4F－5

ＰＳＳ：Ｃ言語プログラムの移植支援システム — 矛盾指摘チェッカ

前田 忠彦[1]、 大浜 美雪[2]、 西風 一[2]
[1]株式会社沖ソフトウェア中国 [2]沖電気工業株式会社

1 はじめに

Ｃ言語プログラム移植支援システム（ＰＳＳ）は、Ｃ言語プログラムの異機種間での移植を支援する統合環境である[1]。矛盾指摘チェッカは、ＰＳＳ中で、Ｃ言語プログラムを解析し、種々の問題点を指摘する。以下では、矛盾指摘チェッカの機能、実現法、及び、評価を報告する。

2 機能概要

矛盾指摘チェッカの機能は、大別すると以下のようになる。

1。文法チェック　通常のコンパイラ並のチェックを行なう。

2。静的診断（狭域）　ＵＮＩＸのｌｉｎｔ第一パス相当のチェックを行なう。

3。文法相違問題指摘　移植元準拠文法と移植先準拠文法の相違による問題を指摘する。

4。処理系相違問題指摘　移植元処理系定義と移植先処理系定義の相違による問題を指摘する。

5。静的診断（広域）　ＵＮＩＸのｌｉｎｔ第二パス相当のチェックを行なう。

6。バイトオーダ依存部指摘　特定のバイトオーダを仮定した部分を検出し、指摘する。

7。バイトオーダ矛盾指摘　プログラマが指定するバイトオーダ仮定の矛盾を検出、指摘する。

8。アクセス境界制約違反検出　ポインタを用いた間接参照の整列境界制約違反を検出、指摘する。

3 構成

矛盾指摘チェッカは、次の二つから構成される。（図1参照）

1。情報収集部　Ｃソースプログラムを解析し、機能1～4を実現するとともに、機能5～8を実現するのに必要な情報を収集する。指摘、収集情報はともに、ソースファイル毎に一つの狭域解析ファイルに入れられる。収集情報の主としては、記号表、後述するポインタデータフローグラフがある。

2。矛盾解析部　情報収集部が収集した情報、及び、後述するシステム情報、パッケージ情報を元に、機能5～8を実現する。指摘は、一つのファイル（広域情報ファイル）に入れられる。このファイルには、記号表、ポインタデータフローグラフも含まれ、指摘情報が参照するとともに、エディタの操作対象となる[1]。

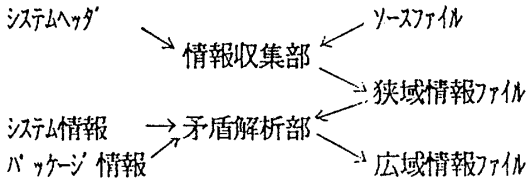

図1　構成

システム情報、パッケージ情報は、移植対象プログラムが引用するシステム、パッケージの情報であり、システム、パッケージを定義する者が情報収集部を用いて作る。システム情報は自動的に引用される。パッケージ情報は必要なものを、移植作業者が指定する。

4 ポインタデータフローグラフ

ポインタデータフローグラフ（以下では、単にグラフと呼ぶ）は、広域の各種解析のための基本データである。ポインタ値の発生とその伝播をグラフ表現したものである。ポインタ値の発生には、オブジェクトのアドレス参照、ポインタ型データ参照がある。伝播には、（直接・間接）代入、（直接・間接）関数呼出しのパラメタ渡し・関数値返却、ポインタ演算等がある。（図2参照）

```
int *p, **pp, **qq;
pp = &p;
qq = pp + 1;
*qq = r;
```

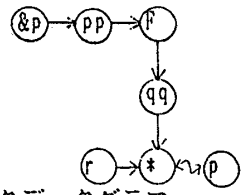

図2　ポインタデータグラフ

PSS:The Porting Support System for C-programs - A Porting Checker
Tadahiko MAEDA[1], Miyuki OHAMA[2], Hajime NISHIKAZE[2]
[1]Oki Software Chuugoku Co., Ltd. [2]Oki Electric Industry Co., Ltd.

