(12) 特許協力条約に基づいて公開された国際出願

(19) 世界知的所有権機関
国際事務局

(43) 国際公開日
2010 年 11 月 4 日(04.11.2010)

PCT

(10) 国際公開番号
WO 2010/125960 A1

(51) 国際特許分類:
G06F 9/52 (2006.01)

(21) 国際出願番号: PCT/JP2010/057122

(22) 国際出願日: 2010 年 4 月 22 日(22.04.2010)

(25) 国際出願の言語: 日本語

(26) 国際公開の言語: 日本語

(30) 優先権データ:
特願 2009-109937 2009 年 4 月 28 日(28.04.2009) JP

(71) 出願人 (米国を除く全ての指定国について): 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) [US/US]; 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 New York (US).

(72) 発明者; および

(75) 発明者/出願人 (米国についてのみ): 石崎 一明 (ISHIZAKI Kazuaki) [JP/JP]; 〒2428502 神奈川県大和市下鶴間１６２３番地１４日本アイ・ビー・エム株式会社東京基礎研究所 Kanagawa (JP).

(74) 代理人: 上野 剛史, 外(UENO Takeshi et al.); 〒2428502 神奈川県大和市下鶴間１６２３番地１４日本アイ・ビー・エム株式会社大和事業所内 Kanagawa (JP).

(81) 指定国 (表示のない限り、全ての種類の国内保護が可能): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) 指定国 (表示のない限り、全ての種類の広域保護が可能): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), ユーラシア (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), ヨーロッパ (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

添付公開書類:

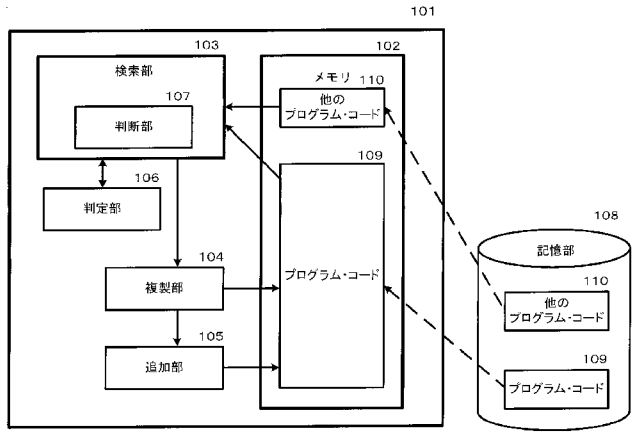
— 国際調査報告（条約第 21 条(3)）

WO 2010/125960 A1

(54) **Title**: METHOD FOR CONVERTING PROGRAM CODE OF PROGRAM RUNNING IN MULTITHREADED ENVIRONMENT TO PROGRAM CODE HAVING FEWER LOCK CONFLICTS, AND COMPUTER PROGRAM AND COMPUTER SYSTEM THEREFOR

(54) 発明の名称 : マルチスレッド上で動作するプログラムのプログラム・コードをロック衝突が少ないプログラム・コードに変換するための方法、並びにそのコンピュータ・プログラム及びコンピュータ・システム

[図1]



102 MEMORY
103 RETRIEVAL UNIT
104 DUPLICATION UNIT
105 ADDITION UNIT
106 DETERMINATION UNIT
107 ASSESSMENT UNIT
108 STORAGE UNIT
109 PROGRAM CODE
110 OTHER PROGRAM CODE

(57) **Abstract**: A method for automatically specifying a portion where performance degradation due to lock conflict tends to occur, and automatically modifying the portion, is required. Provided is a method for converting a program code of a program running in a multithreaded environment to a program code having fewer lock conflicts. The method comprises a step of reading the program code into a memory, and retrieving a first conditional statement for branching to a path which is within a synchronization block and has no adverse effect to the synchronization block, from the read program code; a step of duplicating the path having no adverse effect which has been branched by the retrieved first conditional statement, on the outside of the synchronization block; and a step of adding a second conditional statement into the program code in response to the duplication, the second conditional statement being a conditional statement for branching to the duplicated path having no adverse effect.

(57) 要約:

[続葉有]

ロック衝突による性能低下の問題が発生しそうな箇所を自動的に特定し、該箇所を自動的に修正する方法が必要とされている。 本発明は、マルチスレッド上で動作するプログラムのプログラム・コードをロック衝突が少ないプログラム・コードに変換する方法を提供する。該方法は、上記プログラム・コードをメモリ内に読み出し、該読み出したプログラム・コードから、同期ブロック内にあり且つ該同期ブロックに対する副作用がないパスへ分岐させる第１の条件文を検索するステップと、上記同期ブロック外に、上記検索された第１の条件文によって分岐される副作用のないパスを複製するステップと、上記複製に応じて、上記プログラム・コード内に第２の条件文を追加するステップであって、上記第２の条件文は上記複製された副作用のないパスへ分岐させる条件文である、上記追加するステップとを含む。

明　細　書

発明の名称 ：

マルチスレッド上で動作するプログラムのプログラム・コードをロック衝突が少ないプログラム・コードに変換するための方法、並びにそのコンピュータ・プログラム及びコンピュータ・システム

技術分野

[0001] 本発明は、マルチスレッド上で動作するプログラムのプログラム・コードをロック衝突が少ないプログラム・コードに変換するための方法、並びにそのコンピュータ・プログラム及びコンピュータ・システムに関する。

背景技術

[0002] 近年、プロセッサ・アーキテクチャにおいて、１つの中央演算処理装置（以下、ＣＰＵ：Central Processing Unit）の内部に複数のＣＰＵコアを封入したいわゆるマルチコアへのシフトが進んでいる。マルチコアは、外部的には１つのＣＰＵでありながら、内部的には２つのＣＰＵとして認識させられる。マルチコアにより、主に並列処理を行わせる環境下において、ＣＰＵチップ全体での処理能力を上げ、システムの性能向上を果たすことが可能である。マルチコアは、プロセッサが同時に実行可能なハードウェアスレッドの数を飛躍的に増加させる。例えば、ＳＵＮ（商標）のNiagara２（商標）は、１ＣＰＵ上で６４個のスレッドを実行することが可能である。

