ソフトウェア工学　73−18
（1990．7．20）

入出力データの構造不一致の分類を考慮した ブートストラッピングによるコンパイラの実現法

橋本正明　　岡本克己
(株) ATR 通信システム研究所

JSP の主テーマである構造不一致を自動的に検出して解決するための実験用コンパイラを, 自己記述によるブートストラッピングで実現する方法を述べる. 構造不一致は規模の大きい問題であり, 部分問題へ分類されるので, 筆者らは構造不一致の検出・解決法を段階的に解明してコンパイラを拡張しながら検証している. そこで, ブートストラッピングの初期段階では, 構造不一致の最も基本的な性質を持っている脈絡不一致について実現し, 順序不一致はコンパイラの実現に不可欠であるがソート・プログラムの使用で回避した. 推移閉包による構造不一致も不可欠なため, ブートストラッピングの核になるカーネル・コンパイラはデータ・ドリブン方式で制御されるプログラムを生成した.

Compiler Implementation by Boot-strapping based on Classification of Structure Clashes between Input and Output Data

Masaaki HASHIMOTO and Katsumi OKAMOTO
ATR Communication Systems Research Laboratories
Sanpeidani, Inuidani, Seika-cho, Soraku-gun, Kyoto 619-02, Japan

This paper describes a experimental compiler implementation by self-application and boot-strapping. The compiler detects and solves structure clash which is one of the main concerns in JSP. Since structure clash is classified into several sub-problems, the authors have been studying the structure clash detection and solution method by steps, and have been proving the method by extending the compiler. In the first step of boot-strapping, multi-threading clash which has the fundamental characteristics of structure clash has been implemented, and ordering clash has been avoided by the use of sort program. Since structure clash by transitive closure has been inevitable, the kernel compiler of boot-strapping has generated programs to be controlled by data-driven method.

1 はじめに

筆者らは**ERモデル**[1]に基づいて対象世界を記述した仕様からソフトウェアを自動作成する研究を進めており，その一環としてプログラム仕様記述言語**PSDL** (Program Specification Description Language)を規定するとともに[2, 3, 4]，PSDLで記述されたプログラム仕様からCプログラムを生成するための実験用PSDLコンパイラを作成した[5, 6, 7, 8]. このコンパイラは,JSPの主テーマである**構造不一致**[9]を自動的に検出して解決することを目的としているが，構造不一致は規模の大きな問題であり，部分問題へ分類されるので，構造不一致検出・解決法は段階的に解明してコンパイラを拡張しながら検証している．

ところで，**自己記述**による言語処理系の作成は，*1)* 既存言語より使いやすい新言語でコンパイラを作成できるので，コンパイラの作成や拡張が容易，*2)* 新言語のデバグを兼ねることができる，*3)* 新言語の評価データを得ることができる，*4)* 新言語の実行効率向上を図れば，コンパイラ自体の実行効率も向上，等の利点を備えている．このため，自己記述の適用例は多く,LISPや,SmallTalk,REFINE等があげられる[10]. 筆者らのコンパイラは初期段階から作成しているため，**ブートストラッピング**の核となる**カーネル・コンパイラ**が別途必要になるが，コンパイラを頻繁に拡張するので，上記の利点を生かせるものと考えられる．

本稿では構造不一致の部分問題の分類に従って，自己記述によるブートストラッピングでコンパイラを実現する際の工夫点について述べる．以下，第2章でPSDLを概説して，第3章で構造不一致について検出・解決法も含めてを説明する．第4章ではコンパイラのブートストラッピングについて述べる．第5章では実験について述べ，第6章で考察する．

2 PSDL

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

情報層) 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，その枠組を定める．

データ層) 入出力データの形式を定める．

アクセス層) 入出力ファイルのアクセス方法を定める．

以下,PSDLコンパイラ中の小さなプログラムの仕様を記述した図1と，その仕様を図示した図2を引用してPSDL文を概説する．このプログラムは有向グラフの節点と有向枝の情報を入力して，節点毎に流入枝と流出枝の本数を計算して出力する機能を持っている．

