(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4615795号
(P4615795)
(45)発行日　平成23年1月19日(2011.1.19)　(24)登録日　平成22年10月29日(2010.10.29)

(51)Int.Cl.　F I
G06F 11/00 (2006.01)　G06F 9/06 630B

請求項の数 6　(全 33 頁)

(21)出願番号　特願2001-349299(P2001-349299)
(22)出願日　平成13年11月14日(2001.11.14)
(65)公開番号　特開2002-169702(P2002-169702A)
(43)公開日　平成14年6月14日(2002.6.14)
審査請求日　平成16年11月15日(2004.11.15)
審判番号　不服2007-23318(P2007-23318/J1)
審判請求日　平成19年8月23日(2007.8.23)
(31)優先権主張番号　09/713,633
(32)優先日　平成12年11月14日(2000.11.14)
(33)優先権主張国　米国(US)

(73)特許権者　500046438
マイクロソフト　コーポレーション
アメリカ合衆国　ワシントン州　９８０５２－６３９９　レッドモンド　ワン　マイクロソフト　ウェイ
(74)代理人　100077481
弁理士　谷　義一
(74)代理人　100088915
弁理士　阿部　和夫
(74)復代理人　100115624
弁理士　濱中　淳宏
(74)復代理人　100115635
弁理士　窪田　郁大

最終頁に続く

(54)【発明の名称】プログラムバイナリに対する最小デルタジェネレータ

(57)【特許請求の範囲】
【請求項１】
第１のプログラムバイナリと第２のプログラムバイナリとの間でデルタを生成するコンピュータが実施するデルタ生成方法であって、
コンピュータのプロセッサが、コンピュータ記憶媒体に記憶された第１のプログラムバイナリの第１の制御フローグラフ（ＣＦＧ）表現を作成し、前記コンピュータ記憶媒体に記憶された第２のプログラムバイナリの第２のＣＦＧ表現を作成するステップであって、前記第１のＣＦＧ表現および前記第２のＣＦＧ表現は、それぞれ、前記第１のプログラムバイナリおよび前記第２のプログラムバイナリの各ブロックをノードで表し、各ノード間の制御フローをエッジで表した有向グラフであって、前記ブロックを表すノードには、当該ブロックの内容が同一であれば、第１のＣＦＧ表現及び第２のＣＦＧ表現に共通して同一のノード情報が付与され、当該ブロックの内容が同一でなければ、それぞれ異なるノード情報が付与された有向グラフを作成する前記ステップと、
前記コンピュータのプロセッサが、前記第１のＣＦＧ表現および前記第２のＣＦＧ表現の間でブロックを比較して、前記第１のＣＦＧ表現中および前記第２のＣＦＧ表現中でマッチするブロック（マッチするブロック）を識別し、それにより、前記第１のＣＦＧ表現中でマッチしない前記第２のＣＦＧ表現中のブロック（マッチしないブロック）を識別するステップであって、前記比較は、比較されるブロックを表すノードに対するノード情報と、比較されるブロックの周囲のブロックの拡張された局所近傍のブロックを表すノードに対するノード情報を前記第１のＣＦＧ表現および前記第２のＣＦＧ表現の間で比較する

ことにより行われ、各ブロックの局所近傍は、ＣＦＧ表現中の前記ブロックの近傍であるが前記ＣＦＧ表現中の全てのブロックより少ないブロックからなり、前記ブロックの拡張された局所近傍は、前記ブロックの局所近傍と、前記ブロックの局所近傍より大きい近傍において該ブロックから固定長の一様なランダムウォークを行う間に遭遇するブロックの集合とからなり、ＣＦＧ表現中の拡張された局所近傍は、前記ＣＦＧ表現中の全てのブロックより少ないブロックからなる、識別するステップと、

前記コンピュータのプロセッサが、パッチを当てた前記第１のプログラムバイナリから作成される新たな第１のＣＦＧ表現が前記第２のＣＦＧ表現と同一になるように、パッチを当てた前記第１のプログラムバイナリが前記第２のプログラムバイナリと同一になるように、マッチしないブロックを前記第１のプログラムバイナリにパッチする編集操作を決定するステップと、

前記コンピュータのプロセッサが、前記マッチしないブロックと前記編集操作とを含むデルタを作成するステップと

を備えることを特徴とするデルタ生成方法。

【請求項２】

前記プロセッサが、第１のプログラムバイナリを有するコンピュータに前記デルタを伝送するステップ

をさらに備えることを特徴とする請求項１に記載のデルタ生成方法。

【請求項３】

前記プロセッサが、前記第１のプログラムバイナリのコピーが前記第２のプログラムバイナリと同一になるように前記コピーをパッチするステップであって、前記デルタはこのようなパッチングを導くステップ

をさらに備えることを特徴とする請求項１に記載のデルタ生成方法。

【請求項４】

前記ブロックの局所近傍は、前記ブロックに隣接するブロックからなることを特徴とする請求項１に記載のデルタ生成方法。

【請求項５】

請求項１に記載のデルタ生成方法をコンピュータに実行させるためのコンピュータ実行可能命令からなるプログラムを有することを特徴とするコンピュータ記憶媒体。

【請求項６】

記憶媒体に記憶された第１のプログラムバイナリの第１の制御フローグラフ（ＣＦＧ）表現と、前記記憶媒体に記憶された第２のプログラムバイナリの第２のＣＦＧ表現の間でブロックを比較して、前記第１のＣＦＧ表現中と前記第２のＣＦＧ表現中とでマッチするブロック（マッチするブロック）を識別し、それにより、前記第１のＣＦＧ表現中でマッチしない前記第２のＣＦＧ表現中のブロック（マッチしないブロック）を識別するように構成されたコンパレータであって、前記第１のＣＦＧ表現および前記第２のＣＦＧ表現は、それぞれ、前記第１のプログラムバイナリおよび前記第２のプログラムバイナリの各ブロックをノードで表し、各ノード間の制御フローをエッジで表した有向グラフであって、前記ブロックを表すノードには、当該ブロックの内容が同一であれば、第１のＣＦＧ表現及び第２のＣＦＧ表現に共通して同一のノード情報が付与され、当該ブロックの内容が同一でなければ、それぞれ異なるノード情報が付与された有向グラフであり、前記比較は、比較されるブロックを表すノードに対するノード情報と、比較されるブロックの周囲のブロックの拡張された局所近傍のブロックを表すノードに対するノード情報を前記第１のＣＦＧ表現および前記第２のＣＦＧ表現の間で比較することにより行われ、各ブロックの局所近傍は、ＣＦＧ表現中の前記ブロックの近傍であるが前記ＣＦＧ表現中の全てのブロックより少ないブロックからなり、前記ブロックの拡張された局所近傍は、前記ブロックの局所近傍と、前記ブロックの局所近傍より大きい近傍において該ブロックから固定長の一様なランダムウォークを行う間に遭遇するブロックの集合とからなり、ＣＦＧ表現中の拡張された局所近傍は、前記ＣＦＧ表現中の全てのブロックより少ないブロックからなる、コンパレータと、

パッチを当てた前記第１のプログラムバイナリから作成される新たな第１のＣＦＧ表現が前記第２のＣＦＧ表現と同一になるように、パッチを当てた前記第１のプログラムバイナリが前記第２のプログラムバイナリと同一になるように、マッチしないブロックを前記第１のプログラムバイナリにパッチする編集操作を決定するように構成された編集操作決定機構と、

前記マッチしないブロックと前記編集操作とを含むデルタを作成するように構成された出力サブシステムと

を備えたことを特徴とするデルタジェネレータシステム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は一般に、プログラムバイナリ間で最小限のデルタを生成することに関する。

【０００２】

【従来の技術】

今日のソフトウェアシステムの重要な特徴の１つはアップグレードできることであり、これは「アップグレード性（ｕｐｇｒａｄａｂｉｌｉｔｙ）」と呼ばれることもある。古いソフトウェアは、絶えずより新しいバージョンで置換されており、コードの再使用可能性およびモジュラ開発が、ソフトウェア設計の主要な特徴である。

【０００３】

正確さ

ソフトウェアを古いバージョンから新しいバージョンにアップグレードするときは、完全な正確さが極めて重要である。ターゲットコンピュータ中で新たにアップグレードしたソフトウェア中のあらゆるビットが、媒体ソースにある新しいソフトウェアと正確に合致（マッチ）しなければならない。さもなければ、新しいソフトウェアは誤動作するか全く動作しない恐れがある。

【０００４】

完全な正確さを保証するために、従来の技術では、古いソフトウェアを新しいソフトウェアで完全に置換する。ソフトウェアプログラム（特に主要なアプリケーションスイートおよびオペレーティングシステム）のサイズおよび複雑さが増すにつれて、この十把ひとからげに置換して更新する手法は、このようなソフトウェアの顧客にとってより時間がかかり苛立たしいものになっている。

【０００５】

このような更新のソースを、ローカル、携帯型かつ高帯域幅のリムーバブルメディア（ＣＤ－ＲＯＭなど）からリモート、集中型かつ比較的低い帯域幅のネットワークサーバ（インターネットウェブサーバなど）に移行する傾向により、事態は悪化している。１００ＭＢのソフトウェアをおそらくはＣＤ－ＲＯＭから置換するのには数分かかることがあるが、同じ量のソフトウェアをダイヤルアップインターネット接続で置換するのには数時間かかることがある。

【０００６】

本明細書で、「完全な正確さ」および「ほぼ同一」は、元々製作された新しいソフトウェアとユーザのコンピュータ上に存在する新しいソフトウェアの間での小さく実質的でない差を考慮に入れている。

【０００７】

従来のデルタパッチング

通常、ソフトウェアのより新しいバージョンは、わずかな追加部分、ならびに、古い部分におけるいくつかの小さな変更を有する。したがって、古いバージョンを新しいバージョンで完全に置換する力任せの手法は行き過ぎである。ある代替手法は、古いバージョンから新しいバージョンを再構築できるように、これらの変更を「パッチ」に取り込むものである。古いバージョンと新しいバージョンの間に差があるので、この技法はしばしば「差分パッチング（ｄｉｆｆ－ｐａｔｃｈｉｎｇ）」と呼ばれる。本明細書では、古いバージ

ョンと新しいバージョンとの差を「デルタ」（Δ）と呼び、したがってこの差分パッチング技法を「デルタパッチング」（または「Δパッチング」）と呼ぶ。

【０００８】

デルタパッチングに伴う問題は正確さである。何をパッチして何をパッチしないかを識別するのは難しい。このようなパッチの境界を正確に決定しないと、パッチしたバージョンは、ソフトウェアの所望の新バージョンとは異なるものになる。

【０００９】

この結果、従来のデルタパッチングでは、正確さを達成する効率が損なわれる。一般に、サブモジュールファイル、データファイル、ライブラリファイル、およびこのようなファイルのグループは、どんな変更であれそれらの内に変更があるかどうかがマークされる。このことは例えば、１００ＫｂのＤＬＬ（ダイナミックリンクライブラリ）ファイル内のソースコードの一行が変更される場合でもＤＬＬファイル全体を置換することを意味する。既存のＤＬＬファイル中のフラグメントを置換するのではなくこのようにするのは、一部には、置換が必要なフラグメントを選択してそのフラグメントだけを完全に正確に置換するのが難しいからである。しかし主に、従来の技術ではモジュール全体を置換する方が効率的なので、このことが行われる。ある小さなフラグメント中の小さな変更が、プログラム全体に及ぶ変更に見える場合もある。

【００１０】

【発明が解決しようとする課題】

この従来の非効率的なデルタパッチングは、ソフトウェア全体の大規模な置換よりは効率的で速いものの、依然として可能な限り効率的ではない。古いソフトウェアバージョンと形の異なる、または古いソフトウェアバージョン中に存在しない、モジュールまたはサブモジュールのフラグメントだけをパッチする方が効率的であろう。フラグメントの例としては、サブルーチン、関数、オブジェクト、データ構造、インタフェース、メソッド、またはこれらのいずれかの一部が挙げられる。

【００１１】

不変フラグメントの検出

フラグメントデルタを検出するための前提は、フラグメントの不変性を検出できることである。言い換えれば、プログラムモジュールをパッチできるようになる前に、２つのバージョン間でどのフラグメントが変更されていないかを決定する必要がある。このような不変量を検出してパッチを生み出すことは、各バージョンに対するソースコードを知っていればそれほど難しくはない。

【００１２】

しかし、このようなフラグメントがバイナリとして表されたものを（そのソースコードを知らずに）扱うときは、フラグメントの不変性の検出はかなり難しくなる。主な困難は、プログラムモジュールの異なるバージョン中に、機能的には変更されていないが異なるように見えるコードが存在することある。コードは、その機能の変更を受けないこともあるが、種々の理由で２つのバージョン中で異なって見えることがある。このような理由の例としては、次のことが挙げられる。

・コードのある領域の変更により、別の（変更されない）領域が異なって見える可能性がある。

・小さな２つのバイナリコードシーケンスが、異なる機能を有するソースコードに対応する場合でも同一に見えることがある。

・２つの構造におけるレジスタ割付けの違い。

【００１３】

変更が見かけ上の変更を引き起こす場合

コードのある部分の小さな変更は、しばしば近くで、また時として遠く離れたコード領域でも、連続的な変更を引き起こす。例えば、以下の２つのソースフラグメントを考えてみる。

```
Program P1                Program P2
----------                ----------
function f (int p)        function g (int p)
int a = 3, b = 4;         int b = 4, a = 3;
if (b > p) {              if (b > p) {
    a = p;                    a = p;
return a;                 return a;
```