[0003] 複数のスレッドを同時に実行することが可能なプロセッサ上で従来のプログラムが実行される場合、ロック衝突によるプロセッサの性能低下が問題になる。ここで、ロック衝突とは、あるスレッドにおいて、排他実行が必要なクリティカル・セクションに入るためにロックが獲得された場合、他のスレッドにおいて、上記ロックが解放されるまで次のステップが実行されない状態をいう。次のステップが実行されないスレッドは、典型的には、スピン・ループ（spin loop）又はスリープ（sleep）状態になる。

[0004] ロック衝突が発生する原因はさまざまである。例えば、キューからデータ

を取り出す処理が、複数のスレッドで頻繁に実行されるプログラムの場合を考える。また、上記プログラムのプログラム・コードにおいて、排他実行が必要な処理を記載するための同期ブロック中に、上記取り出す処理についての文が記載されている場合を考える。上記プログラムは、実行時において、上記取り出す処理が呼び出される毎に常にロックを獲得する。よって、ロック衝突が頻繁に発生する。従って、頻繁に発生する上記ロック衝突によって、システムの性能上の問題が引き起こされうる。

[0005] ３２個のハードウェアスレッドを実行しうるコンピュータ・システムにおいて、上記プログラム・コードと似た構造のプログラム・コードのプログラムが、

システムの性能上のボトルネックになることが、SUN（商標） BUG DataBaseに報告されている（http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6525425）。該報告では、上記似た構造のプログラム・コードを有するSUN JDK 1.5.0_13 のjava.lang.ref.RereferenceQueue クラスで上記問題が確認されている。

上記クラスのメソッドである上記取り出す処理に対応するpoll()メソッドは、SUN JDK1.5.0_14において修正されている。該修正では、poll()メソッドの実装において、ロックを獲得する処理についての文の前に、キューが空かどうかを判断し空と判断した場合にメソッドを抜ける処理についての文が挿入されている。

[0006] 下記非特許文献１は、プログラム・コードにmarked bitを付加し、該付加されたmarked bitとcompareAndSwap（ＣＡＳ）命令とを利用することで、プログラム・コードが同時並行性（concurrency）の高いプログラム・コードに変形可能であることを記載する。下記非特許文献１では、上記marked bitの付加は手動で行われている。

先行技術文献

非特許文献

[0007] 非特許文献1：Martin Vechev, Eran Yahav, Deriving Linearizable Fine-Gr

ained Concurrent Objects, PLDI'08, ACM 978-1-59593-860-2/08/06, 2008年6月7日～13日

発明の概要

発明が解決しようとする課題

[0008] ロック衝突による性能低下の問題が発生するたびに、その原因を特定し、人手でプログラム・コードを修正する作業が必要である。しかし、該作業は、現存するプログラム・コードの量から考えると、多量のワークロードを必要とするために不可能に近い。従って、上記問題が発生しそうな箇所を自動的に特定し、該箇所を自動的に修正する方法が必要とされている。

課題を解決するための手段

[0009] 本発明は、マルチスレッド上で動作するプログラムのプログラム・コードをロック衝突が少ないプログラム・コードに変換する方法を提供する。該方法は、コンピュータ・システムに下記ステップを実行させる。該ステップは、

上記プログラム・コードをメモリ内に読み出し、該読み出したプログラム・コードから、同期ブロック内にあり且つ該同期ブロックに対する副作用がないパスへ分岐させる第１の条件文を検索するステップであって、上記副作用がないとは上記同期ブロック内の文による処理中の値が上記同期ブロック外の文によって参照できないことである、上記検索するステップと、

上記同期ブロック外に、上記検索された第１の条件文によって分岐される副作用のないパスを複製するステップと、

上記複製に応じて、上記プログラム・コード内に第２の条件文を追加するステップであって、上記第２の条件文は上記複製された副作用のないパスへ分岐させる条件文である、上記追加するステップと

を含む。上記方法によって、副作用のないパスへ分岐させた場合、同期ブロック内の第１の条件文が実行されない。よって、ロック衝突が少なくなる。上記方法は、特に、条件文が１つで、分岐が２つの場合に有効である。

[0010] 本発明の１つの実施態様では、上記第１の条件文が、第１の条件式を含む

。該第１の条件式は、第１の変数の読み出し、定数又はそれらの組み合わせに基づいてパスを選択させるように定義されている。ただし、前記第１の変数は、前記第１の条件文を含むサブルーチンを呼び出すルーチンの中だけが有効範囲である変数ではない。

[0011] 本発明の１つの実施態様では、上記第２の条件文が、上記第１の条件式の結果を格納するための変数に基づいて、上記第１の条件式の結果に応じてパスを選択させる第２の条件式を含む。

[0012] 本発明の１つの実施態様では、上記検索するステップが、上記読み出したプログラム・コードから、更新文をさらに検索するステップを含み、上記更新文は、上記同期ブロック内にあり且つ上記第１の変数を更新させ、

上記追加するステップが、上記検索された更新文の直後から上記同期ブロックの終了文までの間に、上記第１の条件式の結果を格納するための変数と、該変数に代入される上記第１の条件式の結果とを含む文をさらに追加するステップを含む。

[0013] 本発明の１つの実施態様では、上記検索するステップが、上記読み出したプログラム・コード及び他のプログラム・コードの少なくとも１つから、更新文をさらに検索するステップを含み、上記更新文は、上記同期ブロックと同期する他の同期ブロック内にあり且つ上記第１の変数を更新させ、