2.1 情報層

ERモデルへ，導出関係を表す制約を付記して情報層の仕様を記述する．情報層は，図1では00行目のINFORMATION文と27行目のDATA文の間に記述し，図2では上半分の点線の四角形内に図示している．

(1) 実体型，属性，主キー，実体数

実体としては個々の節点や有向枝が現れ，それらの実体の集合である実体型VertexやArcを太線の四角形で図2に示す．実体型の属性はその四角形の下に示し，主キーの属性には下線をつけている．図1では各々の実体型を01行目のE (Entity type)文で記述し，属性を02行目のA (Attribute)文で記述する．主キー属性には03行目のK (Key)文をつける． 02行目のNUM (NUMber)は属性値の定義域が数値であることを示す．なお，文字列はSTR (STRing)で示す．実体型に含まれる実体の個数は08行目のEN (Entity Number)文で記述し，その文中の“-100”は個数が最大100であることを示している．

(2) 関連型，実体型の対応づけ，関連数

関連としては有向枝とその始節点との対応づけや，有向枝とその終節点との対応づけが現れ，それらの関連の集合である関連型VertexInflowArcやVertexOutflowArcを太線の菱形で図2に示している．

図1では各々の関連型を15行目のR (Relationship type)文で記述し，その関連型で対応づけられた実体型を16行目と18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.EntityTypeの形で指定する．もし，それらの実体型が相互に異なれば，図1のように役割を省略してよい．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17行目のRN (Relationship Number)文で記述する．この文中の“-10”は個数が最大10であることを示す． 19行目の“1”は1個であることを示す．一方，個数が不定ならば,M (Many)と記述する．

(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のに用いる．図2の一方向の矢印で示すように実体Vertexの属性InflowArcNumberの値は，その実体へ関連VertexOutflowArc

```
00 INFORMATION
01   E Vertex
02     A Id                 NUM
03     K
04     A InflowArcNumber   NUM
05       = FRelNumberNum(.VertexOutflowArc..Arc.Id)
06     A OutflowArcNumber NUM
07       = FRelNumberNum(.VertexInflowArc..Arc.Id)
08     EN -100
09   E Arc
10     A Id               NUM
11     K
12     A StartingVertex NUM
13     A TerminalVertex NUM
14     EN -500
15   R VertexInflowArc
16     C .Vertex
17       RN -10
18     C .Arc
19       RN 1
20     RC PEqualNum(.Arc.StartingVertex,.Vertex.Id)
21   R VertexOutflowArc
22     C .Vertex
23       RN -10
24     C .Arc
25       RN 1
26     RC PEqualNum(.Arc.TerminalVertex,.Vertex.Id)
27 DATA
28   I VertexData
29       IX VertexIndex
30     G VertexRecord ON EndOfFile(VertexData)
31     O VertexRecord
32       %5d     VertexNo
33         = Vertex.Id
34       %50s    Filler
35       %POINT VertexCR
36         = "CR"
37   I NewVertexData
38       IX NewVertexIndex
39     G NewVertexRecord ON PEntityNumber(Vertex)
40     O NewVertexRecord
41       %5d     VertexNo
42         = Vertex.Id
43       %29s    Filler1
44       %5d     InflowArcNumber
45         = Vertex.InflowArcNumber
46       %5d     OutflowArcNumber
47         = Vertex.OutflowArcNumber
48       %11s    Filler2
49       %POINT VertexCR
50         = "CR"
51   I ArcData
52       IX ArcIndex
53     G ArcRecord ON EndOfFile(ArcData)
54     O ArcRecord
55       %5d     ArcNo
56         = Arc.Id
57       %5d     StartingVertexNo
58         = Arc.StartingVertex
59       %5d     TerminalVertexNo
60         = Arc.TerminalVertex
61       %16s    Filler
62       %POINT ArcCR
63         = "CR"
64 ACCESS
65   D ArcFile    INPUT  27 ArcData
66   D VertexFile INPUT  56 VertexData
67   D VertexFile OUTPUT 56 NewVertexData
```

図 1: PSDL プログラム仕様

で対応づけられた実体 Arc の個数を数えて得られる. また, 属性 OutflowArcNumber の値は, その実体へ関連 VertexInflowArc で対応づけられた実体 Arc の個数を数えて得られる. 図 1では, 値が得られる属性の A 文に続けて,05 行目の = (equal) 文で制約を記述する. なお,FRelNumberNum は実体の個数を数えるための関数である.