【００１４】
２つのプログラムＰ１およびＰ２中にある２つの関数ｆおよびｇは、名前の違いを別にすれば実際は同じである。ソースコードを知っていれば、各フラグメント中の「ｉｆ（ｂ＞ｐ）」条件が同じでありパッチする必要がないことが確定するのは明らかである。しかし、これらに対応するバイナリを検査する場合、スタックのベースからのｂのオフセットは、この２つのフラグメント中で異なることになる。これは、Ｐ１中でｂの前にａが宣言されている形が、Ｐ２中でａの前にｂが宣言されている形と異なるからである。したがって、他がすべて同じであっても、２つのフラグメントのバイナリは同一ではないことになる。当然、形の上でのこれらの違いは実質的には関係ないが、それでもやはり得られるバイナリは異なる。

【００１５】
次に、以下の抜粋を考えてみる。

```
Program P1            Program P2
----------            ----------
x = f (10)            x = g (10)
```

【００１６】
この例では、前の例と同様に関数ｆおよびｇが定義されていると仮定する。ここでもまた、呼び出される関数ならびに呼出し引数が同一なので、２つの呼出しは同一である。しかし、ｆとｇの一致を知らない場合は、上の呼出しの一致もわからないことになる。これは、局部的な変更が、ともすれば遠くのコード領域を通ってどのようにカスケードする可能性があるかを示す一例である。

【００１７】
同一に見えるが同一ではない場合
時として２つのバイナリフラグメントが、対応するプログラムの構造中で異なる領域に対応するにもかかわらず同一に見える場合がある。以下の場合を考えてみる。

```
Program P1                  Program P2
----------                  -----------------
int a = atoi (argv [1]);    int b = atoi (argv [2]):
int b = atoi (argv [2])     if (b < 10) return;
if (a < 10) return;         ...
if (b < 20) return;
....
```

【００１８】
Ｐ１中の条件「ｉｆ（ａ＜１０）」とＰ２中の条件「ｉｆ（ｂ＜１０）」は、これらの機能が異なっても（これらのソースコードを検査すればはっきりわかるが）、両方とも同じバイナリコードに翻訳される場合がある。これは、Ｐ２中のスタック上のｂのオフセットが、Ｐ１のスタック上のａのオフセットと同じである場合があるために起こる。２つの変数は、これらの上にあるコンテキストを見ればわかるように異なるプログラム引数によって定義されているので、明らかに異なる。しかし、上にあるソースコードコンテキストを

参照せずにこれらのバイナリに相当するものを比較すると、同一であるかのような錯覚が生じる可能性がある。このバイナリに相当するものの表現は、次のように見える場合がある。

```
mov eax, dword ptr [ebp+8h]
cmp eax, 0ah
jge L
ret
L: ...
```

【００１９】

レジスタ割付け

バイナリフラグメントの一致を検出する際のもう１つの問題は、レジスタ割付けによって生じる。コードの一部を変更すると、近くの領域が修正されていなくても、これらの後方の領域でレジスタ割付けの変更を引き起こす場合がある。したがって、バイナリを比較するときは、レジスタオペランドの変更のように見えるものが実際はレジスタの単純な名前変更によって識別が変えられたものだという可能性を考慮しなければならない。

【００２０】

【課題を解決するための手段】

本明細書では、少なくとも２つのプログラムバイナリ間で最小限のデルタを生成する技術について述べる。本明細書に述べる一実施形態では、バイナリフォーマットのソースプログラム（Ｓ）と、バイナリ形式のターゲットプログラム（Ｔ）が提供される。それぞれの制御フローグラフ（ＣＦＧ）を構築する。ＳのＣＦＧとＴのＣＦＧとの共通ブロックを突き合わせる（マッチング）。ブロックは、それらの内容およびそれらの局所近傍（例えばｄ近傍）に基づいてマッチングする。さらにブロックは、計算したハッシュ値に基づくラベルを使用してマッチングする。マッチングは複数のパスで行い、各パスは、マッチのための基準を緩和することによってマッチングを向上させる。さらに、ブロックを公正に比較できるように、レジスタ名の変更の問題を解決する。

【００２１】

上記の実施形態は中間出力を生み出すが、これはマッチしないブロックの内容である。このようなマッチしないブロックは、Ｔ中に見られるがＳ中には見られないブロックである。マッチしないブロックをＳにマージするためのエッジ編集操作のセットを生み出す。マッチしないブロックと編集操作とを組み合わせたものがデルタである。Ｓをパッチしてその再構築コピーを生み出すには、デルタをＳとマージする。

【００２２】

この概要自体は、本発明の範囲を限定するものではない。本発明をよりよく理解するために、以下の詳細な説明および頭記の特許請求の範囲を添付の図面と共に参照されたい。本発明の範囲は頭記の特許請求の範囲に示す。

【００２３】

図面全体にわたり、同じ要素および機構を参照するのに同じ番号を使用する。

【００２４】

【発明の実施の形態】

後続の記述では、頭記の特許請求の範囲に挙げる要素を組み入れた、プログラムバイナリに対する最小デルタジェネレータの具体的な実施形態を述べる。これらの実施形態は、法定の書面による記述、実施可能であること、および最良の形態の要件を満たすために具体的に述べるものである。ただし、この記述自体は本特許の範囲を限定するものではない。

【００２５】

本明細書では、プログラムバイナリに対する最小デルタジェネレータの、１つまたは複数の例示的な実施形態について述べる。発明者は、これらの例示的な実施形態を例として意図している。発明者は、これらの例示的な実施形態が、請求項に記載された本発明の範囲を限定するものであるとは意図しない。むしろ、請求項に記載された本発明が、現在または将来の他の技術を用いて他の形でも実施および実装できることを企図している。

【００２６】
参照による組込み
以下の同時係属の特許出願は、すべて１９９９年6月３０日に出願され、Ｍｉｃｒｏｓｏｆｔ　Ｃｏｒｐｏｒａｔｉｏｎに譲渡されたものである。これらを参照により本明細書に組み込む。
「Ｔｒａｎｓｌａｔｉｏｎ　ａｎｄ　Ｔｒａｎｓｆｏｒｍａｔｉｏｎ　ｏｆ　Ｈｅｔｅｒｏｇｅｎｅｏｕｓ　Ｐｒｏｇｒａｍｓ」という名称の米国特許出願第０９／３４３，８０５号
「Ｉｎｓｔｒｕｍｅｎｔａｔｉｏｎ　ａｎｄ　Ｏｐｔｉｍｉｚａｔｉｏｎ　Ｔｏｏｌｓ　ｆｏｒ　Ｈｅｔｅｒｏｇｅｎｅｏｕｓ　Ｐｒｏｇｒａｍｓ」という名称の米国特許出願第０９／３４３，２９８号
「Ｓｈａｒｅｄ　Ｌｉｂｒａｒｙ　Ｏｐｔｉｍｉｚａｔｉｏｎ　ｆｏｒ　Ｈｅｔｅｒｏｇｅｎｅｏｕｓ　Ｐｒｏｇｒａｍｓ」という名称の米国特許出願第０９／３４３，２７９号
「Ａｐｐｌｉｃａｔｉｏｎ　Ｐｒｏｇｒａｍ　Ｉｎｔｅｒｆａｃｅ　ｆｏｒ　Ｔｒａｎｓｆｏｒｍｉｎｇ　Ｈｅｔｅｒｏｇｅｎｅｏｕｓ　Ｐｒｏｇｒａｍｓ」という名称の米国特許出願第０９／３４３，２７６号
「Ｃｒｏｓｓ　Ｍｏｄｕｌｅ　Ｒｅｐｒｅｓｅｎｔａｔｉｏｎ　ｏｆ　Ｈｅｔｅｒｏｇｅｎｅｏｕｓ　Ｐｒｏｇｒａｍｓ」という名称の米国特許出願第０９／３４３，２８７号
【００２７】
簡単な概略
プログラムバイナリに対する最小デルタジェネレータの例示的な一実施形態は、例示的な「デルタジェネレータ」と呼ぶことができる。本明細書に述べる１つまたは複数の例示的な実施形態は、（全体的または部分的に）図１のデルタジェネレータシステム１００および／または図１１に示すコンピューティング環境によって実装することができる。
【００２８】
効率および速度を促進するために、例示的なデルタジェネレータは、バイナリソースとターゲットプログラムとの最小限の差を見つけ、それを用いてソースプログラムを修正する。しかし、このような効率および速度のために正確さが損なわれることはない。
【００２９】
これを達成するために、例示的なデルタジェネレータは、ソースプログラムとターゲットプログラムのバイナリフラグメントを比較する。これらの間で同じ（すなわち変更されていない）フラグメントを識別する。これらの間で異なる（すなわち変更された）フラグメントを識別することも必要である。変更されたフラグメントをデルタに含める。例示的なデルタジェネレータは、ソースをデルタでパッチしてターゲットプログラムに変形させる。
【００３０】
デルタジェネレータシステムの高レベルな記述
図１に、デルタジェネレータシステム１００を示す。この図には、バイナリソースプログラム１１２およびバイナリターゲットプログラム１２２が示されている。デルタジェネレータシステム１００は、ソースプログラム１１２およびデルタ（Δ）１４２からターゲットプログラム１２２を再構築する。ターゲットプログラムとソースプログラムは、同じプログラムの異なるバージョンであると想定し、したがってこれらのコードの少なくともいくつかの部分は共通する。例えば、ターゲットプログラムはソースプログラムの新しいバージョンとすることができる。
【００３１】
ソースＣＦＧ機構１１０が、ソースプログラム１１２の制御フローグラフ（ＣＦＧ）を作成する。ＣＦＧについては、「用語」セクションでより詳細に述べる。同様に、ターゲットＣＦＧ機構１２０が、ターゲットプログラム１２２のＣＦＧを作成する。当然、この２つのＣＦＧ機構１１０および１２０は共通のＣＦＧ機構としてもよい。
【００３２】

図１には、ソースＣＦＧおよびターゲットＣＦＧを入力として受け取ってこれらを比較するブロックマッチャ（ｍａｔｃｈｅｒ）１３０が示されている。これは、ソースＣＦＧとターゲットＣＦＧの両方に見られるバイナリフラグメントをマッチングする。

【００３３】

デルタジェネレータ１４０は、デルタ（Δ）フラグメントとしてマッチしなかった、ターゲットＣＦＧ中の残りの（すなわちマッチしない）フラグメントを識別する。ブロックマッチャ１３０とデルタジェネレータ１４０がまとまって、ターゲットＣＦＧ中のバイナリフラグメントを、ソースＣＦＧ中でマッチするものとソースＣＦＧ中でマッチしないものとに識別する。

【００３４】

さらに、Δ決定機構１４０がまた、ΔをソースＣＦＧ中にマージするためにどのようにソースＣＦＧを編集するかを決定する。「編集」はΔ１４２の一部であり、Δ１４２はΔ決定機構１４０の結果である。

【００３５】

Δパッチャ１５０が、ソースプログラムをバイナリレベルでパッチする。ソースプログラム１１２をΔ１４２と結合して、ターゲットプログラム１６０を再構築する。再構築したターゲットプログラム１６０は、ターゲットプログラム１２２と同一である。これは、意図される結果がターゲットプログラム１２２の同一コピーなので、同一である。このデルタジェネレータシステム１００は、ソースプログラムを効率的にパッチすることによって誤りを導入することがないので、コピーは同一となる。

【００３６】

図２に、プログラムバイナリに対する最小デルタジェネレータの例示的な実施形態のためのサンプルの適用シナリオを概略的に示す。図２には、インターネットなどのネットワーク接続２１５を介してリンクされたサーバ側２１０とクライアント側２２０が示されている。サーバ側２１０は、バイナリソースプログラム（Ｓ）１１２のコピーおよびバイナリターゲットプログラム（Ｔ）１２２のコピーを有するサーバ２１２を含む。クライアント側２２０は、バイナリソースプログラム（Ｓ）１１２のコピーだけを有するクライアント２２２を含む。クライアントは最初、ターゲットプログラム（Ｔ）１２２のコピーまたはデルタ（Δ）１４２は有しない。

【００３７】

サーバ側２１０のサーバ２１２は、本明細書に述べる実施形態に従ってΔ１４２を作成する。サーバ２１２は、このようなΔをクライアント側２２０のクライアント２２２に送信する。１４２ａおよび１４２ｂ（ならびにこれらの矢印）の表現は、Δがサーバからクライアントに送信されているのを示す。

【００３８】

クライアントは、新たに到着したΔ１４２でＳ１１２をパッチして、Ｔ１２２を再構築する。したがって、例示的なデルタジェネレータの実施形態によれば、限られた帯域幅と思われるネットワーク２１５を介して最小サイズのデルタが送信され、それによりクライアントがＳをパッチしてＴを正確に再構築する。

【００３９】

用語

プログラムバイナリ（またはバイナリプログラム）の一例は、プログラムモジュールがバイナリとして表されたもの（例えばソフトウェア）である。これには、プログラムモジュールに対するソースコードは含まれない。

【００４０】

「基本ブロック」（または単に「ブロック」）は、単一の入口点を有し（外部からは最初の命令だけにしか到達できない）、最後の命令に１つの出口点を有する命令シーケンスである。したがって、基本ブロック内部には一連の制御フローがある。基本ブロックはフラグメントの一例である。具体的には、コードフラグメントの一例である。基本ブロックは、「コードブロック」と呼ぶこともできる。

