(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2014-63285
(P2014-63285A)
(43)公開日　平成26年4月10日(2014.4.10)

| (51)Int.Cl. | | F I | | | テーマコード（参考） |
|---|---|---|---|---|---|
| G06F 9/44 | (2006.01) | G06F | 9/44 | 530S | 5B060 |
| G06F 9/45 | (2006.01) | G06F | 9/44 | 322E | 5B081 |
| G06F 12/00 | (2006.01) | G06F | 9/44 | 530P | |
| | | G06F | 12/00 | 591 | |

審査請求 未請求　請求項の数 12　ＯＬ　（全 13 頁）

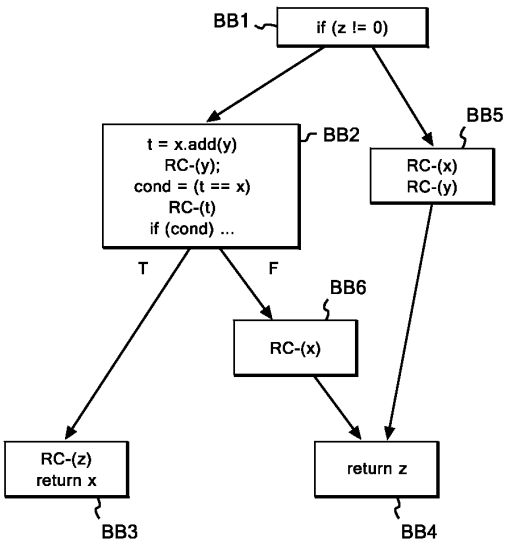
(21)出願番号　特願2012-207105 (P2012-207105)
(22)出願日　平成24年9月20日 (2012.9.20)

(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレーション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国１０５０４　ニューヨーク州　アーモンク　ニュー　オーチャード　ロード
(74)代理人　100108501
弁理士　上野　剛史
(74)代理人　100112690
弁理士　太佐　種一
(74)代理人　100091568
弁理士　市位　嘉宏

最終頁に続く

(54)【発明の名称】ガベージ・コレクションのためのコード変換方法、プログラム及びシステム

(57)【要約】
【課題】　ガベージ・コレクション技法において、リファレンス・カウントのためのコードを挿入するための改善された技法を提供する。
【解決手段】　基本ブロックが２つ以上の後続基本ブロックをもつ場合に、上記後続基本ブロックのうち、１つ以上の基本ブロックの先頭において、ある変数が生きて(live)おり、他の基本ブロックSでその変数が生きていない場合、元の基本ブロックと、変数が生きていない後続基本ブロックの間に新しい基本ブロックを挿入し、その新しい基本ブロック内に、その変数に関するRC-を生成する。
【選択図】　図７



【特許請求の範囲】
【請求項１】
コンピュータの処理により、プログラム・コードを読み込んで、第１の基本ブロックに続く２つ以上の後継基本ブロックをもつコードに、レファレンス・カウントを減らすコードRC-を挿入する方法であって、
前記後継基本ブロックのうちの第２の基本ブロックの先頭において生きている変数と同一の変数が、前記後継基本ブロックのうちの該第２の基本ブロックとは別の第３の基本ブロックにおいて生きていないことに応答して、前記第１の基本ブロックと、前記第３の基本ブロックの間に新たな第４の基本ブロックを挿入し、該基本ブロック内に該変数に関するRC-のコードを挿入するステップを有する、
コード変換方法。
【請求項２】
前記プログラム・コードがPythonバイトコードであり、前記RC-が、Python仮想マシン（ＰＶＭ）上のＪＩＴコンパイラにより生成されるネイティブ・コードに挿入される、請求項１に記載の方法。
【請求項３】
コンピュータの処理により、プログラム・コードを読み込んで、第１の基本ブロックに続く２つ以上の後継基本ブロックをもつコードに、レファレンス・カウントを減らすコードRC-を挿入する方法であって、
前記後継基本ブロックのうちの第２の基本ブロックの先頭において生きている変数と同一の変数が、前記後継基本ブロックのうちの該第２の基本ブロックとは別の第３の基本ブロックにおいて生きていない場合：
(a) 前記第３の基本ブロックの先行基本ブロックの数が１の場合、前記第３の基本ブロックの先頭にRC-のコードを挿入し、
(b) 前記第３の基本ブロックの先行基本ブロックの数が２以上の場合、前記先行基本ブロックのすべての基本ブロックの最後において前記変数が生きているなら、前記第３の基本ブロックの先頭にRC-のコードを挿入するステップを有する、
コード変換方法。
【請求項４】
前記プログラム・コードがPythonバイトコードであり、前記RC-が、Python仮想マシン（ＰＶＭ）上のＪＩＴコンパイラにより生成されるネイティブ・コードに挿入される、請求項３に記載の方法。
【請求項５】
コンピュータの処理により、プログラム・コードを読み込んで、第１の基本ブロックに続く２つ以上の後継基本ブロックをもつコードに、レファレンス・カウントを減らすコードRC-を挿入するプログラムであって、
前記コンピュータに、
前記後継基本ブロックのうちの第２の基本ブロックの先頭において生きている変数と同一の変数が、前記後継基本ブロックのうちの該第２の基本ブロックとは別の第３の基本ブロックにおいて生きていないことに応答して、前記第１の基本ブロックと、前記第３の基本ブロックの間に新たな第４の基本ブロックを挿入し、該基本ブロック内に該変数に関するRC-のコードを挿入するステップを実行させる、
コード変換プログラム。
【請求項６】
前記プログラム・コードがPythonバイトコードであり、前記RC-が、Python仮想マシン（ＰＶＭ）上のＪＩＴコンパイラにより生成されるネイティブ・コードに挿入される、請求項５に記載のプログラム。
【請求項７】
コンピュータの処理により、プログラム・コードを読み込んで、第１の基本ブロックに続く２つ以上の後継基本ブロックをもつコードに、レファレンス・カウントを減らすコー