上記追加するステップが、上記検索された更新文の直後から上記他の同期ブロックの終了文までの間に、上記第１の条件式の結果を格納するための変数と、該同じ変数に代入される上記第１の条件式の結果とを含む文をさらに追加するステップを含む。

[0014] 本発明の１つの実施態様では、上記追加するステップが、上記第１の条件式の結果を格納するための変数が初めて参照される文より前に、上記第２の条件文が上記同期ブロックを含むパスを実行させる値を用いて又は上記第１の条件式の結果を用いて上記第１の条件式の結果を格納するための変数を初期化する文をさらに追加するステップを含む。

[0015] 本発明の１つの実施態様では、上記副作用が、実行結果の取り消しが容易

でない処理である。

[0016] 本発明の１つの実施態様では、上記副作用が、スレッド間の同期を実行する処理、インスタンス変数に値を代入する処理、クラス変数に値を代入する処理、入出力を実行する処理、ネイティブコードを呼び出す処理又は配列要素の位置を示す番号に値を代入する処理である。

[0017] 本発明の１つの実施態様では、上記第１の条件式の数が２以上である場合、該２以上の第１の条件式の結果が、上記２以上の第１の条件式の組み合わせで求められる。

[0018] 本発明の１つの実施態様では、上記第１の条件式によって選択されるパスの数が３通り以上である場合、上記第１の条件式の結果を格納するための変数が、上記３通り以上のパスのうち副作用のないパスそれぞれに対応する。

[0019] 本発明の１つの実施態様では、上記第１の条件式の結果を格納するための変数が、原子性が保証されたデータ型の変数である。

[0020] 本発明の１つの実施態様では、上記検索するステップが、上記プログラム・コード及び他のプログラム・コードの少なくとも１つから、上記同期ブロックと同期する他の同期ブロックをさらに検索するステップを含み、

上記検索の終了に応じて、上記複製するステップを実行する。

[0021] 本発明の１つの実施態様では、上記検索するステップが、上記プログラム・コード及び他のプログラム・コードの少なくとも１つに含まれており且つ上記同期ブロック外の文から上記第１の変数を参照する文をさらに検索するステップを含み、

上記検索の終了に応じて、上記複製するステップを実行する。

[0022] 本発明の１つの実施態様では、上記同期ブロック内にスレッドを起動させる文が含まれる場合、

上記検索するステップが、上記プログラム・コード及び他のプログラム・コードの少なくとも１つに含まれており且つ上記スレッドの実装を記載する文から、上記同期ブロック内の文と同期をとって実行される文をさらに検索するステップを含み、

上記方法は、コンピュータ・システムに、上記同期をとって実行される文が検索されることに応じて、上記スレッドを起動させる文を含むパスを副作用のあるパスと判定するステップをさらに実行させる。

[0023] 本発明の１つの実施態様では、上記同期ブロック内にスレッドを起動させる文が含まれる場合、

上記検索するステップが、上記スレッドを起動させる文を含むパスが副作用のないパスかどうかをユーザに判断することを求めるステップをさらに含む。

[0024] 本発明の１つの実施態様では、上記同期ブロックが実行されるスレッドの実行順序を決定するために使用される変数が上記同期ブロック内に定義されている場合、

上記検索するステップが、上記プログラム・コード及び他のプログラム・コードの少なくとも１つに含まれており且つ上記使用される変数を参照する文を上記同期ブロック外からさらに検索するステップを含み、

上記方法は、コンピュータ・システムに、上記使用される変数を参照する文が検索されることに応じて、上記使用される変数を含むパスを副作用のあるパスと判定するステップをさらに実行させる。

[0025] 本発明の１つの実施態様では、上記同期ブロックが実行されるスレッドの実行順序を決定するために使用される変数が上記同期ブロック内に定義されている場合、

上記検索するステップが、上記使用される変数を含むパスが副作用のないパスかどうかをユーザに判断することを求めるステップをさらに含む。

[0026] 本発明の１つの実施態様では、上記追加するステップが、上記読み出したプログラム・コードに、上記第１の条件式の結果を格納するための変数を宣言する文をさらに追加するステップを含み、

共有変数が上記プログラム・コード及び上記他のプログラム・コードの少なくとも１つに含まれており且つ上記同期ブロック外の文からアクセスされる場合に、上記宣言がメモリのアクセス順序に関する最適化を抑止すること

の指定及び原子性を保証することの指定を含み、上記共有変数は上記副作用のないパスに含まれており、且つ複数のスレッドからアクセスされることができ、

上記アクセスされない場合に、上記宣言が原子性を保証することの指定を含む。

[0027] 本発明の１つの実施態様では、上記検索するステップが、上記同期ブロックと同期する他の同期ブロックがあるかどうかをユーザに判断させることを許すステップをさらに含む。そして、上記方法は、コンピュータ・システムに、上記許すステップの後に、上記複製するステップを実行させる。

[0028] 本発明の１つの実施態様では、上記検索するステップが、上記第１の変数を参照する文があるかどうかをユーザに判断させることを許すステップをさらに含む。そして、上記方法は、コンピュータ・システムに、上記許すステップの後に、上記複製するステップを実行させる。

[0029] 本発明の１つの実施態様では、上記第２の条件式に含まれる変数がメモリのアクセス順序に関する最適化を抑止することを指定された変数の場合、上記第１の条件式の結果を代入する文をさらに追加するステップにおいて、上記同期ブロックの終了文の直前に、上記第１の条件式の結果を代入する文を追加する。

[0030] 本発明はまた、マルチスレッド上で動作するプログラムのプログラム・コードをロック衝突が少ないプログラム・コードに変換する方法を提供する。該方法は、コンピュータ・システムに下記ステップを実行させる。該ステップは、

上記プログラム・コードをメモリ内に読み出し、該読み出したプログラム・コードから、同期ブロック内にあり且つ該同期ブロックに対する副作用がないパスへ分岐させる第１の条件文を検索するステップであって、上記副作用がないとは上記同期ブロック内の文による処理中の値が上記同期ブロック外の文によって参照できないことである、上記検索するステップと、