【００４１】
制御フローグラフ（ＣＦＧ）は、プログラムを抽象化したものである。これは、プログラム中の基本ブロックをノードとし、ブロック間の可能なすべての制御フローをエッジ（視覚的にしばしば矢印で表される）で表した有向グラフである。図３～図５はＣＦＧを含んでいる。
【００４２】
プログラム中の隣接した静的データエリアを「データブロック」と呼び、これらもまた、ＣＦＧ中のノードを形成する。データブロックはフラグメントの一例である。具体的には、データフラグメントの一例である。
【００４３】
データブロックは、静的データ（「生」データと呼ばれる）を有し、プログラムの他の部分へのポインタを有する場合もある。これらのポインタは、例えばオブジェクトの仮想テーブルエントリとすることができる。このようなポインタは再配置可能データであり、したがって、データブロック中の他の静的データとは異なる。
【００４４】
ＣＦＧでは、データブロック中のポインタを、データブロックからターゲットブロックへの有向エッジで表す。さらに、コードブロック中の命令中にあるアドレスオペランドを、コードブロックから対応アドレスにあるブロックへのエッジとして表す。
【００４５】
データブロックの内容にはその生データが含まれるが、ブロック中のポインタは除外する。同様に、コードブロックの内容にはその命令シーケンスが含まれるが、アドレスオペランドは除外する。ＣＦＧ中の基本ブロックの内容は、そのエッジと共に、プログラムを完全に指定する。ＣＦＧ　Ｐ中に所与の任意のブロックνがある場合、その親は、集合｛χ｜χ→νがＰ中のエッジ｝であり、その子は、集合｛χ｜ν→χがＰ中のエッジ｝である。
【００４６】
本明細書で述べるＣＦＧは、プログラムのレイアウトに関する完全な情報を含む。ＣＦＧ中には、バイナリ中の関数に対応してプロシージャがあり、各プロシージャは、単一関数の基本ブロックによって誘導されるサブグラフである。関数のレイアウト（すなわちプログラムのアドレス空間における基本ブロックの構成）は、対応するプロシージャのすべてのブロックのリンクリストによって取り込まれ、したがって、リストの先頭から末尾まで走査すると、関数のレイアウト中の基本ブロックの順序が正確に記述される。
【００４７】
ツールＶが、所与のプログラムバイナリに対し、本明細書で述べるＣＦＧを生み出す。さらにＶは、プログラムに多くの種類の修正を加えることを可能にする（ＣＦＧへの）インタフェースを提供することもできる。これらの修正には、基本ブロックの追加および削除、それらの内容の修正、エッジの変更が含まれるが、これらに限定しない。
【００４８】
本明細書に述べるツールＶをどのように実装して使用するかは、当業者なら理解する。さらに、参照により組み込む特許出願には、ツールＶのようなツールを実装するのに使用できるコンポーネントが記載されている。さらに、以下の刊行物に、ツールＶが使用できるＣＦＧ方法に関する一般的な背景が提供されている。すなわち、Ａｈｏ，Ｈｏｐｃｒｏｆｔ，Ｕｌｍａｎ：「Ｐｒｉｎｃｉｐｌｅｓｏｆ　Ｃｏｍｐｉｌｅｒ　Ｄｅｓｉｇｎ」および２）Ａ．Ａｈｏ，Ｒ．Ｓｅｔｈｉ，Ｊ．Ｕｌｌｍａｎ，「Ｃｏｍｐｉｌｅｒｓ，　Ｐｒｉｎｃｉｐｌｅｓ，　Ｔｅｃｈｎｉｑｕｅｓ，ａｎｄ　Ｔｏｏｌｓ」（１９８６）である。
【００４９】
プログラムバイナリに対する最小デルタジェネレータの実施形態の１つまたは複数の例について述べる際、プログラムのソース（Ｓ）バイナリと、プログラムのターゲット（Ｔ）バイナリについて論じる。ＴとＳの関係は、次のように数学的に記述することができる。

Ｄｉｆｆ（Ｓ，Ｔ）＝Δ
Ｐａｔｃｈ（Ｓ，Δ）＝Ｔ
すなわち言い換えれば次のようになる。
ＳとＴの差異がデルタであり、
Ｓをデルタでパッチすると Ｔになる。
【００５０】
本明細書で述べるように、例示的な実施形態はΔを十分に最小化するが、Ｓをパッチして Ｔを再構築する際に完全な正確さを達成する能力は維持する。本明細書では、パッチングに関して、「完全な正確さ」（および「ほぼ同一」、または同様の用語）は、元のＴと再構築したＴの間での実質的でない差を考慮に入れている。
【００５１】
マッチング
バイナリプログラム（ＳやＴなど）の間の変更を検出するための前提は、バイナリフラグメントの一致を検出できることである。言い換えれば、プログラムモジュールをパッチできる前に、２つのバイナリプログラム間でどのフラグメントが変更されていないかを決定する必要がある。例えば、プログラムモジュール中のあるフラグメントの一致は、別のプログラムモジュール（おそらく同じモジュールの先または後のバージョン）中で、変更されていない同じフラグメントを見つけることによって検出することができる。
【００５２】
単純な例
図３～図５に、例示的なデルタジェネレータの方法論的な一実施形態の単純な一例を示す。この例では、最初の目標は、入力リストであるソース（Ｓ）３１０およびターゲット（Ｔ）３２０に対してΔ３３０を計算することである。
【００５３】
図３にソース３１０を示すが、ソース３１０は６つのノード３１１～３１６を含み、これらのノードの内容はそれぞれＨ、Ｏ、Ｏ、Ｖ、Ｅ、Ｒである。ターゲット３２０は６つのノード３２１～３２６を含み、これらのノードの内容はそれぞれＨ、Ｏ、Ｏ、Ｐ、Ｅ、Ｒである。ソース３１０と比較して、ターゲット３２０は大きく変更されていない。違うのはノード３１４（「Ｖ」を含む）であり、これは、ターゲット中で欠けているが、ノード３２４（「Ｐ」を含む）で置換されている。
【００５４】
例示的なデルタジェネレータは、変更されていないノードをマッチングする。
内容および相対的な配置に基づいて、ノード３１４および３２４を除く各ノードをマッチングする（すなわち識別する）。例えば、ノード３１１と３２１は、これらの内容（具体的には「Ｈ」）および相対位置が同一なのでマッチする。同様に、ノード３１２と３２２、ノード３１３と３２３、ノード３１５と３２５、ノード３１６と３２６がマッチする。
【００５５】
図３に示すように、デルタ３３０は、ノード３３７の内容としてノード３２４の内容（「Ｐ」）を含む。デルタ３３０はまた、この新しいノードをどのようにソース中にパッチするかを指定する「編集」も含む。具体的には、デルタ３３０は、編集ボックス３３９内で「ＡＤＤＥＤＧＥ（３１３，３３７）；ＡＤＤＥＤＧＥ（３３７，３１５）」を指定する。
【００５６】
図４に、新しいノード３３７を挿入した後でソース３１０に追加すべき２つのエッジがあることを示す。したがってデルタは、ソース３１０中のノード３１３から新しいノード３３７へのエッジ３３９ａを追加することになる編集操作ＡＤＤＥＤＧＥ（３１３，３３７）と、ノード３３７からソース中のノード３１５へのリンク３３９ｂを追加することを指定する編集操作ＡＤＤＥＤＧＥ（３３７，３１５）を含む。
【００５７】
図５では、３４０ａにソースとデルタを結合したものを示す。デルタ（新しいノード３３

７および新しい２本のエッジ（３３９ａおよび３３９ｂ））の追加により、ソースは、リストターゲット３２０の正確なコピーに変形する。ターゲットのこの正確なコピーは、再構築したターゲット３４０ｂと呼ぶことができる。

【００５８】

図４と図５を比較すると、ソースのノード３１３と３１４の間のエッジ３１４ａと、ソースのノード３１４と３１５の間のエッジ３１４ｂが削除されていることに気付く。この削除は、エッジ３３９ａおよび３３９ｂを追加することによって非明示である。

【００５９】

マッチするブロックの発見

よいブロックのマッチングは、編集操作を最小限にするマッチングである。内容の類似性だけに基づいて２つのブロックをマッチングすると、よいマッチングにならないことがある。これは、内容が同一であってそれぞれのグラフにおける場所が異なるだけのブロックが、通常、ソース中ならびにターゲット中に複数あるからである。例えば、図６に示すサブグラフを考えてみる。

【００６０】

図６には、ソース４１０およびターゲット４２０のサブグラフが示されている。この例では、内容からみて（４１２、４１６、４２２）、（４１３、４１７、４２３）、（４１４、４１８、４２４）、（４１１、４２１）、（４１５、４２５）の各グループ中のブロックはすべて同一であると仮定する。またこの例では、２つのブロックの内容が同一であればそれだけで、それらのブロックをマッチングすることができるとも仮定する。ブロック４２２をブロック４１２とマッチングする場合（両方とも「Ａ」を含む）、他のブロックの最良のマッチングは次のようになる。

ブロック４１１（D_s）とブロック４２１（D_t）
ブロック４１５（E_s）とブロック４２５（E_t）
ブロック４１３とブロック４２３（両方とも「Ｂ」を含む）
ブロック４１４とブロック４２４（両方とも「Ｃ」を含む）

【００６１】

この場合、このマッチングに従ってローカルエッジ構造が保存されるので、編集操作は必要ない。一方、ブロック４２２をブロック４１６とマッチングする場合（両方とも「Ａ」を含む）、他のブロックの最良のマッチングは次のようになる。

ブロック４１１（D_s）とブロック４２１（D_t）
ブロック４１５（E_s）とブロック４２５（E_t）
ブロック４１７とブロック４２３（両方とも「Ｂ」を含む）
ブロック４１８とブロック４２４（両方とも「Ｃ」を含む）

【００６２】

編集操作は、Ｄｅｌｅｔｅ（４１１，４１２）、Ａｄｄ（４２２，４１６）、Ｄｅｌｅｔｅ（４１６，４１９）、Ａｄｄ（４１６，４１５）である。本明細書では、Ｄｅｌｅｔｅ（ａ，ｂ）は、「ノードａからノードｂまでのエッジを削除する」ことを意味する。同様に、Ａｄｄ（ａ，ｂ）は、「ノードａからノードｂまでのエッジを追加する」ことを意味する。

【００６３】

上の例は、内容だけに基づいている。これらの例は、各ブロックの周囲のブロックを考慮していない。これらの周囲ブロックは、そのブロックの近傍または局所近傍と呼ぶことができる。これらの例は、内容だけに基づく従来のマッチング手法がよいブロックのマッチングを生み出さないことを示す。これは誤ったマッチングさえ生み出す場合もある。

【００６４】

したがって、例示的なデルタジェネレータは、ブロックをマッチングしながらブロックの近傍を考慮する。

【００６５】

近傍の考慮

例示的なデルタジェネレータは、ブロックの内容および近傍に基づいてブロックをマッチングする。実際には、ソースＣＦＧおよびターゲットＣＦＧのいくつかのパスを生み出し、各パスで近傍をサイズの大きい順に考慮する。先にサイズの大きい方の近傍に基づいてマッチングすることが望ましい。こうすることでより正確なマッチが得られる。近傍に基づくマッチを検査した後、最後のパスでブロックの内容だけに基づいてブロックをマッチングする。必要条件ではないものの、最後に内容をマッチングするのが有利である。というのは、マッチしないブロックを記録するオーバーヘッドは普通、それを同一内容の任意のブロックとマッチングした後に必要となる編集操作のサイズよりも大きいからである。

【００６６】

同一内容のブロックを検出するために、コードブロックに対しては操作コード（ｏｐｃｏｄｅ）のシーケンスを得て、データブロックに対しては生の再配置不可能データを得ることから始める。前述のように、アドレスは普通、参照先のブロックが同じのままであっても変化するので、アドレスオペランドは内容の一部と見なさない。この問題は、アドレスオペランドをＣＦＧ中のエッジと見なすことによって対処する。

【００６７】

レジスタオペランドは、扱いに注意を要することがある。いくつかのブロックを追加または削除すると、ＣＦＧの他のいくつかの近接領域にあるレジスタ割付けが、それらの領域に変更が加えられていなくても変化する場合がある。このようなオペランドは、各グラフのレジスタフロー分析を用いて、あり得るレジスタ名の変更を除外してモジュールをマッチングする。スタック変数のオフセットは普通、ローカル変数宣言の変化によって変化するので、これらは問題を呈する。この問題に対処するために、例示的なデルタジェネレータは、いくつかの即値オペランドが変化していてもブロックをマッチングする。上のそれぞれの場合では、オペランドタイプのシーケンスが同じのままであるときはコードブロックの内容は変化しないと見なす。

【００６８】

ＣＦＧ　Ｐ中のノードνのｉ近傍（すなわち局所近傍）は、Ｐに対応する無向グラフ中でνから≦ｉの距離にあるブロックの集合である。したがって、例えば０近傍は集合｛Ｖ｝であり、１近傍はνおよびその親と子である。例示的なデルタジェネレータは、ブロックに対するマッチを見つけようとする間に、異なる値のｄについてそのｄ近傍を計算する。ブロックνに対し、νからその子のうちの１つへのエッジを子エッジとし、νの親からνへのエッジを親エッジとする。

【００６９】

例示的なデルタジェネレータは、ノードνから開始するＣＦＧの幅優先走査を行いながら、νのｉ近傍を計算する。νからのｉよりも遠くなるまで、ブロックの子エッジならびに親エッジを（逆方向に）走査する。別法として、例示的なデルタジェネレータは、すべての親エッジを無視し、子エッジだけを走査して近傍を計算することもできる。

【００７０】

例えば、異なる３つのブロックb_1、b_2、b_3から呼び出されるＳ中のプロシージャｆにある最初のブロックνを考えてみる。これらすべてのブロックならびにプロシージャｆはＴ中で変更されないが、ブロックb_4からの呼出しが余分にｆに追加されると仮定する。νのｉ近傍（ｉ≧１の場合）はこの追加の呼出しによって変化し、したがって、νはこれによりマッチングされずに残ることがある。