ドRC-を挿入するプログラムであって、
前記コンピュータに、
前記後継基本ブロックのうちの第２の基本ブロックの先頭において生きている変数と同一の変数が、前記後継基本ブロックのうちの該第２の基本ブロックとは別の第３の基本ブロックにおいて生きていない場合：
(a) 前記第３の基本ブロックの先行基本ブロックの数が１の場合、前記第３の基本ブロックの先頭にRC-のコードを挿入し、
(b) 前記第３の基本ブロックの先行基本ブロックの数が２以上の場合、前記先行基本ブロックのすべての基本ブロックの最後において前記変数が生きているなら、前記第３の基本ブロックの先頭にRC-のコードを挿入するステップを実行させる、
コード変換プログラム。
【請求項８】
前記プログラム・コードがPythonバイトコードであり、前記RC-が、Python仮想マシン（ＰＶＭ）上のＪＩＴコンパイラにより生成されるネイティブ・コードに挿入される、請求項７に記載のプログラム。
【請求項９】
コンピュータの処理により、プログラム・コードを読み込んで、第１の基本ブロックに続く２つ以上の後継基本ブロックをもつコードに、レファレンス・カウントを減らすコードRC-を挿入するシステムであって、
前記後継基本ブロックのうちの第２の基本ブロックの先頭において生きている変数と同一の変数が、前記後継基本ブロックのうちの該第２の基本ブロックとは別の第３の基本ブロックにおいて生きていないことに応答して、前記第１の基本ブロックと、前記第３の基本ブロックの間に新たな第４の基本ブロックを挿入し、該基本ブロック内に該変数に関するRC-のコードを挿入する手段を有する、
システム。
【請求項１０】
前記プログラム・コードがPythonバイトコードであり、前記RC-が、Python仮想マシン（ＰＶＭ）上のＪＩＴコンパイラにより生成されるネイティブ・コードに挿入される、請求項９に記載のシステム。
【請求項１１】
コンピュータの処理により、プログラム・コードを読み込んで、第１の基本ブロックに続く２つ以上の後継基本ブロックをもつコードに、レファレンス・カウントを減らすコードRC-を挿入するシステムであって、
前記後継基本ブロックのうちの第２の基本ブロックの先頭において生きている変数と同一の変数が、前記後継基本ブロックのうちの該第２の基本ブロックとは別の第３の基本ブロックにおいて生きていない場合：
(a) 前記第３の基本ブロックの先行基本ブロックの数が１の場合、前記第３の基本ブロックの先頭にRC-のコードを挿入し、
(b) 前記第３の基本ブロックの先行基本ブロックの数が２以上の場合、前記先行基本ブロックのすべての基本ブロックの最後において前記変数が生きているなら、前記第３の基本ブロックの先頭にRC-のコードを挿入する手段を有する、
システム。
【請求項１２】
前記プログラム・コードがPythonバイトコードであり、前記RC-が、Python仮想マシン（ＰＶＭ）上のＪＩＴコンパイラにより生成されるネイティブ・コードに挿入される、請求項１１に記載のシステム。
【発明の詳細な説明】
【技術分野】
【０００１】
この発明は、コンピュータにおけるガベージ・コレクション技法に関し、より詳しくは

、リファレンス・カウント(reference counting)方式のガベージ・コレクション技法に関する。

【背景技術】

【０００２】

リファレンス・カウント方式のガベージ・コレクション（ＧＣ）は、停止時間の短さ、ガベージ・コレクションをもたない言語（例えば、C言語）との親和性の高さから、いくつかの言語の実装で実際に採用されている。

【０００３】

リファレンス・カウント方式のガベージ・コレクションを実現するためには、オブジェクト毎にもつ使用回数を示す整数のカウンタ（incする(RC+)、decする(RC-)）のコードを適切に挿入する必要がある。RC-では、カウンタが0になったら、オブジェクトを回収する。

【０００４】

リファレンス・カウント方式のガベージ・コレクションの典型的な従来技術として下記のような公開公報に記載された技法が知られている。

【０００５】

特開２００３－５０７４０号公報は、アプリケーションプログラムの動作によりメモリセルの被参照数が瞬間的に大きなピーク値をとる場合にメモリ使用量を抑えて有効なガベージコレクションを行う装置において、各メモリセル内にメモリセルの被参照数のピーク値が表せない程度に小さいビット列であるカウンタを設けておき、ガベージコレクション装置が、アプリケーションプログラムによるメモリセルへのポインタの変更時にカウンタ変更部によりそのビット列で表せる最大値以下の範囲内でカウンタを増減しカウンタが０になればセル解放部によりメモリセルを解放し、また、メモリ不足時等に再カウント部により、ルートポインタからポインタチェーンを辿ることで各メモリセルが参照されている数を調査し、その結果が０のメモリセルは解放し、０以外のメモリセルのカウンタにはカウンタ再設定部により調査結果の数を設定することを開示する。