上記同期ブロック外に、上記検索された第１の条件文によって分岐される

副作用のないパスを複製するステップと、

上記複製に応じて、上記プログラム・コード内に第２の条件文を追加するステップであって、上記第２の条件文は上記複製された副作用のないパスへ分岐させる条件文である、上記追加するステップと

を含み、

前記第１の条件文が第１の条件式を含み、該第１の条件式は、インスタンス変数若しくはクラス変数の読み出し、定数又はそれらの組み合わせに基づいて、副作用のないパスに分岐させる場合にその結果がtrueとなるように定義されており、一方副作用のあるパスに分岐させる場合にその結果がfalseとなるように定義されており、

上記第２の条件文が、第２の条件式を含み、該第２の条件式は、上記第１の条件式の結果を格納するためのboolean型の変数に基づいて、上記第１の条件式の結果に応じてパスを選択させるように定義されている。

[0031] 本発明はさらに、コンピュータに、上記方法のいずれか１つに記載の方法の各ステップを実行させるコンピュータ・プログラムを提供する。

[0032] 本発明はまた、マルチスレッド上で動作するプログラムのプログラム・コードをロック衝突が少ないプログラム・コードに変換するコンピュータ・システムを提供する。該コンピュータ・システムは、

上記プログラム・コードをメモリ内に読み出し、該読み出したプログラム・コードから、同期ブロック内にあり且つ該同期ブロックに対する副作用がないパスへ分岐させる第１の条件文を検索する検索部であって、上記副作用がないとは上記同期ブロック内の文による処理中の値が上記同期ブロック外の文によって参照できないことである、上記検索部と、

上記同期ブロック外に、上記検索された第１の条件文によって分岐される副作用のないパスを複製する複製部と、

上記複製に応じて、上記プログラム・コード内に第２の条件文を追加する追加部であって、上記第２の条件文は上記複製された副作用のないパスへ分岐させる条件文である、上記追加部と

を含む。

[0033] 本発明の１つの実施態様では、上記検索部が、上記読み出したプログラム・コードから、更新文をさらに検索し、上記更新文は上記同期ブロック内にあり、且つ上記第１の変数を更新させ、

上記追加部が、上記検索された更新文の直後から上記同期ブロックの終了文までの間に、上記第１の条件式の結果を格納するための変数と、該変数に代入される上記第１の条件式の結果とを含む文をさらに追加する。

[0034] 本発明の１つの実施態様では、上記検索部が、上記読み出したプログラム・コード及び他のプログラム・コードの少なくとも１つから、更新文をさらに検索し、上記更新文は上記同期ブロックと同期する他の同期ブロック内にあり、且つ上記第１の変数を更新させ、 上記追加部が、上記検索された更新文の直後から上記他の同期ブロックの終了文までの間に、上記第１の条件式の結果を格納するための変数と、該同じ変数に代入される上記第１の条件式の結果とを含む文をさらに追加する。

[0035] 本発明の１つの実施態様では、上記追加部が、上記第１の条件式の結果を格納するための変数が初めて参照される文より前に、上記第２の条件文が上記同期ブロックを含むパスを実行させる値を用いて又は上記第１の条件式の結果を用いて上記第１の条件式の結果を格納するための変数を初期化する文をさらに追加する。

[0036] 本発明の１つの実施態様では、上記検索部が、上記プログラム・コード及び他のプログラム・コードの少なくとも１つから、上記同期ブロックと同期する他の同期ブロックをさらに検索する。

[0037] 本発明の１つの実施態様では、上記検索部が、上記プログラム・コード及び他のプログラム・コードの少なくとも１つに含まれており且つ上記同期ブロック外の文から上記第１の変数を参照する文をさらに検索する。

[0038] 本発明の１つの実施態様では、上記同期ブロック内にスレッドを起動させる文が含まれる場合、

上記検索部が、上記プログラム・コード及び他のプログラム・コードの少

なくとも１つに含まれており且つ上記スレッドの実装を記載する文から、上記同期ブロック内の文と同期をとって実行される文をさらに検索する。

上記コンピュータ・システムは、上記同期をとって実行される文が検索されることに応じて、上記スレッドを起動させる文を含むパスを副作用のあるパスと判定する判定部をさらに含む。

[0039] 本発明の１つの実施態様では、上記同期ブロック内にスレッドを起動させる文が含まれる場合、

上記検索部が、上記スレッドを起動させる文を含むパスが副作用のないパスかどうかをユーザに判断させることを許す判断部をさらに含む。

[0040] 本発明の１つの実施態様では、上記同期ブロックが実行されるスレッドの実行順序を決定するために使用される変数が上記同期ブロック内に定義されている場合、

上記検索部が、上記プログラム・コード及び上記他のプログラム・コードの少なくとも１つに含まれており且つ上記使用される変数を参照する文を上記同期ブロック外からさらに検索する。

上記コンピュータ・システムは、上記使用される変数を参照する文が検索されることに応じて、上記使用される変数を含むパスを副作用のあるパスと判定する判定部をさらに含む。

[0041] 本発明の１つの実施態様では、上記同期ブロックが実行されるスレッドの実行順序を決定するために使用される変数が上記同期ブロック内に定義されている場合、

上記検索部が、上記使用される変数を含むパスが副作用のないパスかどうかをユーザに判断させることを許す判断部をさらに含む。

[0042] 本発明の１つの実施態様では、上記追加部が、上記読み出したプログラム・コードに、上記第１の条件式の結果を格納するための変数を宣言する文をさらに追加し、

共有変数が上記プログラム・コード及び上記他のプログラム・コードの少なくとも１つに含まれており且つ上記同期ブロック外の文からアクセスされ

る場合に、上記宣言がメモリのアクセス順序に関する最適化を抑止することの指定及び原子性を保証することの指定を含み、上記共有変数は上記副作用のないパスに含まれており、且つ複数のスレッドからアクセスされることができ、