この制約から参照される属性は role1.relationship Type. role2.entityType.attribute または attribute の形で記述する. ここで前者は, 値が得られる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用い, 後者は, 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る. なお, 後者のみが現れる制約を記述してもよい. また, 前者を用いる場合,1 つの制約に 1 つの関連型しか記述できず, 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.

(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. 図 2の両方向の矢印で示すように関連 VertexInflowArc は, 属性 StartingVertex と Id の値が等しい実体 Arc と Vertex の間で得られる. また, 関連 VertexOutflowArc は, 属性 TerminalVertex と Id の値が等しい実体 Arc と Vertex の間で得られる. 図 1では, 関連型の R 文と C 文に続けて,20 行目の RC (Relationship existence Condition) 文で制約を記述する. なお,2 つのパラメータ値が等しければ真となる述語 PEqualNum は関連存在条件を表しており, この条件が成り立つ場合にのみ関連が得られる. また, 条件式から参照される属性は role.entityType.attribute の形で記述している.

(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. この例は図 1に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. この場合, 得られた実体の主キー属性値を決めなければならないので, 実体が得られる実体型の K 文に続けて, 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 "= FEqualStr(.demand..customer.name) ON PPositiveNum(.demand..customer.total)" 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, 実

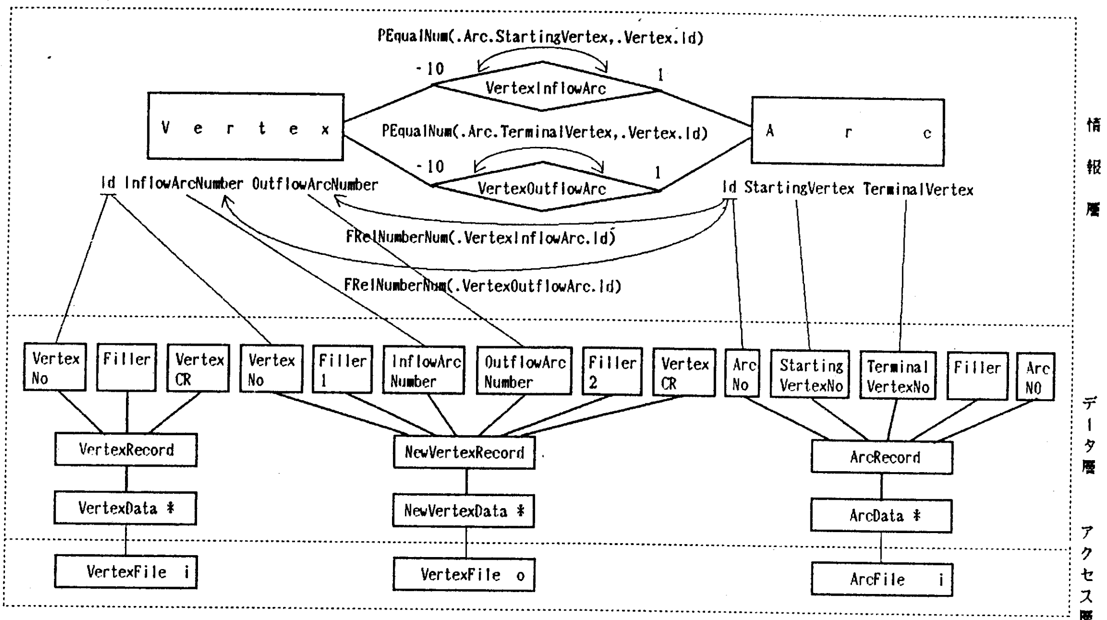


図 2: PSDL プログラム仕様図

体存在条件 PPositiveNum 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

(1) データ層

データ層は, 図 1では 27 行目の DATA 文と 64 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, O (sequence Order) 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を C 言語と同様に定める. たとえば,%5d は 5 桁の数値を定めている. 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. 繰返し集団データ型には繰返し終了条件が必要であり,30 行目の ON 句は End Of File 条件を示し,39 行目の ON 句は実体 Vertex 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 33 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 45 行目の = 文で基本データ型と結合する. 関連については図 1に例はないが, 関連で対応づけられた実体型の主キー属性と基本データ型とを,“= RelationshipType.Role.EntityType.Attribute” の文で結合する. 上記の結合をとることを, 情報層とデータ層の結合制約と呼ぶ.