【０００６】

上記従来技術では、コードの所定位置に最初から明示的にRC+/RC-が挿入されるが、より最近になって、http://www.hpl.hp.com/personal/Pramod_Joisha/Publications/vee08.pdf、あるいは米国特許第７６９３９１９号公報に記述されているように、中間言語レベルで、最初はRC+/RC-を挿入せずに最適化を行い、最後にRC+/RC-を挿入する技法が提案されている。このようにする方がコンパイラの最適化の適用が容易である。

【０００７】

しかし、http://www.hpl.hp.com/personal/Pramod_Joisha/Publications/vee08.pdfや米国特許第７６９３９１９号公報に記述されている技法に従いRC+/RC-を挿入した場合、1) オブジェクトのlast useの後にオブジェクトのアドレスをNULLにする操作が必要であり、2) 基本ブロック(BB)内で生存期間(life time)が閉じる変数についても後続BBでRC-が必要であり、3) RC-でオブジェクトがNULLかどうかの検査が必要という、実行パスを長くする操作が要求される、という問題があった。

【先行技術文献】

【特許文献】

【０００８】

【特許文献１】特開２００３－５０７４０号公報

【特許文献２】米国特許第７６９３９１９号公報

【非特許文献】

【０００９】

【非特許文献１】http://www.hpl.hp.com/personal/Pramod_Joisha/Publications/vee08.pdf

【発明の概要】

【発明が解決しようとする課題】

【００１０】
この発明の目的は、リファレンス・カウント方式のガベージ・コレクション技法において、従来技術よりも改善された態様でリファレンス・カウントのためのコードを挿入できるようにすることにある。
【課題を解決するための手段】
【００１１】
この発明は、上記の課題を解決するためになされたものであり、基本ブロック（basic block、以下BBとも記す)が２つ以上の後続基本ブロック(successor BB)をもつ場合に、
(a) 上記後続基本ブロックのうち、１つ以上の基本ブロックSの先頭において、ある変数が生きて(live)おり、他の基本ブロックSでその変数が生きていない場合、元の基本ブロックと、変数が生きていない後続基本ブロックの間に新しい基本ブロックを挿入し、その新しい基本ブロック内に、その変数に関するRC-を生成する。
【００１２】
式で書くと次のとおりである。
【数１】

$$\left(\bigcup_{s\subset Succ(B)\wedge s\neq S} live_{in}(s)\right)\cap\neg live_{in}(S)$$

この式が空でない変数について、基本ブロックBと基本ブロックSの間に、新しい基本ブロックを生成し、RC-を生成する。なおこの式で、Succ(B)は、基本ブロックBの後続基本ブロックの集合であり、$live_{in}(s)$は、基本ブロックsの入口生存である変数の集合である。さらに、SはSucc(B)に含まれる。
【００１３】
(b) この発明の別の側面として、上記後続基本ブロックのうち、１つ以上の基本ブロックの先頭において、ある変数が生きており、他の１つ以上の基本ブロックでその変数が生きていない場合、
変数が生きていない後続基本ブロックSの先行基本ブロックの数が1である場合、後続基本ブロックの先頭にRC-を生成する。
変数が生きていない後続基本ブロックSの先行基本ブロックの数が2以上である場合、全ての先行基本ブロックの最後において変数が生きていれば、後続基本ブロックの先頭にRC-を生成する。そうでない場合は、上記(a)の手法を適用する。
【発明の効果】
【００１４】
この発明のリファレンス・カウント方式のガベージ・コレクションは、従来技術に比較して、以下の操作が不要となることによって、よりコード生成量が少なく、実行パスを短くすることができる。
- オブジェクトのlast useの後にオブジェクトのアドレスをNULLにすること。
- 基本ブロック内で生存期間(life time)が閉じる変数について後続基本ブロックでRC-すること。
- RC-で、オブジェクトがNULLかどうかの検査。
【図面の簡単な説明】
【００１５】
【図１】本発明を実施するためのハードウェア構成のブロック図である。
【図２】本発明を実施するためのソフトウェアの機能構成のブロック図である。
【図３】本発明の第１の実施例に係る処理のフローチャートを示す図である。
【図４】本発明の第２の実施例に係る処理のフローチャートを示す図である。
【図５】従来技術におけるリファレンス・カウントの挿入処理の例を示す図である。
【図６】従来技術におけるリファレンス・カウントの挿入処理の例を示す図である。
【図７】本発明におけるリファレンス・カウントの挿入処理の例を示す図である。