上記アクセスされない場合に、上記宣言が原子性を保証することの指定を含む。

[0043] 本発明の１つの実施態様では、上記検索部が、上記同期ブロックと同期する他の同期ブロックがあるかどうかをユーザに判断させることを許す判断部をさらに含む。

[0044] 本発明の１つの実施態様では、上記検索部が、上記第１の変数を参照する文があるかどうかをユーザに判断させることを許す判断部をさらに含む。

発明の効果

[0045] 本発明の実施態様によれば、同期ブロック中の第１のパスの中から、ロックが獲得されていなくても実行が可能な第２のパスが自動的に見つけ出される。そして、該見つけ出された第２のパス及びロックを獲得せずに第２のパスを実行させるための条件文が、プログラム・コード中の上記第１のパスの外に追加される。上記追加によって、プログラム・コード実行時のロック衝突時間が減少し、マルチスレッド上のＣＰＵ効率の低下を防止することが可能である。

図面の簡単な説明

[0046] [図1]本発明の実施態様における、プログラム・コードを変換するためのシステムが有する機能を示す機能ブロック図を示す。

[図2]本発明の実施態様における、図１に示す変換される前のプログラム・コードの例を示す。

[図3]本発明の実施態様における、図１に示す変換プログラムにおいて実行される処理ステップのうち、変換に必要な情報を取得する処理ステップの例を示す。

[図4]本発明の実施態様における、図１に示す変換プログラムにおいて実行さ

れる処理ステップのうち、変換を行う処理ステップの例を示す。

[図5]本発明の実施態様における、図４に記載のステップ３１７の処理の詳細を示す。

[図6]本発明の実施態様における、図３～図５に記載の各ステップの動作を具体的に説明するために使用する入力データＣＬを示す。

[図7]本発明の実施態様における、図４に記載のステップ３１６の処理が実行された後のメソッドdequeue()を示す。

[図8]本発明の実施態様における、図６について、図３～図５に記載のステップの処理が実行された後の入力データ（変換後のプログラム・コード）を示す。

[図9]本発明の実施態様における、図３～図５に記載の各ステップの動作を具体的に説明するために使用する入力データＣＬを示す。

[図10]本発明の実施態様における、図４に記載のステップ３１６の処理が実行された後のメソッドdequeue()を示す。

[図11]本発明の実施態様における、図９について、図３～図５に記載の各ステップの処理が実行された後の入力データ（変換後のプログラム・コード）を示す。

[図12]本発明の実施態様における、第１の条件式によって選択されるパスが３通り以上である場合の、図３～図５に記載の各ステップの処理が実行される前のプログラム・コードの一部及び実行された後のプログラム・コードの一部を示す。

[図13]本発明の実施態様における、図１のシステムが有するコンピュータ・ハードウェアのブロック図を示す。

発明を実施するための形態

[0047] 本発明の実施態様において、「マルチスレッド上で動作するプログラムのプログラム・コード」とは、スレッドと呼ばれる処理単位を複数生成し、該生成されたスレッド毎にＣＰＵタイムを割り当てることによって、並行して複数の処理を行なうことができるプログラムのソースコードをいう。該ソー

スコードは、例えばＪａｖａ（商標）言語又はＣ＋＋言語を使用して記述される。

[0048] 本発明の実施態様において、「ロック」とは、１つのスレッドで処理が実行されるときに、他のスレッドで上記処理自身及び上記処理と関係する処理が実行されないようにすることである。例えば、第１のスレッドにおいて上記ロックの対象の処理が開始された場合、第２のスレッドにおいて上記処理又は上記処理に関係する処理は実行されない。上記第２のスレッドにおける上記処理又は上記処理に関係する処理は、上記第１のスレッドにおける処理が終了するまで待機させられ、上記終了後に実行されうる。

[0049] 本発明の実施態様において、「ロック衝突」とは、１つのスレッドで実行される処理がロックの対象であるために、他のスレッドで実行される上記処理自身及び上記処理と関係する処理が、上記第１のスレッドで実行される処理が終了するまで実行待ちになる状態をいう。また、「ロック衝突が少ない」とは、ロック衝突の発生する頻度が少ないこと及びロック衝突による上記実行待ちの時間が短いことを表す。本発明の実施態様を用いて変換されたプログラム・コードは、変換される前のプログラム・コードに比べ、ロック衝突が少なくなる。

[0050] 本発明の実施態様において、「同期ブロック」とは、上記ロックの対象である処理が記載されているプログラム・コード上の文である。該プログラム・コード上の文は、同期ブロックの開始文、同期ブロックの終了文及び上記開始文と終了文に囲まれた文で構成される。

同期ブロックの開始文は、上記プログラム・コードが例えばＪａｖａ（商標）言語で記載されている場合、例えば、synchronized宣言を記載する文又はsynchronized修飾子で修飾されたメソッド名を記載する文である。

同様に、同期ブロックの終了文は、上記プログラム・コードが例えばＪａｖａ（商標）言語で記載されている場合、上記開始文の終了位置を表す「}」である。

サブルーチンの呼び出しが上記開始文と終了文に囲まれた文（すなわち同

期ブロック内）内に含まれている場合、該サブルーチンの実装を記載する文は上記同期ブロック内に含まれていてもよい。上記サブルーチンの実装を記載する文の中に別のサブルーチンの呼び出しが含まれるというように、サブルーチンの呼び出しが階層状になっている場合、全てのサブルーチンの実装を記載する文が、上記同期ブロックにさらに含まれていてもよい。ここで、サブルーチンとは例えば、メソッドである。

