4Y-6

Ａｄａ®プログラムの変更に伴う再コンパイル削減方式

生沼守英，梶原清彦，伊集院正

ＮＴＴ　ソフトウェア研究所

1．はじめに

大部分のＡｄａコンパイラでは、ライブラリ単位の仕様の変更を行なった場合には、それを文脈節（with）で参照している全てのコンパイル単位を再コンパイルすることが必須である。また、再コンパイルされるものがライブラリ単位である場合は、それを参照している全てのコンパイル単位の再コンパイルも必須である。このため、数百から数千のコンパイル単位からなるような大規模プログラムでは、再コンパイルすべきコンパイル単位が多くなり問題となる。このため、処理系の工夫によって再コンパイルを削減する必要がある。

本論文では、ライブラリ単位の仕様内部の変更箇所と、変更の影響範囲を解析し、変更の影響を受けないと判断できるコンパイル単位の再コンパイルを行なわないことによって、再コンパイルを削減する方式について述べる。

2．再コンパイル削減の方式

大部分のコンパイラでは、プログラムに変更を行なった場合、変更したコンパイル単位に関する文脈節に基づいて、再コンパイルが必要となるコンパイル単位を決定する。このため、ライブラリ単位を変更した場合、そのライブラリ単位を文脈節で参照しているコンパイル単位の再コンパイルは、実際に変更部分を参照しているか否かにかかわらず必須であり、更に、そのコンパイル単位を参照しているコンパイル単位についても再コンパイルが必須となる。

これに対して、本方式では、パッケージ仕様内の宣言を単位として変更箇所を解析し、文脈節で得られるコンパイル単位の内で、変更箇所の影響を受けないコンパイル単位を探し、それについての再コンパイルを行なわない。すなわち、変更されたライブラリ単位を参照しているコンパイル単位であっても、変更された宣言部分を実際に参照していなければ、そのコンパイル単位の再コンパイルを不要とすることにより、再コンパイルを行なうコンパイル単位を削減する。

2．1　再コンパイル範囲の解析

変更に伴って再コンパイルが必要となるコンパイル単位を解析するためには、次に述べる２つの解析が必要である。

（１）変更箇所の解析

変更箇所の解析方法としては、変更前後で①ソースプログラムを比較する方法と、②ソースプログラムを意味解析して得られる中間言語を比較する方法がある。②は、（ｉ）注釈の変更など、字面だけの変更を無視できる、（ii）次に行なう影響範囲の解析が行い易い、という利点がある。このため、②の中間言語を比較する方法を採用した。中間言語としては、Ａｄａの標準中間言語として提案されているＤＩＡＮＡ[1)]を採用する。ＤＩＡＮＡは木構造を持つ中間言語であり、変更前と変更後のＤＩＡＮＡの各々について、コンパイル単位のルートノードからノードを一つずつたどって比較することによって、変更箇所を明らかにすることができる。変更前のＤＩＡＮＡには、変更箇所に対応するノードに、変更があったことを示すフラグ（変更フラグ）を立てる。

（２）変更の影響範囲の解析

変更の影響範囲の解析にもＤＩＡＮＡを用いる。プログラム内に現われる名前に対応するＤＩＡＮＡのノードは、その名前の宣言や型を表すノードへのポインタを持っている。従って、あるノードが持っている、これらのポインタが指すノードの一つにでも前記の変更フラグが立っていれば、そのＤＩＡＮＡノードも変更の影響を受けるものと判断することができる（図１）。そして、そのノードにも変更フラグを立てる。

変更したライブラリ単位を根として文脈節によって決まるコンパイル順序にしたがって、コンパイル単位を一つづつたどり、コンパイル単位中の総てのＤＩＡＮＡノードをたどって、上記の操作を行なう。これらの操作がすべて完了した後、少なくとも一つのノードに変更フラグが立っているコンパイル単位は、変更の影響を受けるものと判断できる。そして、その結果求められた、変更の影響を受けるコンパイル単位だけに対して再コンパイルを行なう。

2．2　ＤＩＡＮＡと目的プログラムの維持

再コンパイル削減を実現するためには、一部のコンパイル単位の再コンパイル後にも、ＤＩＡＮＡと目的プログラムを正しく維持する必要がある。従って、前述の解析機能の他に次の２つの機能が必要である。

（１）中間言語ＤＩＡＮＡの再構成

変更されたライブラリ単位やその影響を受けたコンパイル単位が再コンパイルされると、一般にそれらのＤＩＡＮＡは再コンパイル前と違う木を形作る。この

Reducing Recomplation Cost in an Ada Compiler

Morihide OINUMA, Kiyohiko KAJIHARA, Sei IJUIN

NTT Software Lab.