【発明を実施するための形態】
【００１６】
以下、図面に従って、本発明の実施例を説明する。これらの実施例は、本発明の好適な態様を説明するためのものであり、発明の範囲をここで示すものに限定する意図はないことを理解されたい。また、以下の図を通して、特に断わらない限り、同一符号は、同一の対象を指すものとする。
【００１７】
図１を参照すると、本発明の一実施例に係るシステム構成及び処理を実現するためのコンピュータ・ハードウェアのブロック図が示されている。図１において、システム・バス１０２には、ＣＰＵ１０４と、主記憶（ＲＡＭ）１０６と、ハードディスク・ドライブ（ＨＤＤ）１０８と、キーボード１１０と、マウス１１２と、ディスプレイ１１４が接続されている。ＣＰＵ１０４は、好適には、３２ビットまたは６４ビットのアーキテクチャに基づくものであり、例えば、インテル社のPentium(商標)4、インテル社のCore(商標) 2 DUO、AMD社のAthlon(商標)などを使用することができる。主記憶１０６は、好適には、２ＧＢ以上の容量、より好ましくは、４ＧＢ以上の容量をもつものである。
【００１８】
ハードディスク・ドライブ１０８には、オペレーティング・システム（ＯＳ）２０２（図２）が、格納されている。オペレーティング・システム２０２は、Linux（商標）、マイクロソフト社のWindows(商標) 7、Windows XP(商標)、Windows(商標)2003サーバ、アップルコンピュータのMac OS X(商標)などの、ＣＰＵ１０４に適合する任意のものでよい。
【００１９】
ハードディスク・ドライブ１０８にはまた、Apacheなどの、Ｗｅｂサーバとしてシステムを動作させるためのプログラムが保存され、システムの起動時に、主記憶１０６にロードされる。
【００２０】
ハードディスク・ドライブ１０８にはさらに、Python仮想マシン（ＰＶＭ）２０４（図２）を実現するためのCPythonプログラムが保存され、システムの起動時に、主記憶１０６にロードされる。
【００２１】
ハードディスク・ドライブ１０８にはさらに、アプリケーション・プログラムのバイトコード２０６（図２）が保存されている。
【００２２】
ハードディスク・ドライブ１０８にはさらに、ＰＶＭ２０４上で動作し、バイトコード２０６を、ネイティブ・コード２１０に変換するＪＩＴコンパイラ２０８が保存されている。ＪＩＴコンパイラ２０８は、バイトコード２０６を構文解析し、コントロール・フロー・グラフを作成して変数の生存解析などを行い、中間処理言語レベルで最適化を行い、最後に、ガベージ・コレクションのためのRC+/RC-のコードを挿入する処理を行う。図２には特に、ＪＩＴコンパイラ２０８が含む処理ルーチンとして、生存解析ルーチン２０８ａと、リファレンス・カウント・コード挿入ルーチン２０８ｂとが代表的が表示されている。ＰＶＭ２０４には、リファレンス・カウント・コード挿入ルーチン２０８ｂによって挿入されたコードに従い、オブジェクトに関連するリファレンス・カウントがゼロになったとき、当該実装オブジェクトを回収するガベージ・コレクション機能が実装されている。ＪＩＴコンパイラ２０８のリファレンス・カウント・コード挿入ルーチン２０８ｂの動作の詳細は、図３及び図４のフローチャートを参照して後で説明する。
【００２３】
なお、図示しないが、ハードディスク・ドライブ１０８には、Pythonのソースコードと、Pythonのソースコードをバイトコードに変換する、バイトコードコンパイラなどのプログラムが格納されていてもよい。
【００２４】
キーボード１１０及びマウス１１２は、オペレーティング・システム（ＯＳ）２０２が

提供するグラフィック・ユーザ・インターフェースに従い、ディスプレイ１１４に表示されたアイコン、タスクバー、テキストボックスなどのグラフィック・オブジェクトを操作するために使用される。

【００２５】

ディスプレイ１１４は、これには限定されないが、好適には、１０２４×７６８以上の解像度をもち、３２ビットtrue colorのＬＣＤモニタである。ディスプレイ１１４は例えば、ＰＶＭ上で実行されるアプリケーション・プログラムによる動作の結果を表示するために使用される。

【００２６】

通信インターフェース１１６は、好適には、イーサネット(R)プロトコルにより、ネットワークと接続されている。通信インターフェース１１６は、クライアント・コンピュータ（図示しない）からApacheが提供する機能により、ＴＣＰ／ＩＰなどの通信プロトコルに従い、処理リクエストを受け取り、あるいは処理結果を、クライアント・コンピュータ（図示しない）に返す。

【００２７】

アプリケーション・プログラムのバイトコード２０６は、キーボード１１０またはマウス１１２の操作に応答してオペレーティング・システム２０２がＰＶＭ２０４を介してＪＩＴコンパイラ２０８を起動することにより、ＪＩＴコンパイラ２０８によって読み込まれ、ネイティブ・コード２１０に変換される。このようにして生成されたネイティブ・コード２１０は、オペレーティング・システム２０２によって実行されることになる。

【００２８】

次に、図３のフローチャートを参照して、ＪＩＴコンパイラ２０８のリファレンス・カウント・コードを挿入する処理を説明する。以下の処理は主としてＪＩＴコンパイラ２０８のリファレンス・カウント・コード挿入ルーチン２０８ｂによって実行されるが、リファレンス・カウント・コード挿入ルーチン２０８ｂはＪＩＴコンパイラ２０８の一部であるので、便宜上、ＪＩＴコンパイラ２０８が主体であるとして説明する。