[0051] 本発明の実施態様において、「同期ブロックと同期する他の同期ブロック」とは、例えば、ロックの対象である処理によってアクセスされる変数が共通である又は指定されたロックオブジェクトが共通である同期ブロックである。ここで、ロックオブジェクトとは、Ｊａｖａ（商標）言語のsynchronized宣言において指定する変数である。例えば、第１の同期ブロックを宣言するsynchronized宣言においてロックオブジェクトlockが指定され且つ第２の同期ブロックを宣言するsynchronized宣言においてロックオブジェクトlockが指定された場合、第２の同期ブロックは、第１の同期ブロックと同期する他の同期ブロックである。また、第１の同期ブロックがメソッドであり且つ第２の同期ブロックを宣言するsynchronized宣言において上記メソッドを指すthisが指定された場合、第２の同期ブロックは、第１の同期ブロックと同期する他の同期ブロックである。

[0052] 本発明の実施態様において、「同期ブロックに対する副作用」とは、同期ブロック内の文による処理（処理Ａ）の結果が同期ブロック外の文による処理（処理Ｂ）によって参照されうる処理Ａである。同期ブロックに対する副作用はまた、処理結果の取り消しが容易でない処理である。同期ブロックに対する副作用は、例えば、同期ブロック内に記載された別の同期ブロック内の処理、インスタンス変数に値を代入する処理、クラス変数に値を代入する処理、入出力を実行する処理、ネイティブコードに含まれる処理を呼び出す処理、又は配列要素の位置を示す番号に値を代入する処理であるが、これらに限定されない。 上記副作用は、例えばユーザが上記処理についての文をリストに登録し、コンピュータ・システムが該リストに登録された文を参照

することにより判断される。

なお、上記入出力とは、例えば、画面、ファイル又はネットワークへの入出力であるが、これらに限定されない。また、上記ネイティブコードは、マシン語で記載されたプログラムである。ネイティブコードは、例えば、コンパイル済みのプログラムである。ネイティブコードに含まれる処理は、例えば、Ｊａｖａ　Ｎａｔｉｖｅ　Ｉｎｔｅｒｆａｃｅ（ＪＮＩ）によって呼び出される。

[0053] 本発明の実施態様において、「同期ブロックに対する副作用がない」とは、同期ブロック内の文による処理中の値が上記同期ブロック外の文によって参照できないことである。副作用がないとは、副作用を起こす処理、例えば同期ブロック内に記載された別の同期ブロック内の処理、インスタンス変数に値を代入する処理、クラス変数に値を代入する処理、入出力を実行する処理、ネイティブコードに含まれる処理を呼び出す処理、又は配列要素の位置を示す番号に値を代入する処理のいずれもないことである。

[0054] 本発明の実施態様において、「パス」とは、プログラムの実行時において実行される処理に対応する、プログラム・コード上の一連の文である。該文は０であってもよい。

[0055] 本発明の実施態様において、「同期ブロックに対する副作用のないパス」とは、プログラムの実行時において上記同期ブロックに対する副作用のない処理に対応する、上記一連の文である。

[0056] 本発明の実施態様において、「条件文」とは、プログラムにおいて処理の分岐を実現させるためのプログラム・コード上に記載された文である。条件文は、条件式を含む。条件文は、選択されるパスの範囲を示す記号を含んでいてもよい。プログラム・コードが、例えばＪａｖａ（商標）で記載されている場合、上記記号は例えば「{」及び「}」である。条件文は、条件式の結果に応じて、実行されるパスを選択する。プログラム・コードが、例えばＪａｖａ言語で記載されている場合、条件文は例えばif文又はswitch文である。

[0057] 本発明の実施態様において、「条件式」とは、パスを選択するために使用される式である。条件式は、変数、定数及び演算子の少なくとも１を含む。条件文が、例えば、if文の文「if(x==0){y=1;}」である場合、条件式は、「(x==0)」である。また、条件文が、例えば、switch文又はselect case文の場合、両文に関連するcaseの値と該値が格納される変数とを比較する式を条件式とする。条件文が、例えばswitch文の文「switch(x){case 1:y=1;break;}」である場合、条件式は、「(x==1)」である。

[0058] 条件式の数は、１つの条件文に対して１つとは限らない。

（１）条件文が、例えば、if文の文

「if(x==0){y=0;}else if(x==1){y=1;}else{y=2;}」

である場合、「(x==0)」、「(x==1)」及び「その他」の少なくとも１が条件式でありうる。また、複数の条件式、例えば「(x==0)」、「(x==1)」及び「その他」から選択される２つ以上を組み合わせて、新たな条件式としてもよい。ここで、組み合わせとは、上記条件式同士を例えば論理演算することである。

（２）条件文が、例えば、switch文の文

「switch(x){

case 1: y=1; break;

case 2: y=2; break;

default: y=3; break;

}」

である場合、「(x==1)」、「(x==2)」及び「その他」の少なくとも１が条件式でありうる。また、複数の条件式、例えば「(x==1)」、「(x==2)」及び「その他」から選択される２つ以上を組み合わせて、新たな条件式としてもよい。

[0059] 本発明の実施態様において、「第１の変数」とは、第１の条件式にパスを選択させるための１つの変数である。「第１の変数」とは、例えば高々１つの変数である。プログラム・コードが、例えばＪａｖａ言語で記載されてい

る場合、第１の変数は、1つのインスタンス変数又は1つのクラス変数である。

[0060] 本発明の実施態様において、「第１の条件式の結果」とは、第１の条件式によって又は複数の第１の条件式の組み合わせによって表わされる値である。

（１）条件文が例えばif文の場合、第１の条件式の結果は例えば真又は偽で表される。上記真又は偽は、プログラム・コードを記載する言語の上記条件式の仕様に合わせた値で表現されてもよいし、又はユーザが決定した値で表現されてもよい。例えば、真を表す値をtrueとし偽を表す値をfalseとしてもよいし、又は真を表す値を0とし偽を表す値を-1としてもよい。上記真又は偽を表す値がtrue又はfalseの場合、第１の条件式の結果を格納するための変数は、例えばboolean型である。上記真又は偽を表す値が0又は-1の場合、第１の条件式の結果を格納するための変数は、例えば負の値をサポートする数値型である。また、if文が例えばelseif句を含むために上記if文に第１の条件式が複数ある場合、第１の条件式の結果は、複数の第１の条件式によって表わされる値を組み合わせて求められてよい。上記組み合わせるとは、条件式を表す値同士を例えば論理演算することである。