【００７１】

しかしこの例で、ｆ自体は何の変更も受けておらず、したがってνの近傍を計算する間にνの親を無視する方が有利な場合があることを考える。時として近傍は、異なるプロシージャ中にあるブロックはどれも除外することができる。

【００７２】

一般に、マッチングの候補のあいまいさを解決する点で、考慮する近傍サイズが大きいほどマッチングは正確である。しかしこれはマッチがより少なくなり、望ましくない。したがって、例示的なデルタジェネレータは、小さい近傍サイズ（通常３つまたは２つ以下）

を、後で述べる「ランダムウォーク」を行うことによって得られるより長いランダム成分と共に用いる。この「ランダムウォーク」ヒューリスティックがよく機能することが、実験から観測された。

【００７３】

「ランダムウォーク」の一例を次に示す。ある人が、いくつかのバス停Ａ、Ｂ、．．．Ｎ（この順序で）のある、まっすぐな道でランダムウォークを行わなければならないと仮定する。この人は、Ａ（道の一端）から出発する。この人は右にしか行けないので、右すなわちＢに行かなければならない。次にＢでは、２つの選択肢がある。左（Ａ）に行くか右（Ｃ）に行くかである。統一されたランダムウォークモデルでは、公平なコインを投げて、表が出たら一方に行き、裏が出たら他方に行くと決める。到達したあらゆるバス停でこれを行い続けると、ランダムウォークを行ったことになる。

【００７４】

同じ概念を、直線に代えて一般的なグラフに容易に拡張することができる。以下は、前述の「ランダムウォーク」の例示に基づいたヒューリスティックである。無向グラフをＲとするが、この無向グラフは、ＣＦＧのノードを頂点とし、ＣＦＧ中の対応する頂点の対がいずれかの方向の（有向）制御フローエッジを有する場合に限って頂点の対の上にエッジがある。明らかに、Ｒは最大次数３のグラフである。元のノードｗから開始して、いずれかのノードｘで、ｄｘエッジのうちの１つを一様な確率でとる（ｄｘはｘの次数である）。プロシージャ境界（別のプロシージャの呼出しまたはそれへの分岐）に遭遇したときに中止するか、あるいはパスの長さが所定の限度を超えたときに中止する。

【００７５】

他のどんなブロックとも内容がマッチしないブロックｂを考えてみる。このようなブロックがいずれかのブロックνの近傍に含まれている場合、これはνのマッチングを妨げる場合がある。というのは、νを何らかのブロックとマッチングするには、その全近傍をマッチングしなければならないからである。このブロックｂを、「不良」ノードまたはアウトライア（ｏｕｔｌｉｅｒ）と呼ぶ。このようなアウトライアは、マッチングを行う前にフィルタリングして、これらの近隣すべての近傍から除去することができる。

【００７６】

例示的なデルタジェネレータは、漸進的に緩和される基準に基づいて、ＣＦＧのいくつかのパスでマッチを検出する。例示的なデルタジェネレータは事前に、マッチするブロックの数の事前推定値に基づいてＳとＴのプロシージャをマッチングする。プロシージャの例には、バイナリプログラムのルーチンおよびサブルーチンがある。

【００７７】

この事前マッチングは、大域レベルで行う（すなわち、ブロックが位置するプロシージャに関係なくブロックをマッチングする）。その後、マッチするプロシージャにブロックが属する場合に限り、これらのブロックをマッチングする。このようなマッチングはまた、局所マッチングと呼ぶこともできる。プロシージャマッチ情報を用いて呼出し命令のターゲットを検査して、異なる２つの呼出しが同じ関数（名前が変更されているか最初のブロックが異なる場合もある）をターゲットにしているかどうかを検出することもできる。

【００７８】

マッチングパスの間、現在のマッチ基準を使用して、各ノードの短いハッシュ値を計算する。ブロックνのハッシュ値は次のようにして計算する。

【００７９】

そのｄ近傍（何らかの適した値ｄに対する）中の各ブロックをその内容に基づいてハッシュして、各ブロック自体に対するラベルを作成する。

【００８０】

これらのラベルすべてを連結し、得られるストリングを再度ハッシュして、νに対する単一のラベルを作成する。これを、このブロックの「ｄラベル」と呼ぶ。

【００８１】

各グラフ中のノードをこれらのラベルによってソートし、同一のラベルおよびマッチする

プロシージャを有するブロックをマッチングする。

【００８２】

マッチングを向上させるために、他のいくつかのヒューリスティックも使用する。本明細書では、これらのヒューリスティックについては特に後述の「追加の実施形態の詳細」セクションで述べる。例えば２つのブロックc_1およびc_2が、何らかのｉに対して前述の順序付けに従ってそれぞれブロックb_1およびb_2のｉ番目の子である場合、これらのブロックをb_1およびb_2の対応子と呼ぶ。マッチングをより精密するのに使用するヒューリスティックの一例は次のとおりである。すなわち、マッチするブロックが、内容ラベルにおいてマッチする（ただしおそらくｄラベルにおいてはマッチしない）対応子を有する場合、これらの子をマッチングする。

【００８３】

この段階の終わりには、例示的なデルタジェネレータは、ＳとＴのブロックの部分的マッチングを計算し終えている。

【００８４】

編集の計算

マッチングを完了した後、次の段階はΔの計算である。図３～図５の小さな例でΔの計算を簡単に示した。図７に、Δを生み出す際に、上で計算したマッチングをどのようにグラフ（すなわちＣＦＧ）の場合に使用できるかを示す一例を示す。

【００８５】

図７には、ソース５１０およびターゲット５２０のサブグラフが示されている。マッチング段階で、５１１と５２１（「Ａ」）、５１２と５２２（［Ｂ］）、５１３と５２３（［Ｃ］）、５１６と５２６（「Ｅ」）のブロックの対がマッチしたと仮定する。ターゲットサブグラフ５２０中のマッチしなかったブロックは、ブロック５２５（F_t）およびブロック５２４（D_t）である。ブロック５２５（F_t）およびブロック５２４（D_t）の内容と、編集操作Ａｄｄ（５１２，５２５）およびＡｄｄ（５１２，５２４）とをΔが含んでいれば、ソースサブグラフ５１０からターゲットサブグラフ５２０を再構築することができる。

【００８６】

編集操作Ｄｅｌｅｔｅ（５１２，５１４）およびＤｅｌｅｔｅ（５１２，５１５）は、Δに明示的に含める必要はない。これらは非明示的に含まれる。本発明の趣旨および範囲を逸脱することなく非明示的な編集を特にΔに含めることもできることは、当業者なら理解する。しかし、非明示的な編集はΔのサイズを不必要に増大させるので、本明細書に述べる例示的な実施形態のΔには含めない。

【００８７】

したがって例示的なデルタジェネレータは、ターゲット中のマッチしなかったブロックの内容および追加する必要のあるエッジを（ある時点で）出力する。これがΔを構成する。

【００８８】

例示的なデルタジェネレータの方法論的な実施形態

図８および９に、デルタジェネレータシステム１００（またはその一部）によって行う、例示的なデルタジェネレータの方法論的な実施形態を示す。これらの方法論的な実施形態は、ソフトウェア、ハードウェア、またはこれらの組合せにおいて行うことができる。

【００８９】

図８には主に、例示的なデルタジェネレータの方法論的な実施形態の「マッチング」段階を示す。図９には主に、例示的なデルタジェネレータの方法論的な実施形態の「編集」段階を示す。「編集」段階は、「マッチング」段階の後に続く。「編集」段階の結果がデルタである。

【００９０】

図８では、６２０で、例示的なデルタジェネレータが、コードブロックに対しては操作コードのシーケンスを得て、データブロックに対しては生の再配置不可能データを得るのが示されている。このようなブロックは、ソースプログラムバイナリ（Ｓ）６１２からのプ

ロシージャのブロックと、ターゲットプログラムバイナリ（Ｔ）６１４からのプロシージャのブロックである。

【００９１】

６２４で、例示的なデルタジェネレータは、ブロックに対するマッチを見つけようとする間に、異なるｄの値についてそのｄ近傍を計算する。さらに、幅優先走査によってｄ近傍を計算する。

【００９２】

６２８で、各ノードについて短いハッシュ値を計算する。さらに、例示的なデルタジェネレータは、ノードのｄ近傍（すなわち局所近傍）でハッシュ値に基づいてノードに対するラベルを決定する。６２８ではまた、いくつかの即値オペランドが変更されている場合でも、例示的なデルタジェネレータはブロックをマッチングする。６３０で、各グラフ中のノードをこれらのラベルによってソートし、同一ラベルおよびマッチするプロシージャを有するブロックをマッチングする。この段階の終わりの６３２で、例示的なデルタジェネレータは、ＳとＴのブロックの部分的なマッチングを計算し終えている。この逆の、マッチしないブロックもまた計算する。

【００９３】

デルタジェネレータのこの方法論的な実施形態の「マッチング」段階に関するこれ以上の詳細は、後述の「追加の詳細」セクションに提供する。

【００９４】

図９では、７２０で、例示的なデルタジェネレータがP_t（すなわちターゲットプログラムバイナリからのプロシージャ）中のマッチしないブロックの内容７１０を受け取るのが示されているが、この内容７１０は、図８の方法論的な実施形態からの出力の一部である。すべてのエッジ情報はブロックの内容から除外する。したがって、内容が記録されているマッチしないブロックνがアドレスオペランド（コードブロックの場合）またはポインタ（データブロックの場合）を有する場合は、エッジ編集情報を用いてこのようなオペランドまたはポインタを訂正しなければならないことを示す「ダミー」アドレスでこれらのオペランドまたはポインタを置換する。

【００９５】

P_sおよびP_tは、マッチング段階でマッチした（ＳおよびＴそれぞれの中の）すべてのプロシージャの対を表す。ある対につき、P_sはソースからのプロシージャであり、P_tはＴからのマッチするプロシージャである。例示的なデルタジェネレータは、ソースプログラムバイナリ（Ｓ）中のプロシージャP_s７１２と、ターゲットプログラムバイナリ（Ｔ）中のプロシージャP_t７１７との対を受け取る。

【００９６】

７３０で、P_sのリンクリストからP_tのリンクリストを再構築するのに必要なすべての情報を、計算したΔの一部として記録する。プログラムのアドレス空間における各プロシージャのレイアウト（すなわちブロックの配置）をブロックのリンクリストによって取り込むが、これにより、このリストの先頭から末尾まで順次走査すると、プロシージャのレイアウトを正確に記述することができる。

P_sからP_tを再構築する際の最初のタスクは、P_sのリンクリストからP_tのリンクリストを再構築することになる。したがって、一方のリンクリストから他方のリンクリストを再構築するのに必要なすべての情報を、計算したΔの一部として記録する。

【００９７】

７３２で、エッジ編集操作を計算する。前述のように、エッジ編集操作はエッジの明示的な追加である（非明示的な削除ではない）。各エッジは、そのソースおよびターゲットによって完全に指定することができる。編集のコンテキスト内で、用語「ソース」は、このエッジが表す正確なオペランド（コードブロックの場合）または再配置可能ワード（データブロックの場合）を指す。さらに、編集のコンテキスト内で、用語「ターゲット」は一方、このオペランドまたは再配置可能ワードの参照先であるアドレスのブロックを表す。例示的なデルタジェネレータは、ソースおよびターゲットに対する固有の識別子を別々の

リストに記録することにより、編集したエッジ（追加または削除したエッジ）を記録する。

【００９８】

７３４で、ソース中の所与のプロシージャ（P_s）に対するデルタを出力する。このようなデルタは、P_t中のマッチしないブロックと７３２で計算したエッジ編集操作を含む。

【００９９】

例示的なデルタジェネレータの「編集」段階に関する前述の方法論的な実施形態は、レジスタ名、即値、および操作コードにおけるいくつかの小さい変更まで、ターゲットプログラムバイナリを再構築するのに十分な情報を記録する。変更されたこれらのレジスタ名、即値、および操作コードは、別々のリストに記録する。

【０１００】

追加の実施形態の詳細

例示的な実施形態のマッチング段階ではＣＦＧのいくつかのパスを生み出すが、以下に、各パスについてより詳細に述べる。

【０１０１】

マッチング段階における複数パス

マッチング段階における各パスは、ＳとＴの中のブロックを比較するのに異なる基準を使用するが、これは各ブロックについてハッシュ値またはラベルを計算することによって行う。例示的な実施形態では、例示的な２つのサブプロセスを採用し、これらをいくつかのパスで呼び出す。本明細書では、これらをＣｏｍｐｕｔｅＬａｂｅｌおよびＣｏｍｐｕｔｅＤＬａｂｅｌと呼ぶ。

【０１０２】

ＣｏｍｐｕｔｅＬａｂｅｌ：ブロックの内容のハッシュ値を計算する。これはブロックの内容ラベルとも呼ぶ。標準的なハッシュを使用してハッシュ値を計算する。このような標準的なハッシュの一例はＭＤ５であり、これは、所与のバイトストリングのハッシュとして１６バイトストリングを生成する。

【０１０３】

このＣｏｍｐｕｔｅＬａｂｅｌと呼ぶ例示的なサブプロセスは、異なるマッチ基準を可能にするようにパラメータ化されており、ブールパラメータは、Ｉｍｍｅｄｉａｔｅ、ＲｅｇｉｓｔｅｒＣｈａｉｎ、ＯｐｃｏｄｅＳｔｒｉｎｇ、およびＰｒｏｃＩｄである。Ｉｍｍｅｄｉａｔｅは、コードブロック中の即値オペランドをハッシングで使用すべきかどうかを示す。同様に、ＲｅｇｉｓｔｅｒＣｈａｉｎは、レジスタ名変更を使用すべきかどうかを指定し、ＯｐｃｏｄｅＳｔｒｉｎｇは、操作コードがその正確な名前で表されるか、あるいはそのグループ識別子（「グループＩＤ」）で表されるかを示す。