【００２９】

図３において、ＪＩＴコンパイラ２０８は、ステップ３０２で、プログラムPを読み込む。ここでは、プログラムPは、Pythonのバイトコード２０６である。

【００３０】

ステップ３０４で、ＪＩＴコンパイラ２０８は、プログラムPのコントロール・フロー・グラフ（ＣＦＧ）　Ｃを作成する。

【００３１】

次に、ステップ３０６で、ＪＩＴコンパイラ２０８は、コントロール・フロー・グラフに基づき、生存解析ルーチン２０８ａを呼び出して、Ｃ上の変数の生存解析を行なう。なお、コントロール・フロー・グラフの作成とそれに基づく生存解析は、Andrew W. Appel, "Modern Compiler Implementation in ML", Cambridge University Press, 1998の第１０章などに記述されているので、ここでは詳細は説明は行なわない。

【００３２】

次に、ステップ３０８で、ＪＩＴコンパイラ２０８は、Ｃから基本ブロックBを１つ取り出すことを試みる。ここで、基本ブロックとは、コンパイラの分野で周知の概念で、制御のフローがその開始地点から始まり、その最終地点を除き停止や分岐の可能性なく最終地点で終るような連続的なステートメントの並びのことである。

【００３３】

そして、ステップ３１０で取り出すべき基本ブロックBがないと判断すると、ＪＩＴコンパイラ２０８は処理を終る。

【００３４】

ステップ３１０で取り出すべき基本ブロックBがあると、ＪＩＴコンパイラ２０８はステップ３１２で、Bの後続基本ブロックの集合BSを作成する。

【００３５】

ＪＩＴコンパイラ２０８はステップ３１４で、BSの要素が１つ以下かどうかを判断するし、もしそうなら、処理はステップ３０８に戻り、まだ取り出していない別の基本ブロックBを取り出そうと試みる。

【００３６】

ステップ３１４でＪＩＴコンパイラ２０８が、BSの要素が２つ以上だと判断すると、ＪＩＴコンパイラ２０８はステップ３１６で、プログラムP中のすべての変数を含む集合Vを作成する。そしてステップ３１８でBSから基本ブロックbを１つ取り出すことを試みる。

【００３７】

ステップ３２０で取り出すべき基本ブロックbがないと判断すると、処理はステップ３０８に戻り、ＪＩＴコンパイラ２０８は、Cから、まだ取り出していない別の基本ブロックBを取り出そうと試みる。

【００３８】

ステップ３２０で取り出すべき基本ブロックbがあったなら、ＪＩＴコンパイラ２０８は、ステップ３２２で、bの先頭で生きている変数集合Uを作成し、U = U ∩ Vとする。

【００３９】

ステップ３２４でＪＩＴコンパイラ２０８は、Uから変数uを１つ取り出すことを試みる。そしてステップ３２６でuがあるかどうか判断し、もしないなら、ステップ３１８に戻る。一方、uがあるなら、ＪＩＴコンパイラ２０８は、ステップ３２８で、BS - b、すなわちBSからbを除いた集合の先頭でuが生きていない基本ブロックの集合BD(u)を作成する。

【００４０】

ＪＩＴコンパイラ２０８は、ステップ３３０で、BD(u)が空かどうか判断し、もしそうなら、ステップ３２４に戻る。

【００４１】

BD(u)が空でないなら、ＪＩＴコンパイラ２０８は、ステップ３３２で、BD(u)が含むそれぞれの基本ブロックbbについて、Bとbbの間に新しい基本ブロックを生成し、その基本ブロックにRC-(u)を生成して、V = V - uにより、変数集合Vから変数uを取り去り、ステップ３２４に戻る。

【００４２】

以上の処理をまとめると、次のようになる。

基本ブロックが２つ以上の後続基本ブロックをもつ場合に、上記後続基本ブロックのうち、１つ以上の基本ブロックの先頭において、ある変数が生きて(live)おり、他の基本ブロックSでその変数が生きていない場合、元の基本ブロックと、変数が生きていない後続基本ブロックの間に新しい基本ブロックを挿入し、その新しい基本ブロック内に、その変数に関するRC-を生成する。

【００４３】

以上の処理を、式を用いて説明すると、以下のとおりである。

【数２】

$$\left( \bigcup_{s \subset Succ(B) \wedge s \neq S} live_{in}(s) \right) \cap \neg live_{in}(S)$$

プログラムにおいて、この式が空でない変数について、基本ブロックBと基本ブロックSの間に、新しい基本ブロックを生成し、RC-を生成する。なおこの式で、Succ(B)は、基本ブロックBの後続基本ブロックの集合であり、$\mathrm{live}_{\mathrm{in}}(s)$は、基本ブロックsの入口生存である変数の集合である。さらに、SはSucc(B)に含まれる。

【００４４】

次に、図４のフローチャートを参照して、図３に示す処理の別の実施例を説明する。図４のフローチャートの処理は、かなりの部分が図３のフローチャートの処理と同一である

。すなわち、図４のフローチャートのステップ４０２～４３０はそれぞれ、図３のフローチャートのステップ３０２～３３０と実質的に同一である。例えば、ステップ４０２がステップ３０２に対応し、ステップ４０４がステップ３０４に対応し、・・・、ステップ４３０がステップ３３０に対応する。

