoding data in EPER.

4.2 オフセット・フィールドの符号化

(1) 付加条件

符号化すべきデータの中にビット・フィールドを使用する型が存在し、さらに、ビット・フィールドの長さが値によって可変となる以下の場合にオフセット・フィールドを付加する。

- ① BitString 型の場合。
- ② Sequence 型および Set 型でオプションなビット・データの型を含む場合。
- ③ Choice 型で選択肢中にビット・データの型を含む場合。
- ④ Sequence Of 型および Set Of 型で繰り返し要素の型がビット・データとなる場合

(2) 符号化

オフセット・フィールドの符号化は、ビット・フィールドの長さにより以下の方法で行う。

① 長さが 1~7 ビット

第 1 オクテットの先頭ビットに値 '0_{CB}' を割当て、ビット・フィールドが第 1 オクテットのみにあることを示す。すなわち、オフセット・フィールドを 1 ビットで符号化し、残り 7 ビットにビット・フィールドを設定する。

② 長さが 0 または 1~63 オクテット

オフセットを 1 オクテットで符号化し、先頭 2 ビットに値 '10_{CB}' を設定する。残り 6 ビットでビット・フィールドの長さを表す値 (0~63) を設定する。

③ 長さが 64 オクテット以上

オフセット・フィールドを 2 オクテット以上で符号化する。第 1 オクテットの先頭 2 ビットに値 '11_{CB}' を、残り 6 ビットにオフセットの第 2 オクテット以降のオクテット数を設定する。第 2 オクテット以降は、ビット・フィールドの長さを設定する。

4.3 ビット・フィールドの使用

以下の場合には、ビット・データとしてビット・フ

表 1 各データ型の符号化

Table 1 Encoding rules for each type in EPER.

型	符号化規則	フィールドの使用	
		BF	OF
Boolean	BFに1ビットで符号化する。値はTrue(0)/False(1)をとる。LIは付加しない。	○	×
Enumerated	列挙値を昇順に並び換え、並び換えた列挙値に対し最小値を0として昇順に+1した値を割当てる。割当てられた列挙値の範囲(R)が1の時、符号化しない。2≤R≤128の時BFに、R≥129の時OFに値を設定する。LIは付加しない。	△	△
Null	符号化しない。但し、Set型/Sequence型の要素でOPやDFの時、BFのPreambleに有無を示す。	△	×
Integer	表2参照。	×	○
Real	LIと値からなり、OFに設定する。値はBERと同一の符号化方法をとる。	×	○
Octet String / Character String	LIと値からなり、OFに設定する。	×	○
BitString	値のビット長を示すLIと値からなる。LIはOFに設定する。値は、長さが8ビットの倍数でない剰余ビットをBFに設定し、その他はOFに設定する。	△	○
Object ID	LIと値からなり、OFに符号化する。値はBERと同一の符号化方法をとる。	×	○
Any	LIと値からなり、OFに符号化する。値は、別途ASN.1で定義されるデータ型の符号化に従う。	×	○
Sequence/Set	Preambleと1個以上の構成要素からなる。PreambleはOPとDFの要素の存在を表し、要素有(1)無(0)の値をBFに設定する。OPやDFが無い場合には、Preambleは符号化しない。構成要素は、各型に従って、OF, BFに符号化する。各構成要素の符号化順序は、Sequence型では抽象構文の定義順、Set型ではCanonical Orderとする。	△	△
Choice	Indexと選択要素からなる。Indexは選択された要素を識別するための番号を割り当て、選択枝数(S)が1個の時は符号化せず、Sが2~128個の時はBFに、Sが129個以上の時はOFに、符号化する。選択要素は、型に従ってOF, BFに符号化する。	△	△
Sequence Of / Set Of	要素の個数を示すNumber部と要素の型を符号化したComponent部からなる。Number部はOFに符号化され、Component部は使用する型に従って、OFまたはBFに符号化する。Number部はLIと同一の符号化規則により符号化する。	△	○

注) BF: ビット・フィールド, OF: オクテット・フィールド, OP: オプショナル, DF: デフォルト。

○: 使用, ×: 不使用, △: 場合によって使用。LI: Octet String型等の値の長さを示し, PERと同一の符号化とする。

フィールドに値を設定する。また、ビット・フィールドは、後続のオクテット・フィールドとオクテット境界となるように、必要に応じて'0_(B)'をパディングする。

- ① Boolean 型の値。
- ② Enumerated 型で値の範囲が2から128の場合。
- ③ BitString 型で値の長さが8ビットの倍数でない場合の剰余ビット。
- ④ Sequence 型/Set 型の Preamble。
- ⑤ Choice 型の Index で値の範囲が2から128の場合。