【０１０４】

類似する操作コードにグループＩＤを使用すると、機能に影響しない、命令の小さな変更が捉えられる。例えば、命令「ｊｇｅ　ｅａｘ，１０」が「ｊｌｅ　ｅａｘ，１０」に変わったのが唯一の変更であるＳの一部を考えてみる。これは例えば、不等式の向きが正しくないバグが発見された場合に起こり得る。このような場合、「ｊｇｅ」と「ｊｌｅ」の両方をこれらの操作コードグループ「ｂｒａｎｃｈ」で表すことができ、この結果、対応するブロックをマッチングすることができる。

【０１０５】

第４のパラメータ、ＰｒｏｃＩｄは、プロシージャマッチングがすでに行われたかどうかを示し、すでに行われた場合は、プロシージャはマッチするプロシージャに合う固有の識別子を有する。このパラメータが設定されている場合、関数呼出しのターゲットは、それらのプロシージャ識別子で表される。サブプロセスＣｏｍｐｕｔｅＬａｂｅｌに対する擬似コードの一例は、以下のとおりである。

```
ComputeLabel (Block, Immediate, RegisterChain, OpcodeString, ProcId)
If (Blockがデータブロックである)
  ブロック中の再配置可能な各ワードを反復し、クリアしてゼロにする。
  得られるデータバッファをハッシュし、ハッシュ値を返す。
Endif
If (Blockがコードブロックである)
  ハッシュすべきデータを保持できるバッファを生み出す。
  最後の命令を除く各命令ごとに:
    If (OpcodeStringがtrueである)
        バッファ中の操作コードストリングを記録する
    Else
        バッファ中の操作コードグループを記録する
    End
    命令中の各オペランドごとに:
        Case OperandType
            Pointer:定数ストリング「Block」を記録する。
            Register:If (RegisterChainがtrueである)
                        レジスタフローＩＤを記録し、レジスタフローＩＤ
を定義する（先に計算する）。
                    Else
                        レジスタ名を記録する。
                    Endif
            Immediate:If (Immediateがtrueである)
                        即値定数を記録する
                    Endif
        End
    End
End
    最後の命令に対して:
```

```
    If (OpcodeStringがtrueである)
        バッファ中の操作コードストリングを記録する
    Else
        バッファ中の操作コードグループを記録する
    End
    If (ProcIdがtrueである)
        命令のターゲットブロックを計算する。
        Ifターゲットが異なるプロシージャ中にある
            ターゲットプロシージャのプロシージャＩＤを記録する
        Endif
    Endif
  End
 バッファをハッシュしてハッシュ値を返す。
Endif
End ComputeLabel
```

【０１０６】
ＣｏｍｐｕｔｅＤＬａｂｅｌ：内容ならびに近傍に基づいてハッシュ値を計算する。これは、すでに計算した全近傍（ルートノードを含む）を検査し、これらの内容ラベルを連結する。次いで、連結したこのリストのハッシュ値を返す。
【０１０７】
この実施形態は、標準的なハッシュを使用してハッシュ値を計算する。このような標準的なハッシュの一例はＭＤ５であり、これは、所与のバイトストリングのハッシュとして１６バイトストリングを生成する。
【０１０８】
例示的なデルタジェネレータのマッチング段階の方法論的実施形態に関する追加の詳細
図１０に、例示的なデルタジェネレータのマッチング段階に関する広範な方法論的実施形態を示す。この広範な方法論的実施形態は、デルタジェネレータシステム１００（またはその一部）によって実施することができる。この広範な方法論的実施形態は、ソフトウェア、ハードウェア、またはこれらの組合せにおいて実施することができる。
【０１０９】
以下に、この広範な方法論的実施形態の追加の詳細について述べる。
タスク１（図１０の８１０）：基本ブロックの事前マッチング
このタスク（図１０の８１０）は、基本ブロックを大域レベルでマッチングする。
・例示的なデルタジェネレータは、ＳならびにＴ中の各ブロックごとにＣｏｍｐｕｔｅＬａｂｅｌを呼び出し、それに内容ラベルを割り当てる。この段階では、ＣｏｍｐｕｔｅＬａｂｅｌに対する４つのパラメータはそれぞれ偽である。　・例示的なデルタジェネレータは、計算したラベルに基づいてＳのブロックをＴのブロックとマッチングする。Ｓ中の重複ブロック（すなわち同じハッシュ値を有するブロック）は、同じラベルを有するＴ中のブロックのいずれかとマッチングする。２つのブロックをマッチングするときは常に、後で参照するために各ブロックに固有のＭａｔｃｈＩｄを割り当てる。このタスクの後でマッチしないブロックは、アウトライアとして指定する。
・例示的なデルタジェネレータは、各ブロックについて、このブロックから開始する幅優先走査を行うことによってｄ近傍（ｄ＝２）を計算する。走査では、ブロックに子がない

場合はその親に行く。
・例示的なデルタジェネレータは、上で計算した不良ノードをフィルタリングして、すべてのｄ近傍から除去する。
・例示的なデルタジェネレータは、各ブロックごとにＣｏｍｐｕｔｅＤＬａｂｅｌを呼び出し、それにｄラベルを割り当てる。ｄラベルは呼出しから返される値である。
・例示的なデルタジェネレータは、同一のｄラベルを有するＳとＴのブロックをマッチングする。
・いずれかの２つのマッチするブロックb_1およびb_2は、それから出て行くエッジ（ｏｕｔ－ｅｄｇｅ）を同数有さなければならない。これらの出て行くエッジを、ブロック中のそれらの位置によって順序付ける。２つのブロックc_1およびc_2が、何らかのｉに対して前述の順序付けに従ってそれぞれb_1およびb_2のｉ番目の子である場合、この２つのブロックをb_1およびb_2の対応子と呼ぶ。例示的なデルタジェネレータは、以下の規則を用いてマッチングをより精密にする。すなわち、マッチするブロックが、内容ラベルにおいてマッチする（ただしおそらくｄラベルにおいてはマッチしない）対応子を有する場合、これらの子をマッチングする。

【０１１０】
タスク２（図１０の８１２）：プロシージャのマッチング
このタスク（図１０の８１２）は、上で計算した事前マッチングを用いてプロシージャをマッチングする。
・例示的なデルタジェネレータは、T_s中の各プロシージャP_tを反復し、Ｓ中の各プロシージャP_sについて、P_s中のブロックにマッチするP_t中のブロックの数m_{st}を計算する。
・例示的なデルタジェネレータは、プロシージャの対をそれらのm_{st}の値に基づいてマッチングする。m_{st}の値は、プロシージャマッチ基準と呼ぶこともできる。m_{st}の値を用いてプロシージャをマッチングするこの技法については、後で考察する。
・例示的なデルタジェネレータは、マッチするプロシージャに同じ固有の識別子を割り当てる。例示的なデルタジェネレータは、他のすべてのプロシージャにも固有の識別子を割り当てる。

【０１１１】
プロシージャをマッチングするための例示的なデルタジェネレータは、次の条件を満たす。すなわち、任意のP_sと任意の２つP_{t1}およびP_{t2}があるとして、P_sをP_{t1}にマッチングする場合は$m_{st1} \geqq m_{st2}$であり、あるいはP_{t2}をP_sにマッチングするので$m_{st2} \geqq m_{st2}$である。以下の技法がこの条件を満たす。
・（P_t，P_s，m_{st}）トリプレット$\forall s, t$を生み出し、これらをm_{st}に関してソートする。
・ソートしたトリプレットリストを降順で反復する。いずれかの反復でP_tとP_sが両方ともマッチしない場合は、それらをマッチングする。

【０１１２】
タスク３（図１０の８１４）：基本ブロックの局所マッチング
このタスク（図１０の８１４）は、前のタスクで得られたプロシージャマッチング情報を用いて、ブロックの局所マッチングを計算する（すなわち、マッチするプロシージャ間だけでブロックをマッチングする）。

【０１１３】
マッチするブロックがない状態から始める。例示的なデルタジェネレータは、ＳならびにＴ中の各ブロックごとにＣｏｍｐｕｔｅＬａｂｅｌを呼び出し、それに内容ラベルを割り当てる。ＣｏｍｐｕｔｅＬａｂｅｌを呼び出すとき、Ｉｍｍｅｄｉａｔｅ、ＯｐｃｏｄｅＳｔｒｉｎｇ、およびＰｒｏｃＩｄのパラメータは真である。

【０１１４】
例示的なデルタジェネレータは、計算した内容ラベルに基づいてＳのブロックをＴのブロックとマッチングする。Ｓ中の重複ブロック（すなわち同じハッシュ値を有するブロック

）は、同じラベルを有するＴ中のブロックのいずれかとマッチングする。２つのブロックをマッチングするときは常に、後で参照するために各ブロックに固有のＭａｔｃｈＩｄを割り当てる。ブロックが前に割り当てたＭａｔｃｈＩｄをすでに有する場合は、それを再度割り当てる。このタスクの後でマッチしないブロックは、アウトライアとして指定する。

【０１１５】

ｄ＝３、２、１として、例示的なデルタジェネレータは以下のことを行う。　各ブロックについて、このブロックから開始する幅優先走査を行うことによってｄ近傍を計算する。ｄ近傍はランダム成分も有する。ランダム成分は、元のノードから固定長の一様なランダムウォークを行う間に遭遇するブロックの集合である。ランダムウォークは、実際の制御フローエッジだけを見て、一様な確率でそれらを選択する。プロシージャ境界（例えば別のプロシージャの呼出しまたはそれへの分岐）に遭遇したときに停止する。

上で計算した不良ノードをフィルタリングして、すべてのｄ近傍から除去する。

各ブロックごとにＣｏｍｐｕｔｅＤＬａｂｅｌを呼び出し、それにｄラベルを割り当てる。

同一のｄラベルを有するＳとＴのブロックをマッチングする。Ｔ中の２つのブロックがＳ中の同じブロックにマッチする場合は、それらのうちの一方だけを実際にマッチングする。タイは任意に断つことができる。

以下のガイドラインに従ってマッチングを向上させる。すなわち、マッチするブロックが、内容ラベルにおいてマッチする（ただしおそらくdラベルにおいてはマッチしない）対応子を有する場合、これらの子をマッチングする。

【０１１６】

いずれの段階でも、前にマッチしたブロックが再度マッチすること、またはマッチしないことはない。

【０１１７】

例示的なデルタジェネレータは、マッチするブロックの対にそれぞれ固有のＭａｔｃｈＩｄを割り当てる。以後これを、この２つのブロックに対する内容ラベルとして使用する。

【０１１８】

タスク４（図１０の８１６）：レジスタ名変更の問題の解決

このタスク（図１０の８１６）は、レジスタ名変更の問題を解決する（すなわち、レジスタ名変更の可能性があってもマッチするブロックを検出する）。例示的なデルタジェネレータは、ＳおよびＴ中のすべてのレジスタフロー連鎖を計算し、それぞれに識別子を割り当てる。マッチするブロックの場合は、対応するレジスタフローに同じ識別子を割り当てる。

【０１１９】

例示的なデルタジェネレータは、ＲｅｇｉｓｔｅｒＣｈａｉｎを真としてＣｏｍｐｕｔｅＬａｂｅｌを呼び出し、各ブロックに内容ラベルを割り当てる。例示的なデルタジェネレータは、これらのラベルおよびｄ近傍（ｄ＝２）に基づいて、マッチしていないブロックをマッチングする。例示的なデルタジェネレータは、マッチがそれ以上見つからなくなるまでこのタスクを繰り返す。こうするのは、新しいマッチするブロックによってさらにいくつかのレジスタ連鎖に同じ識別子が割り当てられ、さらにマッチが見つかることがあるからである。

【０１２０】

これについては後で、「レジスタ名変更の問題の解決」という題のセクションでより詳細に論じる。

【０１２１】

タスク５（図１０の８１８）：最終パス

最後のマッチングタスク（図１０の８１８）で、同一の内容ラベル（前のタスクで計算したもの）を有するブロックの対があればそれらをマッチングする。

【０１２２】

レジスタ名変更の問題の解決
例示的なデルタジェネレータは、各プロシージャについてそのｕｓｅ－ｄｅｆ連鎖、ｄｅｆ－ｕｓｅ連鎖、およびドミネータ情報を計算する。ｕｓｅ－ｄｅｆ連鎖は、レジスタｕｓｅから開始して、このｕｓｅに到達するそのレジスタのすべての定義（ｄｅｆ）を通るリストである。同様に、ｄｅｆ－ｕｓｅ連鎖は、レジスタｄｅｆから開始して、このｄｅｆにより到達する同じレジスタの各ｕｓｅを通るリストである。ブロックｄを、別のブロックｂのドミネータと呼ぶ。ｂはｄから到達可能であり、プロシージャの入口点からｂへのどんなパスも最初にｄを通過しなければならない。通常、ドミネータ情報の計算は、プロシージャ中のあらゆるブロックのすべてのドミネータを計算することを意味する。

【０１２３】
例示的なデルタジェネレータは、各プロシージャ中の各基本ブロックごとに、ブロック中のすべてのレジスタｄｅｆを見て、ｄｅｆ　ｉｄと呼ぶ各ブロック固有のＩＤをそれぞれに割り当てる。例示的なデルタジェネレータは、そのブロック中のすべてのレジスタｕｓｅを見て、いずれかのｕｓｅが同じブロック中で唯一のｄｅｆを有する場合に、そのｕｓｅにｄｅｆ識別子を割り当てる。したがってこの段階では、すべてのｄｅｆおよびいくつかのｕｓｅにＩＤが関連付けられている。未定義のＩＤを有するレジスタｕｓｅがいくつかある場合もある。ＣｏｍｐｕｔｅＬａｂｅｌ（上で定義した）がＲｅｇｉｓｔｅｒＣｈａｉｎ＝ｔｒｕｅを伴って呼び出されたとき、ＩＤが定義されていない場合はそれらの名前を使用する。