(2) アクセス層

図 1の 64 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 65 行目の D (DatasetType) 文で記述する. この文にはファイル名や,INPUT と OUTPUT 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 65 行目の ArcData のようにデータ型を指定する. なお, ファイルは順アクセスされる.

3 構造不一致

プログラムの入出力データの構造不一致は, プログラムで同時に処理される入出力データの間で入出力タイミングが同期していない問題として定義されている [9]. 以下, 種々の観点から構造不一致について述べる.

3.1 構造不一致と非手続き型言語の関係

構造不一致に着目すると非手続き型言語は以下の3つのクラスに分けられる．まず，**クラス1**の言語の利用者は手続き型言語を使う場合と同じように構造不一致を検出して解決しなければならない．一方，**クラス2**と3の言語の利用者は構造不一致を意識しなくてもよい．その中でクラス2の言語のコンパイラは構造不一致の検出は行わず，すべてのデータに構造不一致が存在するものと仮定してプログラムを生成する．一方，クラス3の言語のコンパイラは構造不一致を検出して解決のうえ，プログラムを生成する．そこで,PSDLはクラス3の言語として設計している．

3.2 構造不一致の統一的な見方

文献[9]では構造不一致が脈絡不一致と順序不一致，境界不一致に分けられ，個々に例示的に説明されている．一方，自動化のための構造不一致検出・解決法を解明するには，構造不一致の統一的な見方が必要である．ところで，構造不一致における同期性は以下の3つの要素に依存している．

まず，同期性のタイミングを担う対象として，情報層では実体や属性値，関連があげられ，データ層ではデータがあげられる．これらの対象は実体型や属性，関連型，データ型毎に集合をなしており，コンパイラで生成された手続き型プログラムを実行させると，集合中の対象間に到着の全順序ができる．ところで対象相互の間には，導出関係を表した制約によって関係がとられている．その関係が，前述の全順序を保存する同型または準同型の写像になっていれば，構造不一致は存在しない．その他の場合は構造不一致が存在する．なお，写像は値を1つしか持たない．

3.3 構造不一致の分類

すでに判明している構造不一致[9, 6]を前述の統一的な見方から解釈しながら，構造不一致の分類を試みる．

(1) 境界不一致

1つのレコードがブロックをまたがっているような境界不一致では対象間の関係が全順序を保存しているが，レコードとブロックの関係がn対mになってしまうように関係が写像にはなっていない．

(2) 順序不一致

出力帳票のソーティングのようなデータ間の順序によって生じうる順序不一致では対象間に1対1の写像が存在している．しかし，その写像は順序を保存していない．順序不一致は同じ型のデータ間の順序に関するものと，データ型間の順序に関するものに細分できる．

(3) 脈絡不一致

上記の2種の構造不一致は関係の形または順序の保存性のみで定義できるが，脈絡不一致はその他の構造不一致を指しており，頻繁に生起するものと考えられる．

(4) 推移閉包による構造不一致

PSDLプログラム仕様中に推移閉包(一方向の閉路)[11]が存在する場合，対象のつながりは同じ集合を何回も通る螺旋を描くものと解釈している．この螺旋の形によって構造不一致が生起しうる．