【００４５】

そこで、ステップ４０２～４３０については、上記ステップ３０２～３３０についての説明を援用することにして、図４のフローチャートの処理の説明を、ステップ４３０から開始すると、ＪＩＴコンパイラ２０８は、ステップ４３０で、BD(u)が空かどうか判断し、もしそうなら、ステップ４２４に戻る。

【００４６】

一方、ＪＩＴコンパイラ２０８は、ステップ４３０でBD(u)が空でないと判断すると、ステップ４３２でBD(u)から基本ブロックb1を取り出すことを試みる。

【００４７】

ステップ４３４でＪＩＴコンパイラ２０８は、b1があるかどうか判断し、もしないなら、ステップ４２４に戻る。

【００４８】

ステップ４３４でb1があると判断すると、ＪＩＴコンパイラ２０８はステップ４３６で、b1の先行基本ブロックの数が１かどうか判断し、もしそうなら、ステップ４３８で、b1の先頭にRC-(u)を生成し、V = V - uにより、Vからuを取り去る。そしてステップ４２４に戻る。

【００４９】

b1の先行基本ブロックの数が１でないなら、ＪＩＴコンパイラ２０８はステップ４４０で、b1の全ての基本ブロックの最後でuが生きているかどうか判断し、もしそうならステップ４３８で、b1の先頭にRC-(u)を生成し、V = V - uにより、Vからuを取り去り、ステップ４２４に戻る。

【００５０】

テップ４４０で、b1の全ての基本ブロックの最後でuが生きているわけではないと判断すると、ＪＩＴコンパイラ２０８はステップ４４２で、BD(u)が含むそれぞれの基本ブロックbbについて、Bとbbの間に新しい基本ブロックを生成し、その基本ブロックにRC-(u)を生成して、V = V - uにより、変数集合Vから変数uを取り去り、ステップ４２４に戻る。なお、ステップ４４２は、図３のステップ３３２と処理内容が同一である。

【００５１】

次に、本発明を下記のようなコードに適用した例を説明する。

```
foo(x, y, z) {
  if (z != 0) {
   t == x.add(y)
   if ((t == x) { return x; }
  }
  return z;
}
```

【００５２】

比較のため先ず、米国特許第７６９３９１９号公報あるいはhttp://www.hpl.hp.com/personal/Pramod_Joisha/Publications/vee08.pdfに示す技法を適用した場合について示す。

【００５３】

すると、図５に示すような、BB1、BB2、BB3、BB4及びBB5の５つの基本ブロックに分割される。

【００５４】

http://www.hpl.hp.com/personal/Pramod_Joisha/Publications/vee08.pdfには下記の式が示されている。

【数３】

$$D' = \left( live_{in}(B)) \cup \left( \bigcup_{s \in B} defs_{must}(s) \right) \right) - live_{in}(B')$$

この式で、$def_{must}()$は、必須の定義を示す。

【００５５】

この式を用いて、図５の場合の計算をしてみる。

B’= BB3について

B= BB2: $live_{in}$(BB2) = {x,y,z}, $defs_{must}$(s in BB2) = {t}

$live_{in}$(BB3) = {x}

この値を用いて

D’ for BB3

= ($live_{in}$(BB2) ∪ $defs_{must}$(s in BB2)) - $live_{in}$(BB3)

= ({x, y, z} ∪ {t}) - {x} = {y, z, t}

B’= BB4について

B= BB1: $live_{in}$(BB1) = {x,y,z}, $defs_{must}$(s in BB1) = {}

B= BB2: $live_{in}$(BB2) = {x,y,z}, $defs_{must}$(s in BB2) = {t}

$live_{in}$(BB4) = {z}

この値を用いて

D’ for BB4

= (($live_{in}$(BB1) ∪ $defs_{must}$(s in BB1)) ∪ ($live_{in}$(BB2) ∪ $defs_{must}$(s in BB2))) - $live_{in}$(BB4)

= (({x, y, z}) ∪ ({x, y, z} ∪ {t})) - {z}

= {x, y, z, t} - {z}

= {x, y, t}

【００５６】

例えば変数yは、BB2において、x.add(y)でlastuseであり、dieacross(x.add(y)) = yとなりx.add(y)の後ろにRC-(y)が挿入される。ここで、RC-()とは、RC-()の中ではreference countがdecrementされた後にy=NULLが代入されるものである(DAN semantics)。さらに、D’に従って、BB3とBB4の前にRC_-(y)を挿入する。RC_-(y)とは、yがNULLでないときのみRC(y)を呼ぶものである。

【００５７】

ここで、もし米国特許第７６９３９１９号公報あるいはhttp://www.hpl.hp.com/personal/Pramod_Joisha/Publications/vee08.pdfには記述されていない、定数伝搬の最適化を導入するならば、 RC_-(var)について、BB3に到達するyが常にNULLであることがわかるので、RC_-(y)を削除することは可能である。

CPythonやYARVのように実装上オブジェクトのアドレスがNULLにならない、という知識を利用して、BB3のRC_-(z)はRC-(z)に、BB4のRC_-(x)はRC-(x)に、置き換え可能である。このようにして最適化した後のコードを図６に示す。すなわち、BB4では、yがNULLであるかどうかのチェックが常に行われBB2から到達した場合必要ないにもかかわらず、実行命令数を増やす。