【０１２４】
例示的なデルタジェネレータは、ＳおよびＴ中のマッチしない各ブロックごとにＣｏｍｐｕｔｅＬａｂｅｌ（ＲｅｇｉｓｔｅｒＣｈａｉｎ＝ｔｒｕｅを伴う）を呼び出し、返されるラベルをその内容ラベルに割り当てる。例示的なデルタジェネレータは、マッチしない各ブロックについてｄ近傍（ｄ＝２）を計算する。
例示的なデルタジェネレータは、ＣｏｍｐｕｔｅＤＬａｂｅｌを呼び出してそれにｄラベルを割当て、新たに計算したｄラベルに基づいてＳとＴをマッチングする。

【０１２５】
コードブロックの現在マッチングしているある対（b_s、b_t）に対して、例示的なデルタジェネレータは以下のことを行う。

【０１２６】
２つのブロック中の対応する（同じ場所にある）レジスタｄｅｆを反復する。

【０１２７】
プロシージャ全体で固有の固有識別子（正）を、現在検査中の２つのｄｅｆそれぞれに割り当てる。

【０１２８】
これらの識別子は、そのｄｅｆが有することのある、前に割り当てたどんなｄｅｆ識別子にも置き換わる。２つのｄｅｆがマッチすることを記録し、これらの両方に単一の識別子を関連付ける。

【０１２９】
次に、例示的なデルタジェネレータは、新たに割り当てたｄｅｆ識別子に基づいてレジスタｕｓｅをマッチングする。例示的なデルタジェネレータは、ＳとＴの両方に対して各基本ブロック中のすべてのレジスタｕｓｅを反復する。このような各ｕｓｅについて、それに到達しそれに支配されないすべてのｄｅｆを検査する。例示的なデルタジェネレータは、それらのｄｅｆ識別子を合計し、合計を現在のｕｓｅの識別子に割り当てる。したがって、各レジスタオペランド、ｄｅｆまたはｕｓｅに識別子が関連付けられる。マッチするブロックは、対応するレジスタオペランドに関連するマッチング識別子を有する。より具体的には、正確にマッチするｄｅｆを有する２つのレジスタオペランドは、レジスタ名が異なる場合があっても、マッチング識別子も有することができる。

【０１３０】
例示的なデルタジェネレータがＲｅｇｉｓｔｅｒＣｈａｉｎ＝ｔｒｕｅを伴ってＣｏｍｐ

ｕｔｅＬａｂｅｌを呼び出すこと（前述）を一番最近に実行した際に新しいマッチが発見された場合、プロセスはそこにループバックする。

【０１３１】

例示的なデルタジェネレータの編集段階の方法論的な実施形態に関する追加の詳細

例示的なデルタジェネレータは、ｂｉｎと呼ぶ新しいダミーＣＦＧを生み出すが、このｂｉｎは、Ｔのすべての追加ブロックのためのコンテナとして働くことのできる単一プロシージャを有する。これらのブロックは、例えばリンクリストの形で維持する。

【０１３２】

例示的なデルタジェネレータは、Ｔ中の各ブロックを順次走査し、マッチしない各ブロックに連続した識別子（０で始まる）を割り当てる。例示的なデルタジェネレータはまた、マッチしないブロックをｂｉｎにダンプして、コードブロック中のアドレスオペランドおよびデータブロック中のポインタが、これらのポインタを含むブロックを参照する「ダミー」アドレスに確実に修正されるようにする。

【０１３３】

例示的なデルタジェネレータは、Ｓ中の各ブロックを順次走査し、マッチする各ブロックに連続した識別子を割り当てる。同じ識別子を、Ｔ中のマッチするブロックに割り当てる。このときおよび前のタスクで割り当てる識別子をＮｅｗ＿Ｎｏｄｅ＿Ｉｄと呼ぶ。

【０１３４】

例示的なデルタジェネレータは、ソースグラフ中のマッチするブロックを検査する。Ｓ中のブロックｘがＴ中のブロックｙとマッチする場合、それぞれから出るエッジをチェックする。各エッジ$x \to z_1$につき、対応する$y \to z_2$がなければならない。

ケースｉ　z_2がz_1とマッチする：何もしない。

ケースｉｉ　z_2が$z_3 \neq z_2$とマッチする：リンク$x \to z_3$を記録する。

ケースｉｉｉ　z_2がマッチしない：リンク$x \to z_2$を記録する。

【０１３５】

エッジは次のように記録する。ＴａｒｇｅｔＩｄ、ＯｐｅｒａｎｄＩｎｄｅｘ、およびＳｏｕｒｃｅＰｒｏｃＦｉｒｓｔＢｌｏｃｋという名前の３つのリストを生み出す。エッジｘ→ｙを記録する場合、ｙのＮｅｗ＿Ｎｏｄｅ＿ＩｄをＴａｒｇｅｔＩｄに挿入する。このエッジが現プロシージャのｉ番目のエッジであった場合、ｉをリストＯｐｅｒａｎｄＩｎｄｅｘにプッシュする。最後に、Ｓ中のいずれかのプロシージャP_sのエッジが少なくとも１つ記録されている場合、その最初のブロックのＮｅｗ＿Ｎｏｄｅ＿ＩｄをＳｏｕｒｃｅＰｒｏｃＦｉｒｓｔＢｌｏｃｋにプッシュする。

【０１３６】

例示的なデルタジェネレータは、ターゲットグラフ中のマッチしないブロックすべてを検査する。例示的なデルタジェネレータは、それぞれから出るエッジｘ→ｙを見る。ｙもまたマッチしないブロックである場合は、ｘからｙに行くエッジをｂｉｎ中に生み出す。ｙがソースグラフのｚとマッチする場合は、ｚのＮｅｗ＿Ｎｏｄｅ＿ＩｄをＴａｒｇｅｔＯｐｅｒａｎｄＴａｒｇｅｔｓと呼ぶリストにプッシュすることにより、リンクｘ→ｚを記録する。

【０１３７】

例示的なデルタジェネレータは、Ｔ中のマッチするプロシージャP_tを有するＳ中の各プロシージャP_sごとに、P_tのブロックのリンクリストを走査する。

走査のいずれかの時点で、ptr_1およびptr_2がＴ中の現ノードおよび次のノードを指す場合、$mptr_1$および$mptr_2$をこれらの「マッチング」ポインタとする。ptr_iがＴ中のブロックｘを指す場合、そのマッチングポインタ$mptr_i$は、Ｓ中にマッチするブロックがあればそれを指すか、あるいはｂｉｎ中にダンプしたｘのコピーを指す。走査のいずれかの段階で、$mptr_1$および$mptr_2$がリスト中で（この順序で）隣接しないノードを指す場合、エッジ$mptr_1 \to mptr_2$を記録する。これは、これらに対応するブロックのＮｅｗ＿Ｎｏｄｅ＿ＩｄをＬｉｎｋｅｄＬｉｓｔｓＬｅｆｔおよびＬｉｎｋｅｄＬｉｓｔｓＲｉｇｈｔと呼ぶ別々のリストにプッシュすることによって行う

。さらに、P_tの最初のブロックのＮｅｗ＿Ｎｏｄｅ＿Ｉｄを、後の部分のサイズ（ブロック数で表す）およびP_t中の記録済みリンクの数と共に、ＰｒｏｃｅｄｕｒｅＩｎｆｏと呼ぶ別個のリストに記録する。

【０１３８】

例示的なデルタジェネレータは、上で計算した編集情報をＳと共に用いてＴを再構築する。

【０１３９】

別法として、例示的なデルタジェネレータは、元バージョンのＴと再構築バージョンのＴとで異なるレジスタ名、即値、および操作コードを検出し、これらの差異をＭｉｎｏｒＤｉｆｆｅｒｅｎｃｅｓと呼ぶ別個のリストに出力することもできる。

【０１４０】

小さな差異

デルタジェネレータの例示的な一実施形態では、これらのリストすべてがｂｉｎと共に、計算されたΔを構成する。

【０１４１】

デルタ圧縮

以上のセクションは、ＳおよびΔがある場合に例示的なデルタジェネレータを使用してどのようにＴのコピーを再構築するかを十分に述べている。図２に示すように、生成したΔはサーバ２１２からクライアント２２２に送信することができる。

【０１４２】

このΔは、ネットワーク２１５上で利用可能な帯域幅をより効率的に使用するために圧縮することができる。したがって例示的なデルタジェネレータは、パッチをよく圧縮できる方式でフォーマットすることができる。パッチはいくつかの部分に分割するが、それぞれの部分は異なる種類の情報を保持し、したがって各部分は別々に圧縮することができる。この手法は、異なるタイプのデータ（例えばコードに対して整数リスト）は異なる圧縮アルゴリズム（すなわちエンジン）で最もよく圧縮できるという観測結果に基づく。

【０１４３】

この手法は、先に「追加の実施形態の詳細」セクションで述べたステップと組み合わせることができる。例示的なデルタジェネレータにおいて、本明細書でいう「いくつかの部分」とは、前述の編集段階で言及した異なるいくつかのリスト、およびそこで言及したダミー「ｂｉｎ」である。一実施形態では、これらの各リスト（およびｂｉｎ）を別々のファイルにストアし、したがって別々に圧縮する。使用できる圧縮アルゴリズムの一例は、ＬＺＷである。

【０１４４】

これは、大きな実行可能ファイル（約数百キロバイト）の場合によりよく機能すると思われ、我々の最初の実験では、このような大きな実行可能ファイルの場合に、例示的なデルタジェネレータは平均して、我々の知る利用可能な他のどんなパッチングツールよりも小さなパッチを作成することが示されている。

【０１４５】

例示的な実施形態に対する適用例

例示的なデルタジェネレータの実施形態に対して考えられる適用例には、帯域幅の制約のある媒体を介してソフトウェアのアップグレードまたはパッチを送信するためのツールとしての適用例がある。インターネット上でソフトウェアを入手できることが日々ますます一般的になりつつある中で、このようなパッチングツールはソフトウェアの使用にとって有益であろう。

【０１４６】

例示的なデルタジェネレータの実施形態に対して考えられる別の適用例は、ソフトウェアの海賊行為防止の分野にある。例示的なデルタジェネレータの一実施形態を用いれば、海賊行為防止ツールがプログラムの類似性を検出することができる。考えられる別の適用例は、ソフトウェアの異なるバージョン間の変更の性質を分析することにあろう。

【０１４７】
例示的なコンピューティング環境
図１１に、例示的なデルタジェネレータを例えば実装できる、好適なコンピューティング環境９２０の一例を示す。
【０１４８】
例示的なコンピューティング環境９２０は、好適なコンピューティング環境の一例でしかなく、例示的なデルタジェネレータの使用または機能の範囲に関するいかなる制限も示すものではない。コンピューティング環境９２０は、例示的なコンピューティング環境９２０中に示すコンポーネントのいずれか１つまたは組合せに関して、いかなる依存または要件も有すると解釈すべきではない。
【０１４９】
例示的なデルタジェネレータは、他の多くの汎用または専用コンピューティングシステム環境または構成で動作可能である。例えば例示的なデルタジェネレータと共に使用するのに適した、周知のコンピューティングシステム、環境、および／または構成の例としては、パーソナルコンピュータ、サーバコンピュータ、シン（ｔｈｉｎ）クライアント、シック（ｔｈｉｃｋ）クライアント、ハンドヘルドまたはラップトップデバイス、マルチプロセッサシステム、マイクロプロセッサベースのシステム、セットトップボックス、プログラム可能な消費者電子機器、ネットワークＰＣ、ミニコンピュータ、メインフレームコンピュータ、前述のシステムまたはデバイスのいずれかを含む分散コンピューティング環境などが挙げられるが、これらに限定されない。
【０１５０】
例示的なデルタジェネレータは、例えば、コンピュータによって実行されるプログラムモジュールなどのコンピュータ実行可能命令の一般的なコンテキストで述べる。一般にプログラムモジュールは、特定のタスクを実行するか特定の抽象データ型を実装するルーチン、プログラム、オブジェクト、コンポーネント、データ構造などを含む。例示的なデルタジェネレータはまた、通信ネットワークを介してリンクされたリモート処理装置によってタスクが実行される分散コンピューティング環境で実施することもできる。分散コンピューティング環境では、プログラムモジュールは例えば、メモリ記憶装置を含めたローカルとリモートの両方の記憶媒体中にある。
【０１５１】
図１１に示すように、コンピューティング環境９２０は、コンピュータ９３０の形をとる汎用コンピューティングデバイスを含む。コンピュータ９３０のコンポーネントには、１つまたは複数のプロセッサまたはプロセッシングユニット９３２と、システムメモリ９３４と、システムメモリ９３４を含めた種々のシステムコンポーネントをプロセッサ９３２に結合するバス９３６を含めることができるが、これらに限定しない。
【０１５２】
バス９３６は、メモリバスまたはメモリコントローラ、周辺バス、ＡＧＰ（ａｃｃｅｌｅｒａｔｅｄ ｇｒａｐｈｉｃｓ ｐｏｒｔ）、および種々のバスアーキテクチャのいずれかを使用するプロセッサバスまたはローカルバスを含めた、いくつかのタイプのバス構造のうちのいずれか１つまたは複数を表す。限定ではなく例として、このようなアーキテクチャには、ＩＳＡ（Ｉｎｄｕｓｔｒｙ Ｓｔａｎｄａｒｄ Ａｒｃｈｔｅｃｔｕｒｅ）バス、ＭＣＡ（Ｍｉｃｒｏ Ｃｈａｎｎｅｌ Ａｒｃｈｔｅｃｔｕｒｅ）バス、ＥＩＳＡ（Ｅｎｈａｎｃｅｄ ＩＳＡ）バス、ＶＥＳＡ（Ｖｉｄｅｏ Ｅｌｅｃｔｒｏｎｉｃｓ Ｓｔａｎｄａｒｄｓ Ａｓｓｏｃｉａｔｉｏｎ）ローカルバス、およびメザニンバスとも呼ばれるＰＣＩ（Ｐｅｒｉｐｈｅｒａｌ Ｃｏｍｐｏｎｅｎｔ Ｉｎｔｅｒｃｏｎｎｅｃｔｓ）バスが含まれる。
【０１５３】
コンピュータ９３０は通常、種々のコンピュータ可読媒体を備える。このような媒体は、例えばコンピュータ９３０からアクセス可能な任意の利用可能媒体であり、揮発性媒体と不揮発性媒体、取外し可能媒体と取外し不可能媒体の両方がこれに含まれる。