（２）条件文が例えばswitch文の場合、第１の条件式の結果は例えば、switch文において指定された変数がcase句で指定された値であることを示す値（以下、示す値）又は該示す値に対応する値で表される。上記示す値は、例えば、上記case句で指定された値であってもよい。上記対応する値は、例えばユーザが決定した値である。ユーザは、上記対応する値に、例えば連番を用いる。上記示す値を第１の条件式の結果とする場合、第１の条件式の結果を格納するための変数は、例えば上記指定された変数と同じデータ型である。また、第１の条件式の結果は、複数の上記示す値又は複数の上記対応する値を組み合わせて求められてもよい。上記組み合わせるとは、例えば、上記示す値又は上記対応する値のとる範囲に応じて、グループ分けすることである。

[0061] 本発明の実施態様において、「原子性が保証されたデータ型の変数」とは

、メモリへの変数の読み込みが１回の操作で実行されるデータ型の変数であり、該実行がプログラム・コードを記載する言語の仕様によって保証されているデータ型の変数である。

上記言語が、例えばＪａｖａ（商標）言語であり且つ上記プログラム・コードが３２ビット以上のＯＳ上で実行される場合、例えばboolean型、byte型、short型、int型又はfloat型が、原子性が保証されたデータ型である。

[0062] 本発明の実施態様において、「他のプログラム・コード」とは、変換される前のプログラム・コードからリンクされるプログラム・コードである。ここで、リンクされるとは、例えば呼び出されること又は変数を共有することであるが、これらに限定されない。また、他のプログラム・コードは、本発明の実施態様によって解析可能なプログラム・コードである。他のプログラム・コードとは、例えば、ソースコード又はバイトコードである。

[0063] 本発明の実施態様において、「メモリのアクセス順序に関する最適化を抑止すること」とは、変換される前のプログラム・コードがコンパイルされるときに、コンパイラによるメモリのアクセス順序に関する最適化をさせないことである。最適化を抑止することの指定は、例えば、変換される前のプログラム・コードがＪａｖａ(商標）で記載された言語の場合、修飾子volatileで変数を修飾することである。

[0064] 以下、図面に従って、本発明の実施態様を説明する。本実施態様は、本発明の好適な態様を説明するためのものであり、本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、特に断らない限り、同一の符号は、同一の対象を指す。

[0065] 図１は、本発明の実施態様における、プログラム・コードを変換するためのシステムが有する機能を示す機能ブロック図を示す。

コンピュータ・システム（１０１）は、メモリ（１０２）、検索部（１０３）、複製部（１０４）、追加部（１０５）、判定部（１０６）及び判断部（１０７）を含む。また、記憶部（１０８）は、コンピュータ・システム（１０１）に含まれていても、又は別のシステムに含まれていてもよい。

コンピュータ・システム（１０１）は、記憶部（１０８）から、変換される前のプログラム・コード（１０９）をメモリ内（１０２）に読み出す。ここで、コンピュータ・システムは（１０１）は、記憶部（１０８）から、他のプログラム・コード（１１０）をさらにメモリ内に読み出してもよい。

[0066] 検索部（１０３）は、上記メモリに読み出されたプログラム・コード（１０９）及び任意的に他のプログラム・コード（１１０）を検索する。該検索により、検索部（１０３）は、同期ブロック内にある第１の条件文、更新文、同期ブロックと同期する他の同期ブロック、同期ブロック外の文から第１の変数を参照する文、スレッドを起動させる同期ブロック内の文と同期をとって実行される他のスレッドの実装を記載する文、又は、スレッドの実行順序を決定するために使用される変数を参照する文を検索する。

検索部（１０３）は、判断部（１０７）を含む。

判断部（１０７）は、上記検索をするための情報が不十分の場合、ユーザに該情報を入力させることを許す。判断部（１０７）はまた、上記検索の条件をさらに絞り込むために、ユーザに該条件を入力させることを許す。

[0067] 判定部（１０６）は、検索部（１０３）から検索結果を受け取る。判定部（１０６）は、上記検索結果から、プログラム・コード（１０９）中の副作用のないパスを判定する。

[0068] 複製部（１０４）は、上記判定された副作用のないパスを、プログラム・コード（１０９）の同期ブロック外に複製する。

[0069] 追加部（１０５）は、上記複製された副作用のないパスへ分岐させる第２の条件文を、上記メモリに読み出されたプログラム・コード（１０９）に追加する。追加部（１０５）はまた、第１の条件式の結果を格納するための変数と該変数に代入される上記第１の条件式の結果とを含む文、該変数を初期化する文、又は該変数を宣言する文を、上記メモリに読み出されたプログラム・コード（１０９）に追加する。

[0070] 図２は、本発明の実施態様における、図１に示す変換される前のプログラム・コードの例を示す。

プログラム・コード（２００）は、変換される前のプログラム・コードである。以下では、該プログラム・コード（２００）を例として、明細書に記載の各用語を説明する。

[0071] 「クラス」に対応する文は、フォーマット「class クラス名 {クラスの実装}」を満たす文である。よって、変換される前のプログラム・コード（２００）において「クラス」に対応する文は、クラスＷ（２０１Ａ）及びクラスＸ（２０１Ｂ）である。

[0072] 「メソッド」に対応する文は、フォーマット「戻り値　メソッド名　(引数型　引数名){メソッドの実装}」を満たす文である。よって、変換される前のプログラム・コード（２００）において「メソッド」に対応する文は、メソッドenqueue(W w)（２０２Ａ）及びメソッドdequeue()（２０２Ｂ）である。また、メソッド内に記載された文に別のメソッド呼び出しが含まれる場合、該別のメソッドの実装を記載する文も、上記同期メソッド内に記載された文として扱う。