【００５８】

本発明を適用した場合は、図７のようになる。ここで、

$live_{in}$(BB2) = {x,y,z}, $live_{in}$(BB4) = {z}である。

BB B = BB1, BB S = BB4、の場合を考えると、

$live_{in}$(BB2) ∩ ¬ $live_{in}$(BB4) = {x,y}

となり、BB1とBB4の間に新しいBB5を挿入し、RC-(x)とRC-(y)を生成する。すると、BB1->BB4のパスで、使われないオブジェクトxとyを回収可能である。

同様にBB2->BB4のパスにBB6を挿入し、RC-(x)を生成する。

【００５９】

なお、以上の説明では、RC+のコードを挿入する処理については言及されていないが、RC+は従来技術に従い挿入してよい。

【００６０】

図７のコードは、図５あるいは図６のコードと比較して、以下の操作が不要となる。

- オブジェクトのlast useの後にオブジェクトのアドレスをNULLにすること。
- 基本ブロック内で生存期間(life time)が閉じる変数について後続基本ブロックでRC-すること。
- RC-で、オブジェクトがNULLかどうかの検査。

【００６１】

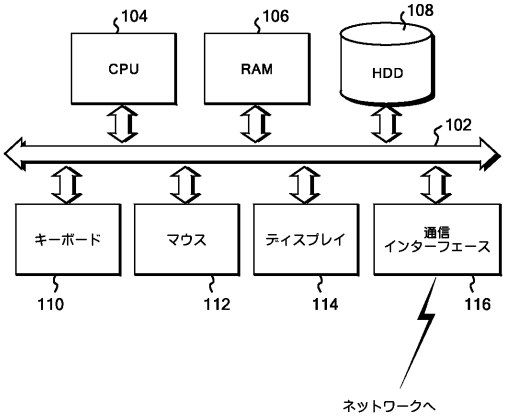
以上のように本発明を、Python上のＪＩＴコンパイラに実装した例として説明してきたが、本発明はこれには限定されず、PHPなど、レファレンス・カウント方式のガベージ・コレクションを実装する任意の処理系に適当であることを理解されたい。

【符号の説明】

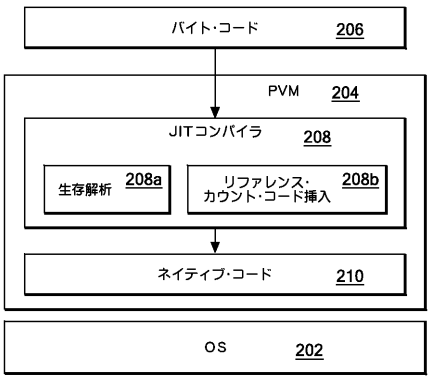
【００６２】

１０４・・・ＣＰＵ
１０６・・・ＲＡＭ
１０８・・・ＨＤＤ
２０２・・・ＯＳ
２０４・・・ＰＶＭ
２０６・・・バイトコード
２０８・・・ＪＩＴコンパイラ
２０８ａ・・・生存解析ルーチン
２０８ｂ・・・リファリンス・カウント・コード挿入ルーチン