まず螺旋が分流も合流もせず，対象間の関係が同型写像になっていれば，構造不一致は存在しない．一方，その他の場合は構造不一致が生起するが，特に螺旋が分流する場合は計算の実行の可能な箇所が何箇所にも増えるので，計算可能箇所をフラグで示すデータドリブン方式のプログラム制御が必要になる．また，螺旋が合流する場合は再計算のためにバックトラッキング方式のプログラム制御が必要になることもある．ただし，バックトラッキング方式が必要となる仕様は記述しないように制限している．

(5) 階層的構造不一致

前述の集合が部分集合に分割され，各々の部分集合の中でのみ構造不一致が生起し，部分集合のクラスの中では構造不一致が生じないことがある．逆に，各々の部分集合の中では構造不一致が生起せず，部分集合のクラスの中で構造不一致が生じることがある．

(6) 複合的構造不一致

上記の階層的構造不一致において，部分集合の中で生起する構造不一致の種類と，部分集合のクラスの中で生起する構造不一致の種類が異なることがある．

3.4 構造不一致の検出・解決法

現在解明ずみの構造不一致検出・解決法は脈絡不一致を対象にしており，仕様に推移閉包は含まれないことや，1つの入出力レコードは同じ型の実体や関連を高々1つしか表さないこと，入出力レコードに対応したデータ型はファイル毎に1つしかないこと等の制限条件を設けている．

3.4.1 構造不一致の検出方法

仕様から有向グラフを作成して,それを局所的かつ大局的に解析して構造不一致を検出する.

(1) 有向グラフの作成

PSDL プログラム仕様から節点と有向枝を抽出して有向グラフを作成する.有向枝は同期型か非同期型かに区分され,以下の解析によって非同期型有向枝が検出される.なお,非同期型有向枝の箇所に構造不一致が存在する.

(2) 局所的な解析

流入有向枝を2本以上持っている実体型節点や関連型節点について,その流入有向枝を非同期型とする.同時に,その実体型の非主キー属性節点の流入有向枝も非同期型にする.また,制約節点につながった有向枝について,その枝に対応したRN文が2以上の数値を持っていれば,その有向枝を非同期型とする.なお,関連存在従属性制約の節点につながった有向枝については,仮想的にRN文が2以上の数値を持っているものとする.

(3) 大局的な解析

有向グラフの中で方向混在閉路を見つけて,閉路上の非同期型有向枝がすべて同一方向を向いていれば,逆方向の有向枝を少なくとも1つ,非同期型へ変える.

3.4.2 構造不一致の解決方法

検出された構造不一致はプログラムの構造を以下のように決めて解決する.

(1) データ構造

非同期型有向枝がつながっている実体型節点や非主キー属性節点,関連型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.そこで,非同期型節点にはテーブルを割り当て,同期型節点にはスカラ変数を割り当てる.

(2) 手続き構造

有向グラフを以下の方法で連結サブグラフに分割する.まず,非同期型節点はすべてサブグラフの境界点として,同期型節点はすべてサブグラフの内部点とする.節点を介してつながった同期型有向枝どうしは同じサブグラフに入れる.また,制約節点を介してつながった非同期型有向枝と同期型有向枝も同じサブグラフに入れる.

プログラムの手続きブロックはサブグラフ毎に生成して,そのブロックの中の手続きは節点毎に生成する.なお,サブグラフ間には半順序が存在しており,節点間にも半順序が存在している.それらの半順序から全順序を得て手続きの実行順序とする.

4 コンパイラのブートストラッピング

構造不一致の観点から見たコンパイラ処理とカーネル・コンパイラについて述べる.

4.1 コンパイラ処理における構造不一致

前述の検出・解決法を適用したPSDLコンパイラは図3 (1)に示すフェーズ構成を持っている.それらのフェーズと構造不一致の関係について以下に述べる.

(1) 脈絡不一致

頻繁に起きる脈絡不一致はコンパイラの全フェーズのいたる所で現れている.このため,脈絡不一致はブートストラッピングの最初の段階で解決しなければならない.