4.4 各データ型の符号化

各データ型の符号化の一覧、ビット・フィールドおよびオクテットフィールドの使用の有無を表1に示す。以下に主なデータ型の符号化を示す。

(1) Integer 型

表2に Integer 型の符号化を示す。符号化データは、値の長さ (Indicator) および値から構成され、オクテットフィールドに符号化する。Indicator は、値の範囲が $-2^5 \sim (2^5 - 1)$, $-2^{35} \sim (2^{35} - 1)$, $-2^{67} \sim (2^{67} - 1)$ の場合に、それぞれ、2, 4, 4 ビットで表現する。

この符号化では、以下の点に留意した。

- ① Indicator および値を少ないビット数で表現する。
- ② Indicator を多オクテットに振り分けない。
- ③ 1 オクテット内での値の表現範囲を狭くしないた

表 2 Integer 型の符号化

Table 2 Encoding rules for Integer type.

値の範囲	符号化規則
-25 ~ 25-1	1Oct.で符号化。長さとして、先頭2bitに値'00 _(B) 'を設定し、残り6bitに値を設定。
-235 ~ 235-1	2Oct.~5Oct.で符号化。長さを第1, 第2Oct.の先頭2ビットに割当て、それぞれ、値'01 _(B) '、後続するオクテット数(0~3)を設定。第1および第2Oct.の残り6bitと後続するOct.により値を設定。
-267 ~ 267-1	6Oct.~9Oct.で符号化。第1Oct.の先頭4bitに長さを割当て、2bitは値'10 _(B) 'を設定し、第3, 4bitは、第6Oct.以降の後続Oct.数を設定。第1Oct.の残り4bitと第2Oct.以降のOct.が値を示す。
-2324 ~ 2323	長さを10Oct.で符号化。長さの先頭3bitに値'110 _(B) 'を設定し、残り5ビットに第10Oct.以降の後続するオクテット数(1~32)を示す値(0~31)を設定。
上記以外	長さを20Oct.以上で符号化。長さの第1Oct.の先頭3bitに値'111 _(B) 'を設定し、残り5bitで長さオクテット数(1~32)を示す値(0~31)を設定。第2Oct.以降に、長さ、値を符号化。

め、Indicator の長さを短くする。

- ④ 64 ビットで表現できる整数の符号化を効率良く行うとともに、大きな値も表現可能とする。

(2) Sequence 型/Set 型

符号化データは、Preamble と1個以上の構成要素から構成する。Preamble は構造形を構成するオプショナルまたはデフォルトの構成要素の存在を表すビット・データの集合とし、各ビットには構成要素有り(1_(B))、または、構成要素無し(0_(B))の値を、ビット・フィールドに設定する。オプショナルやデフォルトの構成要素が無い場合にはPreambleは符号化しない。また、構成要素は、各型に従って、オクテット・フィー

ルド, ビット・フィールドに符号化を行う。

Set 型の符号化は, 上記の方法に加え, 各構成要素の符号化順を Canonical Order とする。

(3) Choice 型

符号化データは, 選択された要素を識別するための Index と選択要素から構成する。Index の値は, 定義された選択肢の要素を Canonical Order の順序として割当てた 0 以上の値を使用し, その符号化は選択肢の数により以下の方法で行う。また, 選択された要素は, 型に従ってオクテット・フィールド, ビット・フィールドに符号化する。

- ① 1 個: 符号化しない。
- ② 2~128 個: ビット・フィールドに, (選択肢の数 - 1) を表現するのに必要なビット数で符号化する。
- ③ 129 個以上: オクテット・フィールドに LI と同様な方法で符号化する。

4.5 符号化データ例

図 1 (a)(b)の例に対する EPER の符号化データを図 3 に示す。図 1 (a)の抽象構文は, 4.2 節(1)で示したオフセット・フィールドの付加条件②に該当し, 4.2 節(2)の①を適用する。すなわち, 第 1 オクテットの先頭ビットにオフセット・フィールドを割当て, Preamble および Boolean 型の値は, 第 1 オクテットのビット・フィールドに設定する。また, Integer 型の値は第 2, 第 3 オクテットのオクテット・フィールドに設定する。

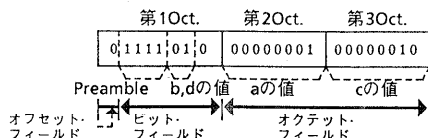


図 3 EPER の符号化データ例
Fig. 3 An example of encoding data in EPER.