【０１５４】
図１１では、システムメモリは、ランダムアクセスメモリ（ＲＡＭ）９４０などの揮発性メモリ、および／または、読出し専用メモリ（ＲＯＭ）９３８などの不揮発性メモリの形をとるコンピュータ可読媒体を含む。ＲＯＭ９３８には、起動中などにコンピュータ９３０内の要素間で情報を転送するのを補助する基本ルーチンを含むＢＩＯＳ（ｂａｓｉｃ ｉｎｐｕｔ／ｏｕｔｐｕｔ ｓｙｓｔｅｍ）９４２がストアされる。ＲＡＭ９４０は通常、プロセッサ９３２からすぐにアクセス可能な、かつ／またはプロセッサ９３２が現在作用している、データおよび／またはプログラムモジュールを含む。
【０１５５】
コンピュータ９３０はさらに、その他の取外し可能／取外し不可能、かつ揮発性／不揮発性のコンピュータ記憶媒体を備えることもできる。例として示すだけだが、図１１には、取外し不可能かつ不揮発性の磁気媒体（図示していないが通常「ハードドライブ」と呼ばれる）に対して読み書きするためのハードディスクドライブ９４４と、取外し可能かつ不揮発性の磁気ディスク９４８（例えば「フロッピー（登録商標）ディスク」）に対して読み書きするための磁気ディスクドライブ９４６と、ＣＤ－ＲＯＭ、ＤＶＤ－ＲＯＭ、またはその他の光学媒体など取外し可能かつ不揮発性の光ディスク９５２に対して読み書きするための光ディスクドライブ９５０とが示されている。ハードディスクドライブ９４４、磁気ディスクドライブ９４６、および光ディスクドライブ９５０はそれぞれ、１つまたは複数のインタフェース９５４によってバス９３６に接続される。
【０１５６】
ドライブおよびそれらに関連するコンピュータ可読媒体は、コンピュータ可読命令、データ構造、プログラムモジュール、およびコンピュータ９３０に対するその他のデータの、不揮発性記憶域を提供する。本明細書に述べる例示的な環境は、ハードディスク、取外し可能磁気ディスク９４８、および取外し可能光ディスク９５２を採用するが、磁気カセット、フラッシュメモリカード、ディジタルビデオディスク、ランダムアクセスメモリ（ＲＡＭ）、読出し専用メモリ（ＲＯＭ）など、コンピュータからアクセス可能なデータをストアすることのできる他のタイプのコンピュータ可読媒体をこの例示的な動作環境で使用することもできることを、当業者は理解されたい。
【０１５７】
ハードディスク、磁気ディスク９４８、光ディスク９５２、ＲＯＭ９３８、またはＲＡＭ９４０には、いくつかのプログラムモジュールが例えばストアされるが、これらのプログラムモジュールには、限定ではなく例として、オペレーティングシステム９５８、１つまたは複数のアプリケーションプログラム９６０、その他のプログラムモジュール９６２、およびプログラムデータ９６４が含まれる。
【０１５８】
このようなオペレーティングシステム９５８、１つまたは複数のアプリケーションプログラム９６０、その他のプログラムモジュール９６２、およびプログラムデータ９６４（またはこれらの何らかの組合せ）はそれぞれ、例示的なデルタジェネレータの一実施形態を含むことができる。より具体的には、これらはそれぞれ、デルタジェネレータシステム、コンパレータ、編集操作決定機構、および出力サブシステムの一実施形態を含むことができる。
【０１５９】
ユーザは、キーボード９６６やポインティングデバイス（「マウス」など）９６８などの入力デバイスを介してコンピュータ９３０にコマンドおよび情報を入力することができる。その他の入力デバイス（図示せず）には、マイクロホン、ジョイスティック、ゲームパッド、衛星受信アンテナ、シリアルポート、スキャナなどを含めることができる。これらおよび他の入力デバイスは、バス９３６に結合されたユーザ入力インタフェース９７０を介してプロセッシングユニット９３２に接続されるが、例えばパラレルポート、ゲームポート、ユニバーサルシリアルバス（ＵＳＢ）など、他のインタフェースおよびバス構造でも接続される。

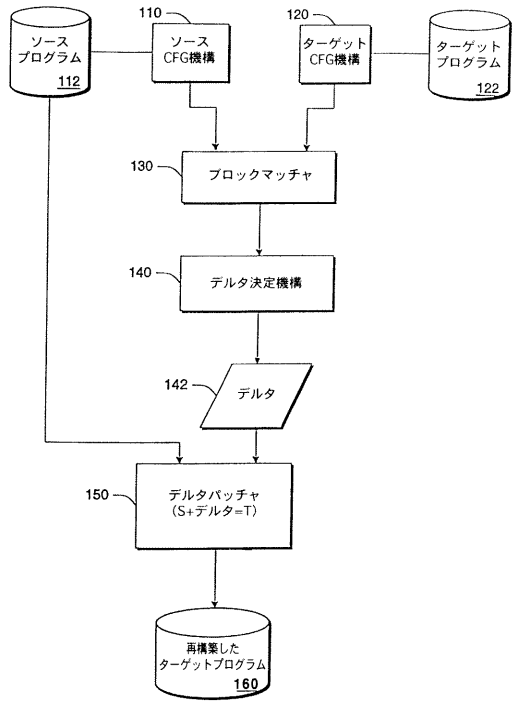
【０１６０】
モニタ９７２または他のタイプの表示装置もまた、ビデオアダプタ９７４などのインタフェースを介してバス９３６に接続される。モニタに加え、パーソナルコンピュータは通常、スピーカやプリンタなど他の周辺出力デバイス（図示せず）も備え、これらは出力周辺インタフェース９７５を介して接続される。
【０１６１】
コンピュータ９３０は、リモートコンピュータ９８２など１つまたは複数のリモートコンピュータへの論理接続を用いて、ネットワーク化された環境で動作することができる。リモートコンピュータ９８２は、コンピュータ９３０に関して本明細書に述べた要素および機構の多くまたはすべてを備えることができる。
【０１６２】
図１１に示す論理接続は、ローカルエリアネットワーク（ＬＡＮ）９７７および一般的なワイドエリアネットワーク（ＷＡＮ）９７９である。このようなネットワーキング環境は、オフィス、企業全体のコンピュータネットワーク、イントラネット、およびインターネットによくあるものである。
【０１６３】
ＬＡＮネットワーキング環境でコンピュータ９３０を使用するときは、ネットワークインタフェースまたはアダプタ９８６を介してコンピュータ９３０をＬＡＮ９７７に接続する。ＷＡＮネットワーキング環境で使用するときは、コンピュータは通常、モデム９７８を備えるか、あるいはＷＡＮ９７９を介して通信を確立するための他の手段を備える。モデム９７８は、例えば内蔵または外付けであり、例えばユーザ入力インタフェース９７０または適した他の機構を介してシステムバス９３６に接続される。
【０１６４】
図１１には、インターネットを介したＷＡＮの具体的な実施形態が示されている。コンピュータ９３０は通常、モデム９７８を備えるか、あるいはインターネット９８０を介して接続を確立するための他の手段を備える。モデム９７８は、例えば内蔵または外付けであり、インタフェース９７０を介してバス９３６に接続される。
【０１６５】
ネットワーク化された環境では、パーソナルコンピュータ９３０に関して示したプログラムモジュールまたはこれらの一部が、リモートメモリ記憶装置に例えばストアされる。図１１には、限定ではなく例として、リモートコンピュータ９８２のメモリ装置上にあるリモートアプリケーションプログラム９８９が示されている。図示および記述したネットワーク接続は例示的なものであり、コンピュータ間に通信リンクを確立する他の手段も例えば使用されることを理解されたい。
【０１６６】
例示的な動作環境
図１１には、例示的なデルタジェネレータを実装できる、適した動作環境９２０の一例が示されている。具体的には、図１１中のいずれかのプログラムモジュール９６０～９６２および／またはオペレーティングシステム９５８、あるいはこれらの一部により、本明細書に述べた例示的なデルタジェネレータが（全体的または部分的に）実装される。
【０１６７】
この動作環境は、適した動作環境の一例でしかなく、本明細書に述べる例示的なデルタジェネレータの機能の範囲または使用に関するいかなる制限も示すものではない。例示的なデルタジェネレータとともに使用するのに適した他の周知のコンピューティングシステム、環境、および／または構成には、パーソナルコンピュータ（ＰＣ）、サーバコンピュータ、ハンドヘルドまたはラップトップデバイス、マルチプロセッサシステム、マイクロプロセッサベースのシステム、プログラム可能な消費者電子機器、無線電話機および無線機器、汎用および専用アプライアンス、特定用途向けＩＣ（ＡＳＩＣ）、ネットワークＰＣ、ミニコンピュータ、メインフレームコンピュータ、以上のシステムまたはデバイスのいずれかを含む分散コンピューティング環境などが挙げられるが、これらに限定しない。

【０１６８】
コンピュータ実行可能命令
例示的なデルタジェネレータの一実施形態は、１つまたは複数のコンピュータまたはその他のデバイスによって実行される、プログラムモジュールなどのコンピュータ実行可能命令の一般的なコンテキストで述べることができる。一般にプログラムモジュールは、特定のタスクを実行するか特定の抽象データ型を実装するルーチン、プログラム、オブジェクト、コンポーネント、データ構造などを含む。通常、プログラムモジュールの機能は、所望の種々の実施形態で組み合わせることも分散させることもできる。
【０１６９】
コンピュータ可読媒体
例示的なデルタジェネレータの一実施形態は、何らかの形のコンピュータ可読媒体上にストアすることもでき、それを介して伝送することもできる。コンピュータ可読媒体は、コンピュータからアクセス可能な任意の利用可能媒体とすることができる。限定ではなく例として、コンピュータ可読媒体には、「コンピュータ記憶媒体」および「通信媒体」を含めることができる。
【０１７０】
「コンピュータ記憶媒体」には、コンピュータ可読命令、データ構造、プログラムモジュール、またはその他のデータなどの情報をストアするために任意の方法または技術で実装される、揮発性および不揮発性、取外し可能および取外し不可能媒体が含まれる。コンピュータ記憶媒体には、ＲＡＭ、ＲＯＭ、ＥＥＰＲＯＭ、フラッシュメモリ、またはその他のメモリ技術や、ＣＤ－ＲＯＭ、ＤＶＤ（ｄｉｇｉｔａｌ　ｖｅｒｓａｔｉｌｅ　ｄｉｓｋ）、またはその他の光学記憶装置や、磁気カセット、磁気テープ、磁気ディスク記憶装置、またはその他の磁気記憶装置や、所望の情報をストアするのに使用できコンピュータからアクセスできるその他の任意の媒体が含まれるが、これらに限定しない。
【０１７１】
「通信媒体」は通常、コンピュータ可読命令、データ構造、プログラムモジュール、またはその他のデータを、搬送波やその他の搬送機構など変調されたデータ信号に具現化する。通信媒体にはまた、任意の情報送達媒体も含まれる。
【０１７２】
用語「変調されたデータ信号」は、情報が信号に符号化される形でその１つまたは複数の特徴が設定または変更された信号を意味する。限定ではなく例として、通信媒体には、有線のネットワークや直接有線接続などの有線媒体と、音波、ＲＦ、赤外線などの無線媒体と、その他の無線媒体が含まれる。これらの任意の組合せもまた、コンピュータ可読媒体の範囲に含まれる。
【０１７３】
結び
プログラムバイナリに対する最小デルタジェネレータについて、構造上の特徴および／または方法論的ステップに特定した言葉で述べたが、頭記の特許請求の範囲に定義するプログラムバイナリに対する最小デルタジェネレータは、上述した特定の特徴またはステップに必ずしも限定しない。むしろ、これらの特定の特徴およびステップは、特許請求する本発明の実施の好ましい形として開示するものである。
【図面の簡単な説明】
【図１】プログラムバイナリに対する最小デルタジェネレータの一実施形態を示す概略ブロック図である。
【図２】プログラムバイナリに対する最小デルタジェネレータの別の実施形態を示す概略ブロック図である。
【図３】プログラムバイナリに対する最小デルタジェネレータの一実施形態の単純化した適用例を示す図である。
【図４】プログラムバイナリに対する最小デルタジェネレータの一実施形態の単純化した適用例を示す図である。