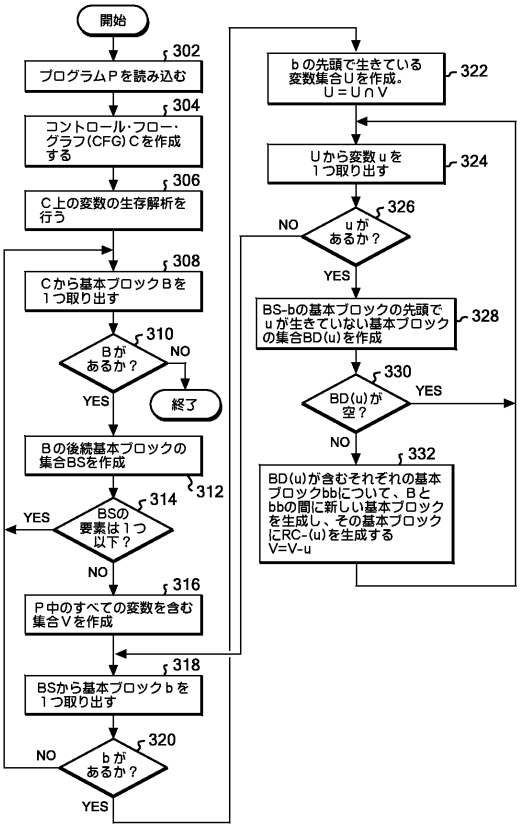
【図１】



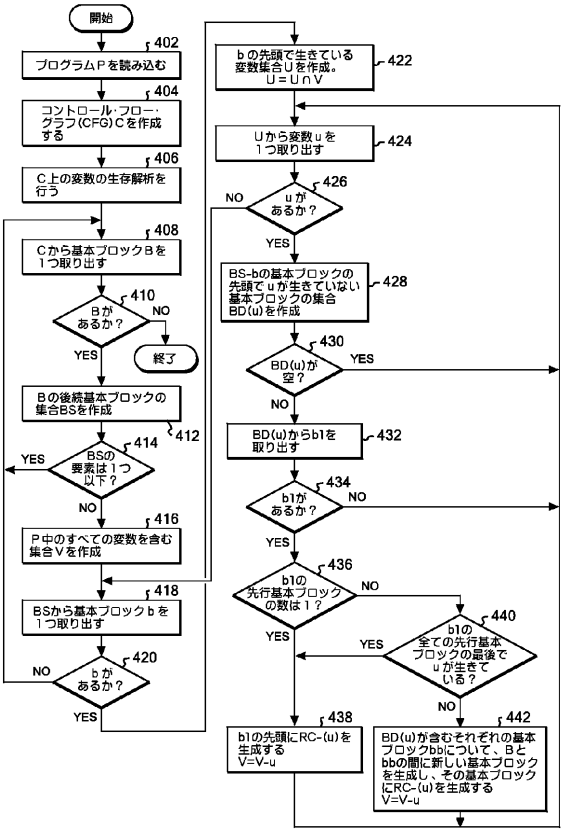

【図２】



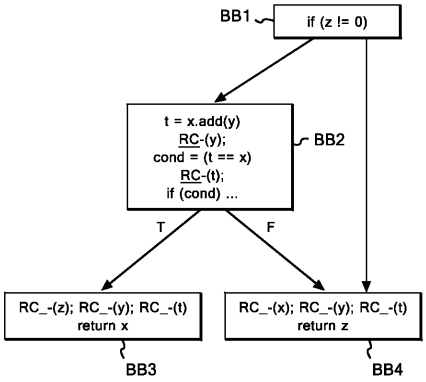

【図３】



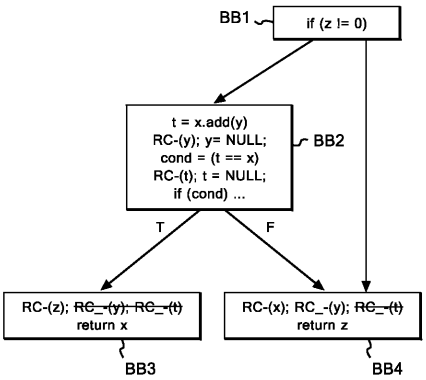
【図４】


開始
402
プログラムPを読み込む
404
コントロール・フロー・グラフ(CFG)Cを作成する
406
C上の変数の生存解析を行う
408
Cから基本ブロックBを1つ取り出す
410
Bがあるか？
NO
YES
終了
Bの後続基本ブロックの集合BSを作成
412
414
BSの要素は1つ以下？
YES
NO
416
P中のすべての変数を含む集合Vを作成
418
BSから基本ブロックbを1つ取り出す
420
bがあるか？
NO
YES
bの先頭で生きている変数集合Uを作成。U=U∩V
422
Uから変数uを1つ取り出す
424
426
uがあるか？
NO
YES
BS-bの基本ブロックの先頭でuが生きていない基本ブロックの集合BD(u)を作成
428
430
BD(u)が空？
YES
NO
BD(u)からb1を取り出す
432
434
b1があるか？
NO
YES
436
b1の先行基本ブロックの数は1？
NO
YES
440
b1の全ての先行基本ブロックの最後でuが生きている？
YES
NO
438
b1の先頭にRC-(u)を生成するV=V-u
442
BD(u)が含むそれぞれの基本ブロックbbについて、Bとbbの間に新しい基本ブロックを生成し、その基本ブロックにRC-(u)を生成するV=V-u

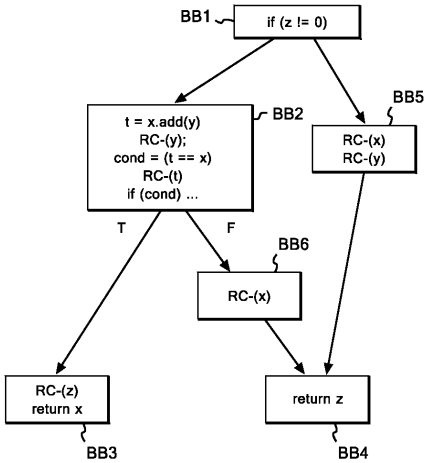
【図５】


BB1
if (z != 0)
t = x.add(y)
RC-(y);
cond = (t == x)
RC-(t);
if (cond) ...
BB2
T
F
RC_-(z); RC_-(y); RC_-(t)
return x
RC_-(x); RC_-(y); RC_-(t)
return z
BB3
BB4

【図６】


BB1
if (z != 0)
t = x.add(y)
RC-(y); y= NULL;
cond = (t == x)
RC-(t); t = NULL;
if (cond) ...
BB2
T
F
RC-(z); ~~RC_-(y); RC_-(t)~~
return x
RC-(x); RC_-(y); ~~RC_-(t)~~
return z
BB3
BB4

【図７】


BB1
if (z != 0)
t = x.add(y)
RC-(y);
cond = (t == x)
RC-(t)
if (cond) ...
BB2
BB5
RC-(x)
RC-(y)
T
F
BB6
RC-(x)
RC-(z)
return x
return z
BB3
BB4

フロントページの続き

(72)発明者　石崎　一明
東京都江東区豊洲五丁目６番５２号　ＮＢＦ豊洲キャナルフロント　日本アイ・ビー・エム株式会社　東京基礎研究所内

Ｆターム(参考) 5B060 AA10
5B081 AA09 CC30 CC41