5. EPER の評価

EPER を評価するため, PER および EPER の符号化/復号処理プログラムを抽象構文から自動生成するコンパイラを作成し, コンパイラから生成した符号化/復号処理プログラムを用いた符号化/復号処理時間および符号化データ長の測定結果を以下に述べる。

5.1 PER および EPER のためのコンパイラの実装

本コンパイラは, 抽象構文定義から, ASN.1 に対

応した C 言語の型定義, および, PER および EPER の符号化/復号関数を自動生成する (図 4)。

5.1.1 生成される C 言語の型定義

図 5 に, 入力抽象構文例と生成される C 言語の型定義例を示す。ここでは, 各 ASN.1 定義に対応する構造体 struct Person, struct Name, struct Children が生成されている。

5.1.2 EPER の符号化/復号関数の処理

生成される符号化関数は, 生成された C 言語の型定義を持つ変数に設定した値を用いて符号化を行う。EPER の符号化関数の処理は以下となる。①データ要素を作成するのに必要となる領域の計算を行う。この際, ビット・フィールドとオクテット・フィールドの長さの計算も行う。また, 4.3 節の付加条件に該当し, オフセット・フィールドが必要な場合は, ビット・フィールドの長さを符号化データの先頭に設定する。②変数に設定した値をトレースして, 型に応じて, ビット・フィールドとオクテット・フィールドに符号化を行う。図 6 に符号化関数で使用するマクロ, および, 図 5 (a)の抽象構文の入力に対して, 型 Person の符号化関数の生成例を示す。ここでは, 型 Flist の変数に符号化データが設定され, ビット・データおよびオクテット・データは, それぞれ, bfield, ofield のフィールドに設定し, bitpos はビット・フィールドの符号化位置を示す。

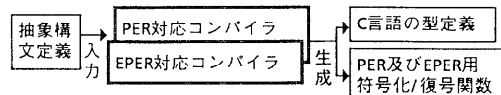


図 4 PER および EPER 対応 ASN.1 コンパイラの構成

Fig. 4 Software structure of ASN.1 compiler for PER and EPER.

```
Person ::= SEQUENCE { number INTEGER, name Name,
    maleOrFemale BOOLEAN, age INTEGER,
    single BOOLEAN, children Children OPTIONAL }
Name ::= SEQUENCE { first IA5String, last IA5String }
Children ::= SEQUENCE OF Name
```

(a) 入力抽象構文

(a) Input ASN.1 specification

```
struct Name { struct field first; struct field last; };
struct Children { int number; struct Name *name; };
struct Person { int number; struct Name name;
    int maleOrFemale; int age;
    int single; struct Children *children; };

```

(b) 生成する C 言語の型定義

(b) Generated type definition in C language

図 5 入力抽象構文と生成する構造体

Fig. 5 Examples of input ASN.1 specification and generated type definition in C language.

EPER の復号関数の処理では、符号化処理と逆に、符号化データを、抽象構文定義の型定義の順に、EPER の型毎の符号化に従ってビット・フィールドとオクテット・フィールドをトレースし、復号結果を生成された C 言語の型定義を持つ変数に、順次設定する。

5.1.3 PER の符号化／復号関数の処理

PER の符号化処理では、変数に設定された値を、各型の符号化に従って、符号化データの先頭から順次書き込む。この際、Aligned 転送構文の時はビット・データ書き込み時のパディング、Unaligned 転送構文の時はオクテット境界がずれた際のシフト演算等を行う。また、PER の復号処理では、符号化処理の逆に符号化データから変数の値へ設定を行う。

```
typedef struct { unsigned char *data; int bitpos;
  int size; } Bfield;
typedef struct { Bfield bfield; String ofield; ERROR *err;
  int asize; } Flist;

#define SetNextBitPos( Bfield )\
{ if( Bfield.bitpos ) Bfield.bitpos --;\
  else { Bfield.bitpos = INITPOS; Bfield.data ++; } }
#define SetBit( Bfield )\
( (*Bfield.data) |= (PATTERN_SETBIT << Bfield.bitpos) )
#define Set1Byte( Ofield, Data )\
{ (*Ofield.str) = ((UCHAR)Data); Ofield.str ++; }
#define SetBoolean( Flist, Elem )\
{ if( Elem ) SetBit( Flist.bfield );\
  else ResetBit( Flist.bfield ); SetNextBitPos( Flist.bfield ); }
```