(2) 順序不一致

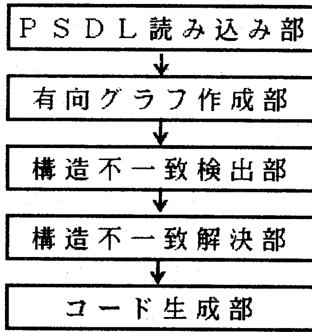
順序不一致は,コンパイラが外部とインタフェースを持つPSDL読み込み部とコード生成部で現れうる.そのうち,同じ型のデータの順序は関数や述語のパラメータの順序で問題になるが,パラメータの順序は保存されるので構造不一致は生じない.一方,データ型間の順序による順序不一致は,C言語の文法に合わせてプログラムを生成するコード生成部で現れる.しかし,文型毎にキー値を割り当ててソートすることによって回避することができる.

(3) 推移閉包による構造不一致

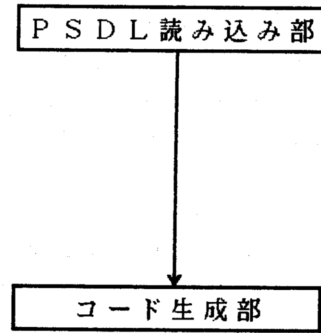
推移閉包による構造不一致は,有向グラフにおいて有向枝から有向道を得る処理や,有向道から閉路を得る処理,有向グラフを連結部分グラフへ分割する処理で現れている.しかも,対象のつながりの螺旋が分流と合流を起こす.このため,分流に対してデータ・ドリブン方式が必要である.しかし,合流については多値の主キー属性が同じ値をとる場合に限定できたので再計算は起きず,バックトラッキング方式は不要である.

(4) その他の構造不一致

境界不一致はと,階層的構造不一致,複合的構造不一致とは初期段階では考慮しなかった.



(1) コンパイラ



(2) カーネル・コンパイラ

図 3: フェーズ構成

4.2 カーネル・コンパイラ

カーネル・コンパイラは PSDL プログラム仕様から実行可能なプログラムを生成できれば, そのプログラムの実行効率を犠牲にしても役割は果たせる. このため, 構造不一致に対して以下の考え方をとって実現した.

(1) クラス 2 の言語

構造不一致の検出処理を省略するため,PSDL をクラス 2 の言語として実現し, すべての実体型と関連型へテーブルを割り当ててプログラムを生成した. このため, フェーズ構成は図 3 (2) のようになった.

(2) データ・ドリブン方式

前述のようにコンパイラ処理には推移閉包が出現して, 対象のつながりの螺旋が分流することによってデータ・ドリブン方式が必要である. 一方, データ・ドリブン方式は構造不一致のない推移閉包や, 脈絡不一致にも対処できるので, 生成されるプログラムは一律に本方式で制御している.

5 実験

PSDL プログラム仕様から C プログラムを生成する実験用コンパイラを SUN 上に実現した. 現在,PSDL の制約の中には関数や述語とそのパラメータを記述し, 関数や述語自体は C 言語で作成している. カーネル・コンパイラは C 言語で実現し, その規模は 5,800 行であった. そのうちの 1,000 行はスケルトンである. 一方, コンパイラは自己記述で実現したが, PSDL 読み込み部はカーネル・コンパイラと共用した. そこで,PSDL 読み込み部以外のコンパイラの規模は 19,700 行であった. そのうちの 500 行はスケルトンである. なお, 関数や述語自体を記述している C 言語の部分は 4,000 行であった. フェーズ毎にコンパイラを見ると, 特にコード生成部が大きく, 約 5 割を占めた. その他のフェーズは同程度の大きさであった.

コンパイラの処理時間は第 2 章で述べた PSDL プログラム仕様で約 3 分かかった (SUN3/60). 一方, カーネル・コンパイラの処理時間は約 2 秒であった. また, コンパイラから生成された C プログラムの規模は 148 行であった. 一方, カーネル・コンパイラから生成された C プログラムの規模は 351 行であった.

現在は自己記述されたコンパイラをコンパイルしている段階である.

6 考察

(1) 記述性と拡張性

規模の小さなプログラム仕様や, 計算の流れが整然と一方向を向いているプログラム仕様は記述や理解は困難でなかった. これらのプログラムについては, PSDL プログラム仕様と, その仕様を図示した PSD 図がドキュメントの役割も果たすことができた. このため,PSD 図を取り扱うエディタの必要性が高い.