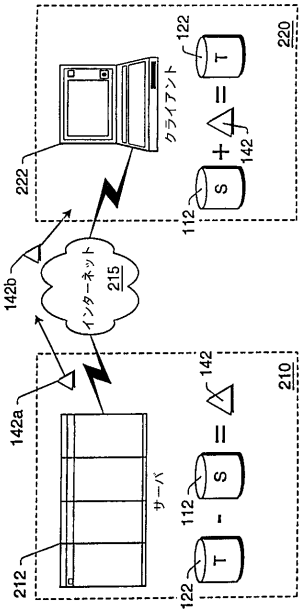
【図５】プログラムバイナリに対する最小デルタジェネレータの一実施形態の単純化した適用例を示す図である。
【図６】プログラムバイナリに対する最小デルタジェネレータの実施形態の適用例を示す図である。
【図７】プログラムバイナリに対する最小デルタジェネレータの実施形態の適用例を示す図である。
【図８】プログラムバイナリに対する最小デルタジェネレータの方法論的な一実施形態を示す流れ図である。
【図９】プログラムバイナリに対する最小デルタジェネレータの別の方法論的な実施形態を示す流れ図である。
【図１０】プログラムバイナリに対する最小デルタジェネレータの別の方法論的な実施形態を示す流れ図である。
【図１１】プログラムバイナリに対する最小デルタジェネレータの一実施形態を実装できるコンピューティング動作環境の一例を示す図である。
【符号の説明】
１００　Δジェネレータシステム
１１０　ソースＣＦＧ機構
１１２　ソースプログラム
１２０　ターゲットＣＦＧ機構
１２２　ターゲットプログラム
１３０　ブロックマッチャ
１４０　デルタ決定機構
１４２　デルタ
１５０　デルタパッチャ
１６０　再構築したターゲットプログラム
２１０　サーバ側
２１２　サーバ
２１５　インターネット
２２０　クライアント側
２２２　クライアント
９２０　コンピューティング環境
９３０　コンピュータ
９３２　プロセッサまたはプロセッシングユニット
９３４　システムメモリ
９３６　バス
９３８　ＲＯＭ
９４０　ＲＡＭ
９４２　ＢＩＯＳ
９４４　ハードディスクドライブ
９４６　磁気ディスクドライブ
９４８　磁気ディスク
９５０　光ディスクドライブ
９５２　光ディスク
９５４　データ媒体インタフェース
９５８　オペレーティングシステム
９６０　アプリケーションプログラム
９６４　プログラムデータ
９６６　キーボード
９６８　ポインティングデバイス
９７０　ユーザ入力インタフェース

９７２　モニタ
９７４　ビデオアダプタ
９７５　出力周辺インタフェース
９７７　ＬＡＮ
９７８　モデム
９７９　ＷＡＮ
９８０　インターネット
９８２　リモートコンピュータ
９８６　ネットワーク
９８９　リモートアプリケーション

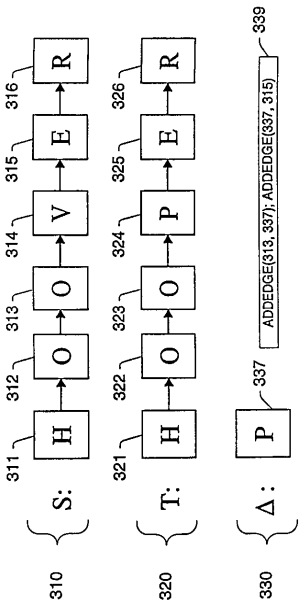
【図１】



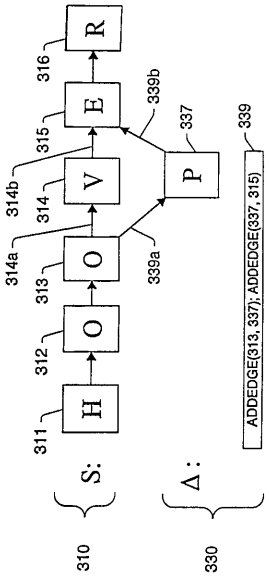
【図２】



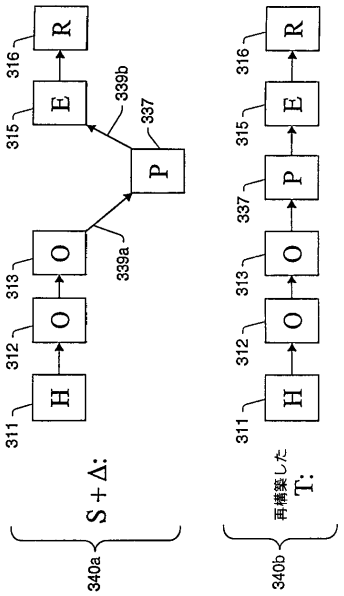
【図３】


S:
T:
Δ:
H
O
V
E
P
R
ADDEDGE(313, 337); ADDEDGE(337, 315)
311
312
313
314
315
316
321
322
323
324
325
326
337
339
310
320
330

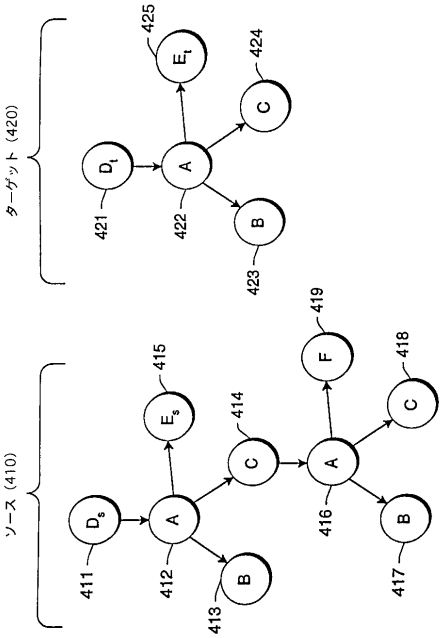
【図４】


S:
Δ:
H
O
V
E
P
R
ADDEDGE(313, 337); ADDEDGE(337, 315)
311
312
313
314a
314b
315
316
339a
339b
337
339
310
330

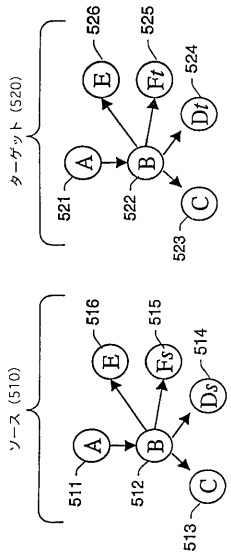
【図５】


S + Δ:
再構築した
T:
H
O
E
P
R
311
312
313
315
316
337
339a
339b
340a
340b

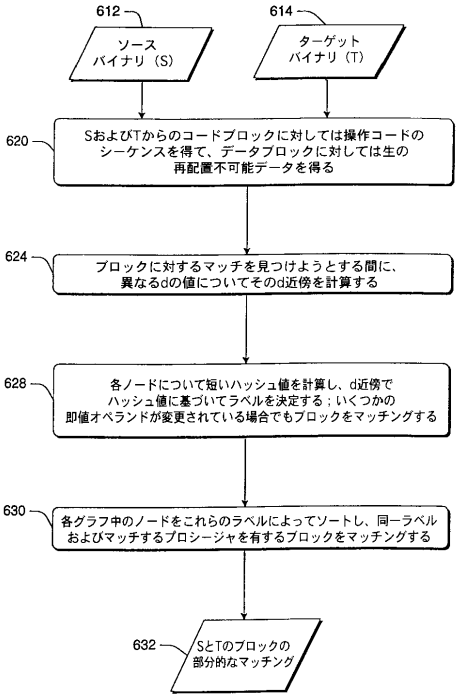
【図６】


ターゲット (420)
ソース (410)
A
B
C
F
411
412
413
414
415
416
417
418
419
421
422
423
424
425

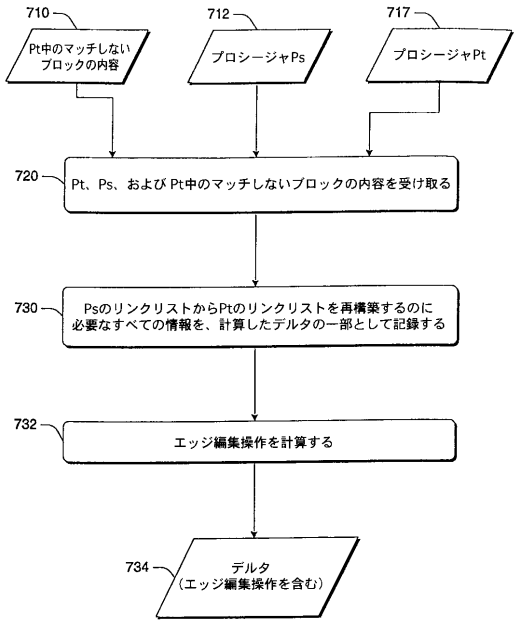
【図７】


ターゲット（520）
526
525
524
E
Ft
Dt
A
B
C
521
522
523
ソース（510）
516
515
514
E
Fs
Ds
A
B
C
511
512
513

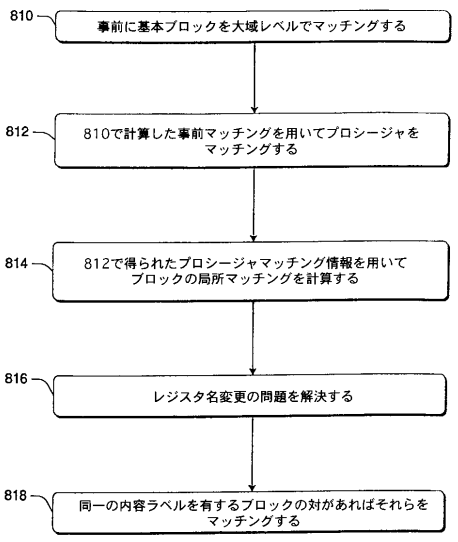
【図８】


612
ソース
バイナリ（S）
614
ターゲット
バイナリ（T）
620
SおよびTからのコードブロックに対しては操作コードのシーケンスを得て、データブロックに対しては生の再配置不可能データを得る
624
ブロックに対するマッチを見つけようとする間に、異なるdの値についてそのd近傍を計算する
628
各ノードについて短いハッシュ値を計算し、d近傍でハッシュ値に基づいてラベルを決定する；いくつかの即値オペランドが変更されている場合でもブロックをマッチングする
630
各グラフ中のノードをこれらのラベルによってソートし、同一ラベルおよびマッチするプロシージャを有するブロックをマッチングする
632
SとTのブロックの部分的なマッチング

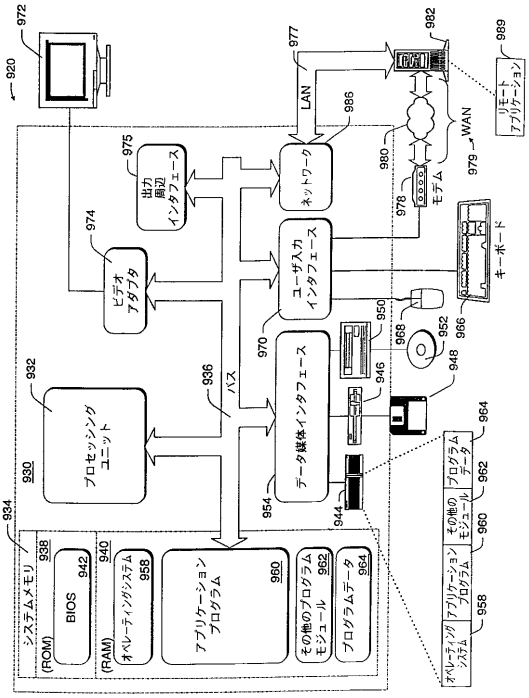
【図９】


710
Pt中のマッチしないブロックの内容
712
プロシージャPs
717
プロシージャPt
720
Pt、Ps、およびPt中のマッチしないブロックの内容を受け取る
730
PsのリンクリストからPtのリンクリストを再構築するのに必要なすべての情報を、計算したデルタの一部として記録する
732
エッジ編集操作を計算する
734
デルタ
（エッジ編集操作を含む）

【図１０】


810
事前に基本ブロックを大域レベルでマッチングする
812
810で計算した事前マッチングを用いてプロシージャをマッチングする
814
812で得られたプロシージャマッチング情報を用いてブロックの局所マッチングを計算する
816
レジスタ名変更の問題を解決する
818
同一の内容ラベルを有するブロックの対があればそれらをマッチングする

【図１１】


920
972
977
LAN
982
989
リモート
アプリケーション
986
975
出力
周辺
インタフェース
ネットワーク
980
WAN
979
978
モデム
974
ビデオ
アダプタ
ユーザ入力
インタフェース
キーボード
950
968
952
966
970
データ媒体インタフェース
946
948
932
プロセッシング
ユニット
936
バス
930
964
962
960
958
954
944
934
システムメモリ
938
(ROM)
BIOS
942
940
(RAM)
オペレーティングシステム
958
アプリケーション
プログラム
960
その他の
モジュール
962
プログラム
データ
964
オペレーティング
システム
アプリケーション
プログラム
その他の
モジュール
プログラム
データ

フロントページの続き

(72)発明者　ラマラスナム　ベンカテサン
アメリカ合衆国　９８０５２　ワシントン州　レッドモンド　ノースイースト　２２　コート　１７２０８
(72)発明者　サウラブ　シンハ
アメリカ合衆国　９８１０３　ワシントン州　シアトル　オーロラ　アベニュー　ノース　４７１０　ナンバー２０２

合議体
審判長　赤川　誠一
審判官　宮司　卓佳
審判官　石井　茂和

(56)参考文献　国際公開第９９／１１５４９号パンフレット（WO，Ａ１）
特開平９－１６３８９号公報（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F　9/06　630B