(a) 符号化で使用するマクロ

(a) Macro functions for encoding

```
int mEPerson_per(elem, flist)
struct Person *elem; Flist *flist;
{
  Bfield preamble;

  preamble.data = flist->bfield.data;
  preamble.bitpos = flist->bfield.bitpos;
  SetNextBitPos( flist->bfield, 1 );
  SetInteger( (*flist), (elem->number) );
  if( (mEName_per(&(elem->name), flist)) == FALSE )
    return FALSE;
  SetBoolean( (*flist), (elem->maleOrFemale) );
  SetInteger( (*flist), (elem->age) );
  SetBoolean( (*flist), (elem->single) );
  if( elem->children ) { SetBit( preamble );
    if( (mEChildren_per( elem->children, flist )) == FALSE )
      return FALSE; }
  else { ResetBit( preamble ); }
  SetNextBitPos( preamble );
  return TRUE;
}
```

(b) 生成された符号化関数

(b) Generated encoding functions

図 6 EPER の符号化関数

Fig. 6 An example of encoding functions for EPER.

5.2 評価実験

上述のコンパイラにより生成した、PER の問題点等に関連する抽象構文、および、実際の応用層プロトコルの抽象構文に対する符号化／復号関数を用いて、PER と EPER の比較評価を行った。また、あわせて、BER²⁾、¹¹⁾ 使用時の測定も行った。測定は NEWS 3860 (CPU: R 3000) を使用し、符号化／復号処理時間は

1,000 回ループさせた平均をとった。

5.2.1 PER の問題点に関する抽象構文の場合

実験に使用した抽象構文定義を図 7 に示す。抽象構文 Test-1 は、3 章で示した問題点①、②、③に関連し、Test-2 は問題点①、②に関連する。符号化／復号処理時間および符号化データ長を表 3 に示す。

```
Test-1 ::= SEQUENCE OF SEQUENCE { a BOOLEAN, b
  INTEGER }
Test-2 ::= SEQUENCE OF SEQUENCE { a BOOLEAN, b
  OctetString }
```

図 7 評価実験に使用した抽象構文

Fig. 7 ASN.1 specification used for evaluation.

表 3 評価実験結果

Table 3 Results of experiments.

抽象構文定義	符号化規則	符号化時間(msec)	復号時間(msec)	データ長(oct.)
Test-1	EPER	0.045	0.035	25
	PER-Aligned	0.078	0.060	61
	PER-Unaligned	0.084	0.062	44
	BER	0.340	0.380	163
Test-2	EPER	0.106	0.039	385
	PER-Aligned	0.129	0.054	401
	PER-Unaligned	0.318	0.223	384
	BER	0.345	0.436	504

注) Test-1 及び Test-2 の Sequence Of の要素は 20 個。

5.2.2 応用層プロトコルに関する抽象構文の場合

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構造は以下の 3 種類に大別できる¹²⁾。

「構造 A」 長さの短い基本形を要素とする構造形で、その要素数が多い場合。

「構造 B」 構造形に含まれる特定の基本形の長さが大きな場合。

「構造 C」 長さの短い基本形を要素とする構造形で、その要素数が少ない場合。

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／復号処理時間および符号化データ長を測定した結果を表 4 に示す。ここでは、構造 A、B、C の例として、それぞれ、CMIP の m-Get 操作結果、84 年版 MHS の IM-UAPDU、および、CMIP の m-Get 操作要求を用いた。なお、PER および EPER における応用層プロトコルごとのビット・データは、それぞれ、54 ビット、95 ビット、17 ビットであり、Integer 型の数は、それぞれ、50 個、2 個、5 個であった。

6. 考察

6.1 符号化／復号処理時間および符号化データ長

(1) PER の問題点に関する抽象構文の評価実験よ

表 4 実際の応用層プロトコルにおける評価実験
Table 4 Results of experiments in actual OSI
application layer protocol data units.