一方, 生成されるプログラムの手続き実行順序を有向グラフの中の半順序から得る処理のような複雑な仕様は, 必ずしも PSDL プログラム仕様と PSD 図のみから理解しやすいわけではない. その他にロジックの説明資料が必要であった.

(2) コンパイラの構造

コンパイラのフェーズ構成はコンパイラの機能で決まるので, ブートストラッピングを適用したための影響は受けていない. 一方, 各フェーズの中で使用される

個々のプログラムについては,カーネル・コンパイラによって,すべてのデータがテーブル上に展開されるのでメモリ不足が想定された.このため,大きなプログラムの作成は避けた.これはブートストラッピングの影響である.

(3) コンパイラのロジック

同一の入出力関係を満たすプログラム仕様はいくつか存在するので,理解しやすい方の仕様を記述した.しかし,推移閉包が出現するプログラムでは,実行中に出現する実体数が膨大になり,すなわちデータ量が膨大になり,メモリ不足が想定された.このため,実行中に出現する実体数が少なくなるような仕様を選択した.

7 おわりに

構造不一致を自動的に検出して解決するコンパイラを,構造不一致の分類に従って,自己記述によるブートストラッピングで実現する際の工夫点を述べた.ブートストラッピングの初期段階では,構造不一致の最も基本的な性質を持っている脈絡不一致について実現し,順序不一致はコンパイラの実現に不可欠であるがソート・プログラムの使用で回避した.推移閉包による構造不一致も不可欠なため,カーネル・コンパイラはデータ・ドリブン方式で制御されるプログラムを生成した.

今後は自己記述によるコンパイラを拡張しながら,推移閉包や順序不一致等から構造不一致の研究を深めて行く予定である.

謝辞 日頃ご指導いただく葉原会長,山下社長,竹中室長に深謝いたします.また,ご討論いただいた研究室の諸氏,ならびにPSDLコンパイラの作成にご協力いただいた日本電子計算株式会社の諸氏に感謝いたします.

参考文献

[1] P.P. Chen: The Entity-Relationship Model — Toward a Unified View of Data, *ACM Trans. Database Syst.*, Vol. 1, No. 1,pp. 9-36 (1976).

[2] 橋本正明: プログラム仕様記述のための計算指向EARモデル, 情報処理学会論文誌, Vol. 27, No. 3, pp. 330-338 (1986).

[3] -: EARモデルに基づく情報構造記述を用いたプログラム仕様記述法PSDM, 情報処理学会論文誌, Vol. 27, No. 7, pp. 697-706 (1986).

[4] K. Okamoto, M. hashimoto: On Real-Time Software Specification Description with a conceptual Data Model-Based Language, *Proc. of ICCI*, pp. 186-190, 1990.

[5] 橋本正明: 非手続き型言語と入出力データの構造不一致, 情報処理学会論文誌, Vol. 29, No. 12, pp. 1141-1150 (1988).

[6] -: プログラム構造設計の自動化について, 情報処理学会CASE環境シンポジウム論文集, pp. 101–108 (1989).

[7] M. hashimoto, K. Okamoto: A Set and Mapping-based Detection and Solution Method for Structure Clash between Program Input and Output Data, *Proc. of COMPSAC 90* 掲載予定.

[8] —: 非手続き型言語の集合と写像の性質に着目した入出力データの構造不一致検出・解決法, 情報処理学会プログラミング言語研究会報告,25–3,1990.

[9] M.A. Jackson (鳥居宏次訳): 構造的プログラム設計の原理, p. 318, 日本コンピュータ協会, 東京 (1980).

[10] C. Green, S.J. Westfold: Knowledge-Based Programming Self-Applied, *Machine Intelligence*, Vol. 10, pp. 339-359, 1982.

[11] M.M. Zloof: Query-by-Example: Operations on the Transitive Closure, *IBM Research Report*, RC-5526, Oct. 1976.