5　狭域解析

情報収集部が行なう狭域解析のうち、移植に直接関連する部分の概要を説明する。

1。文法相違問題指摘　　K&R文法、ANSI－C文法等の相違を知っており、移植元と移植先で文法解釈が異なる場合、その点を指摘する。

例：３文字表記

2。処理系定義相違問題指摘　　C文法のうち処理系定義部分について、各処理系での定義を知っており、継承元と継承先で意味が異なる場合、その点を指摘する。

例：char型の符号の有無

6　バイトオーダ依存部検出

バイトオーダ依存は、次の二つで表現できる。

・同じオブジェクトを異なる型で参照している。

```
例：    int     i;
        i = 0x12345678;   ・・・int型
        *(char *)&i       ・・・char型
```

・オブジェクトの境界をまたいで参照している。

```
例：    int     a[2];
        a[0]
        *(int *)((short *)a+1)
              a[0]の後半とa[1]の前半を参照
```

このことから、各オブジェクトに対し全ての参照開始ビット位置と参照する型を調べ、それらの間の関係を解析すればバイトオーダ部を検出できる事がわかる。実現法は次のようにした。オブジェクトをオブジェクトへのポインタの発生源とし、そのポインタの流れ（代入関係とポインタの加減算）をグラフ上で追い、到達する全ての参照に対し、参照開始ビット位置と参照型を調べる。指摘は、異なる型で参照をしている２点を両端とする、グラフ上の経路で表現する。（図３参照）

```
int   i,*p;       +0   +0        +0  ビット位置
short *q;         int  int       short  型
p = &i;           (&i)→(p)→(T)→(q)
q = (short *)p;
```

図３　バイトオーダ依存

7　バイトオーダ矛盾検出

同じオブジェクトを同じ型で参照していても（つまり、バイトオーダ依存になっていなくても）リトルエンディアンと解釈している部分とビッグエンディアンと解釈している場合、正しく動作しない。同一のグラフ中に、異なるエンディアンデータが混在していたら、バイトオーダ矛盾として検出する。指摘は、異なるエンディアンデータを両端とする、グラフ上の経路で表現する。

8　アクセス境界制約違反検出

プロセッサによっては、アクセスするデータの型（大きさ）によるアクセスアドレスに制約がある。コンパイラはこれらの制約をできる限り守るようデータを割り付けていく。しかし、ポインタを使ったデータアクセスに関しては、コンパイラは介入できない。更に、移植した場合、制約が異なることがあり、移植元で制約を満たしていたプログラムが移植先でも制約を満たしている保証はない。

```
例：    short   b[10];
        *(int *)(b+1)
```

ポインタによる参照が制約を満たしているかどうか調べるため、次の処理を行なった。

各オブジェクトはコンパイラにより型に応じたアドレス境界値を持つものと仮定する。オブジェクトをアドレス境界の基点とし、そのポインタ値の流れ（代入関係とポインタの加減算）をグラフ上で追い、到達する全ての参照に対し、ポインタ値と参照する型の適合性を調べる。（図４参照）

```
int i,*p,*q;
p = &i;
q = (int *)((char *)p + 1);
  W    W    W    B    B    B      W：語境界
(&i)→(p)→(T)→(F)→(T)→(q)        B：バイト境界
```

図４　アクセス境界制約違反

9　評価

試験的な使用により、次のような評価が得られている。

・狭域解析ファイルの大きさは、OMの数倍の大きさ。処理時間は、コンパイル程度。

・広域解析ファイルの大きさは、狭域解析ファイルの大きさと指摘の大きさ。処理時間は、リンクの５倍～。

・検出した問題点をすべて出力すると、その量が非常に大きくなり（処理時間も長くなるので）、出力量を選択する必要がある。

１０　おわりに

現在、実際の移植に適用しながら、指摘の的中率向上、指摘機能の向上を検討中である。

参考文献

[1]西風　他：ＰＳＳ：Ｃ言語プログラムの移植支援システムーアーキテクチャ　情処全大４４回　４Ｆ－０５