構造	抽象構文定義	符号化規則	符号化時間(msec)	復号時間(msec)	データ長(oct.)
A	GetResult	EPER	0.310	0.170	509
		PER-Aligned	0.430	0.330	602
		PER-Unaligned	0.670	0.560	558
		BER	0.924	1.356	758
B	IM- UAPDU	EPER	0.200	0.160	429
		PER-Aligned	0.290	0.200	438
		PER-Unaligned	0.570	0.450	430
		BER	0.400	0.730	481
C	GetArgument	EPER	0.070	0.060	124
		PER-Aligned	0.100	0.070	134
		PER-Unaligned	0.120	0.090	128
		BER	0.340	0.400	160

注)・GetResultは、50個の属性値の検索結果。

・IM-UAPDUは、200オクテットの文字列を含むボディパートを持つ。

り、EPER の符号化と復号処理時間は、PER の Aligned の場合より、それぞれ、1.2~1.7 倍、1.4~1.7 倍に高速となった。また、Unaligned の場合より、それぞれ、1.9~3.0 倍、1.7~5.7 倍高速となった。EPER の符号化データ長は、PER の Aligned の 41%~96% となった。さらに、実際の応用層プロトコルを使用した評価実験においても、PER の符号化と復号処理時間より、それぞれ、1.4~2.9 倍、1.2~3.3 倍高速となり、符号化データ長は PER の Aligned の場合の 84~91% 程度に短縮しており、EPER の有効性を確認できた。

EPER は、符号化データ中にビット・データや Integer 型の要素が無い場合に、PER と同程度の符号化/復号処理時間および符号化データ長となると予想されるが、実際の応用層プロトコルでは、これらの要素が含まれることが多く、ほとんどの場合に EPER が有効となると言える。

BER との比較では、EPER の符号化と復号処理時間は、それぞれ、3.0~7.6 倍、3.3~11.0 倍に高速となり、符号化データ長は 15%~77% 程度に短縮され大幅な性能向上となる。

(2) EPER における Integer 型の符号化/復号処理においては、Indicator を使用するため、 $-2^5 \sim 2^5 - 1$ 以外の値の際にはシフト演算が発生する場合がある。しかしながら、この場合でも符号化結果はオクテット・データとなり、3章の①の問題点が発生するシフト演算とはならない。また、EPER の Integer 型の符号化/復号処理では、1)符号化データ長を極力短くする方法を採用したため、多くの場合 PER の符号化データ

長より短くなること、2) PER では LI と値を別個のオクテットとしているのに対して、EPER では Indicator と値を同一のオクテットで表しているため、符号化時のバッファへの書き込み、復号時の変数への設定や LI のエラーチェックが簡略化されることにより、PER の Integer 型の符号化/復号より符号化/復号処理時間が増大することは無いと考えられる。実際に、10 個の Integer 型を構成要素とする Sequence 型の抽象構文を用いて、EPER と PER の符号化/復号処理時間を測定した結果、Integer 型の値に $-2^5 \sim 2^5 - 1$ 以外の任意の値を設定した場合でも EPER は PER より符号化で約 1.3 倍、復号で約 1.6 倍高速となった。

6.2 EPER の符号化/復号処理

EPER は、オフセット・フィールドが必要か否かの条件判定を行うが、これは値とは独立に、抽象構文定義から判定可能であるので、符号化/復号処理時間には影響しない。また、PER の符号化処理は符号化データの先頭から順次書き込む方式をとるのに対し、EPER の符号化処理ではビット・データを一括してオクテット・データと分離して配置するので、ビット・フィールドとオクテット・フィールドへ振り分けて値を設定するという大きな差があるが、EPER および PER コンパイラの生成する符号化/復号処理プログラムの複雑さは同程度であった。また、EPER および PER コンパイラを使用した場合の CMIP の符号化/復号処理プログラムの規模は、いずれも 6.1k ステップとなった。

6.3 EPER と LWER⁹⁾ との比較

高速化を主目的としたライトウェイト符号化規則 (LWER) との比較のため、実験で使用した応用層プロトコルの抽象構文定義に対する符号化/復号処理時間および符号化データ長を測定した (表 5)。ここでは、文献 12) のコンパイラで生成した符号化/復号処理プログラムを使用した。

表 5 LWER の評価実験

Table 5 Results of experiments in actual OSI application layer protocol data units using LWER.