ため、再コンパイル前のＤＩＡＮＡを参照しているコンパイル単位のＤＩＡＮＡのポインタは不正となる。従って、参照先コンパイル単位の再コンパイル後も、参照元コンパイル単位の再コンパイルを行なうことなく、ＤＩＡＮＡの参照関係を維持する機能が必要である。

これを解決するため、変更の影響が全く無かったコンパイル単位のＤＩＡＮＡの中で、再コンパイル前と異なる位置へ移動した参照先のノードを指す総てのポインタを、新しいノード位置を指すように書き換える（図２）。これにより、ＤＩＡＮＡ上の参照関係を正しく保つことができる。

この処理が必要となるのは、再コンパイルされたコンパイル単位を指すノードが存在するコンパイル単位だけであるため、そうでない多くのコンパイル単位に対しては何もする必要が無い。

（２）変数等のアドレスの変更への対応

変更に伴って、変更したプログラム中の変数等の割り付けが変更になる。従って、処理系によっては、他のコンパイル単位から、再コンパイルしたプログラム中の変数等に対して正しく参照できるように目的プログラムの生成方法を変更する必要がある。

そのための方法として、次の二つをあげることができる。

（ｉ）外部の変数等のアドレスをコンパイル時に決定するのではなく、参照関係の解決はリンク時に行なうこととする。これにより、実際の変数の割り付けアドレスは、他のコンパイル単位へは影響しなくなる。

（ii）外部に見える変数等のアドレスは、何も変更がなかったものについては、変更前と同じアドレスを割り付ける。参照側の目的プログラムは何等変更の必要が無い。これを実現するために、各変数毎に、その割付アドレスを記録しておく。

（ii）の方法では、削除された変数等の領域が無駄になり、これが無視できない大きさになる場合がある。また、（ｉ）の方法では、領域の無駄はないが、これまでよりも、リンカで処理すべき外部名が増え、リンカの負担が増加する。

３．おわりに

Ａｄａコンパイラにおいて、中間言語の情報を利用して、ソースプログラムの変更箇所、変更の影響範囲を認識し、不必要な再コンパイルを削減する方式を提案した。

今後、再コンパイル削減の効果について評価を進める。

［参考文献］

1)AJPO:"DIANA REFERENCE MANUAL",(1983)

Ａｄａは米国政府ＡＪＰＯの登録商標である

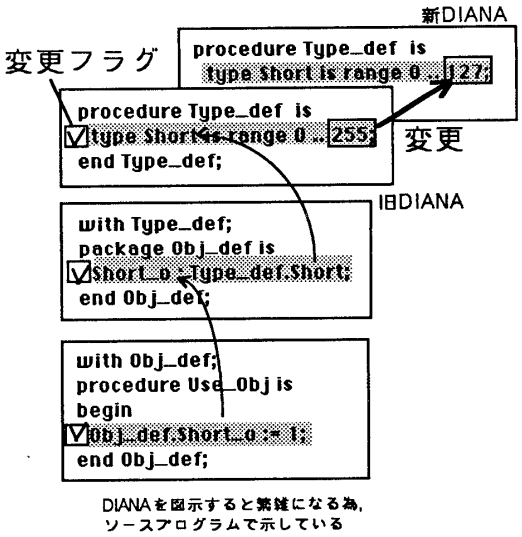

図1 変更フラグ表示方式

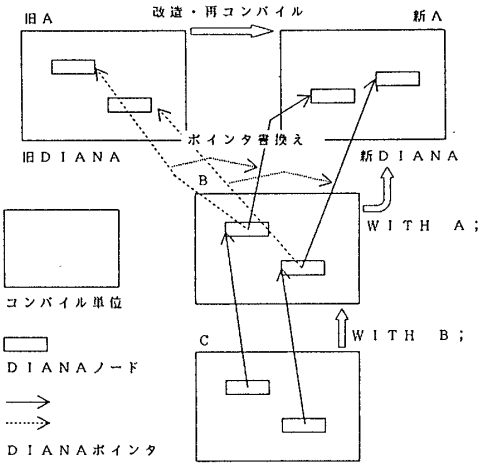

図２　ＤＩＡＮＡのポインタ書換え