抽象構文定義	転送構文	符号化時間(msec)	復号時間(msec)	データ長(oct.)
GetResult	32bit-Big Endian	0.370	0.060	1,424
	32bit-Little Endian	0.700	0.280	
IM- UAPDU	32bit-Big Endian	0.290	0.040	1,124
	32bit-Little Endian	0.540	0.190	
GetArgument	32bit-Big Endian	0.100	0.020	356
	32bit-Little Endian	0.190	0.080	

測定結果より, EPER は, LWER における同一のワード長/バイト順序の転送構文 (32 bit, Big Endian) 使用時の復号処理時間よりはわずかに遅くなるものの, 符号化処理時間, および, LWER におけるそれ以外の転送構文 (32 bit, Little Endian 等) 使用時の符号化/復号処理時間より高速となり, 符号化データ長も LWER の約 3 分の 1 となった。さらに, LWER はワード長が 16/32/64 ビットの 3 種, バイト順序が Big Endian か Little Endian かの 2 種の組み合わせによる計 6 種類の転送構文が定義されているのに対し, EPER は単一の転送構文で済むため, 転送構文の使い分けの煩雑さがなくとも, ソフトウェア量が少なく済むという利点がある。

7. おわりに

本論文では, 従来の ASN.1 圧縮符号化規則 (PER) において, ビットのシフト演算を頻繁に行って符号化/復号処理時間を増大させる場合や, オクテット境界とするためのパディングにより符号化データ長が長くなる場合があるという問題点を解決するために, 新たな符号化規則として高効率圧縮符号化規則 (EPER) を提案した。EPER は, ビット・データとオクテット・データを分離した符号化規則, Integer の符号化方法の変更, 転送構文の単一化といった特徴を持つ。

評価の結果, PER との比較では, EPER は符号化と復号処理時間において, それぞれ, 1.2~3.0 倍, 1.2~5.7 倍高速となり, 符号化データ長は PER の Aligned の転送構文の 41%~96% 程度に短縮できることを示した。また, 高速化を主目的とした LWER との比較においても, 通信相手が同一のワード長/バイト順序の計算機であるという特定の場を除き, EPER は LWER よりも高速化を達成していることから, EPER は広範囲で有効な符号化規則であると言える。今後は, 1 層から 7 層までのプロトコル処理全体の性能に対する EPER の効果の評価を行う予定である。

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 次長に感謝します。

参考文献

- 1) ISO/IEC 8824: Specification of Abstract Syntax Notation One (ASN.1) (1989).
- 2) ISO/IEC 8825: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1) (1989).

- 3) Rose, M. T.: *The Open Book: A Practical Perspective on OSI*, Prentice-Hall, Inc. (1990).
- 4) 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).
- 5) Bilgic, M. and Sarikaya, B.: Performance Comparison of ASN.1 Encoder/Decoders Using FTAM, *Computer Communications*, Vol. 16, No. 4, pp. 229-240 (1993).
- 6) ISO/IEC DIS 8825-2: Specification of Basic Encoding Rules for ASN.1—Part 2: Packed Encoding Rules (1994).
- 7) Cardoso, A. and Tovar, E.: Defining More Efficient Transfer Syntax for Application Layer PDUs in Field Bus Applications, *Sigcomm Computer Communication Review*, Vol. 22, No. 3, pp. 98-105 (1992).
- 8) ISO/IEC JTC 1/SC 21 N 6131: LightWeight Encoding Rules (1991).
- 9) 堀内, 小花, 鈴木: ASN.1 における圧縮符号化規則 (PER) の改善提案, 第 48 回情報処理学会全国大会論文集, 2D-4 (1994).
- 10) 堀内, 小花, 鈴木: ASN.1 のための高効率圧縮符号化規則 (EPER) の提案と評価, 情報処理学会マルチメディア通信と分散処理研究会, DSP-66-25 (1994).
- 11) Hasegawa, T., Nomura, S. and Kato, T.: Implementation and Evaluation of ASN.1 Compiler, *Information Processing in Japan*, Vol. 15, No. 2, pp. 157-167 (1992).
- 12) 堀内, 小花, 鈴木: 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, 情報処理学会論文誌, Vol. 34, No. 6, pp. 1325-1335 (1993).

(平成 6 年 8 月 1 日受付)

(平成 6 年 12 月 5 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, 同社研究所網管理グループ主

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 年慶応義塾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、同社研究所網管理グループリーダー。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、データベース、国際ビデオテキスト通信、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。工学博士。同年国際電信電話(株)入社。現在、同社研究所高速通信グループリーダー。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、高速・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所客員教授。電子情報通信学会、IEEE 各会員。