[0073] 「同期ブロック」に対応する文は、例えば、synchronizedブロックの宣言がされた文又はsynchronized修飾子が指定されたメソッドである。よって、変換される前のプログラム・コード（２００）において「同期ブロック」に対応する文は、第１の同期ブロック（２０３Ａ）、第２の同期ブロック（２０３Ｂ）及び第３の同期ブロック（２０３Ｃ）である。

[0074] 「同期ブロックのロックオブジェクト」は、例えば、上記synchronizedブロックの宣言で又は上記synchronized修飾子が指定されたメソッドの引数で指定される変数である。よって、第１の同期ブロック（２０３Ａ）のロックオブジェクトは、w（２０４Ａ）である。第２の同期ブロック（２０３Ｂ）のロックオブジェクトは、lock（２０４Ｂ）である。第３の同期ブロック（２０３Ｃ）のロックオブジェクトは、lock（２０４Ｃ）である。

また、同期ブロック内に記載された文にメソッド呼び出しが含まれる場合、該メソッドの実装を記載する文も、上記同期ブロック内に記載された文として扱う。

[0075] 「条件文」に対応する文は、例えば「ＩＦ」文、「ＳＷＩＴＣＨ」文、又は「３項演算子」が記載された文である。よって、変換される前のプログラム・コード（２００）において「条件文」に対応する文は、第１の条件文（２０５Ａ）、第２の条件文（２０５Ｂ）、第３の条件文（２０５Ｃ）及び第４の条件文（２０５Ｄ）である。

[0076] 「条件式」は、条件分岐文に含まれ実行パスを選択するために使用される式である。よって、第１の条件文（２０５Ａ）の条件式は、w.val==null（２０６Ａ）である。第２の条件文（２０５Ｂ）の条件式は、head==null（２０６Ｂ）である。第３の条件文（２０５Ｃ）の条件式は、head!=null（２０６Ｃ）である。第４の条件文（２０５Ｄ）の条件式は、w.next==w（２０６Ｄ）である。

[0077] 「パス」に対応する文は、プログラム・コード（２００）が実行された場合に、処理される可能性のある文である。例えば、メソッドenqueue(W w)（２０２Ａ）が実行され終了するまでのパスは、パス（２０７Ａ）であり、及びメソッドdequeue()（２０２Ｂ）が実行され終了するまでのパスは、パス（２０７Ｂ）である。

また、条件文が実行され終了するまでのパスは、それぞれの分岐の処理が開始する文から、それぞれの分岐の処理が終了する文までである。

（１）第１の条件文（２０５Ａ）は、メソッドenqueue(W w)（２０２Ａ）から抜ける処理（２０８Ａ及び２０８Ｂ）の次の文においてそれぞれの分岐の処理が終了する。よって、第１の条件文（２０５Ａ）が実行され終了するまでのパスは、条件式（２０６Ａ）が成立した場合に実行されるパス（２０９Ａ）及び条件式（２０６Ａ）が成立しなかった場合に実行されるパス（２０９Ｂ）である。

（２）第２の条件文（２０５Ｂ）は、「if(head==null){w.next=w;}else{w.next=head;}」と同じ意味である。よって、w又はheadをw.nextに代入する処理の次の文においてそれぞれの分岐の処理が終了する。従って、第２の条件文（２０５Ｂ）が実行され終了するまでのパスは、条件式（２０６Ｂ）が成

立した場合に実行されるパスである「w.next=W;」、及び条件式（２０６Ｂ）が成立しなかった場合に実行されるパスである「w.next=head;」である。

（３）第３の条件文（２０５Ｃ）は、メソッドdequeue()（２０２Ｂ）から抜ける処理（２０８Ｃ及び２０８Ｄ）の次の文でそれぞれの分岐の処理が終了する。よって、第３の条件文（２０５Ｃ）が実行され終了するまでのパスは、条件式（２０６Ｃ）が成立した場合に実行されるパス（２０９Ｃ）及び条件式（２０６Ｃ）が成立しなかった場合に実行されるパス（２０９Ｄ）である。

（４）第４の条件文（２０５Ｄ）は、「if(w.next==w){head=null;}else{head=w.next;}」と同じ意味である。よって、headにnull又はw.nextを代入する処理の次の文でそれぞれの分岐の処理が終了する。従って、第４の条件文（２０５Ｄ）が実行され終了するまでのパスは、条件式（２０６Ｄ）が成立した場合に実行されるパスである「head=null;」及び条件式（２０６Ｄ）が成立しなかった場合に実行されるパスである「head=w.next;」である。

[0078] 「第１の変数」は、分岐の判断に使用される変数である。よって、第１の条件文（２０５Ａ）における第１の変数は、w.val（２１０Ａ）である。第２の条件文（２０５Ｂ）における第１の変数は、head（２１０Ｂ）である。第３の条件文（２０５Ｃ）における第１の変数は、head（２１０Ｃ）である。第４の条件文（２０５Ｄ）における第１の変数は、w.next（２１０Ｄ）である。

[0079] 「更新文」は、上記第１の変数を含む同期ブロック内の文であって、上記第１の変数を更新させる文である。w.val（２１０Ａ）を更新させる更新文は、w.val（２１０Ａ）を含む第１の同期ブロック（２０３Ａ）には含まれていない。head（２１０Ｂ）を更新させる更新文は、更新文（２１１Ｂ）である。head（２１０Ｃ）を更新させる更新文は、更新文（２１１Ｃ）である。w.next（２１０Ｄ）を更新させる更新文は、更新文（２１１Ｄ）である。

[0080] 図３～図５は、本発明の実施態様における、図１に示す変換プログラムにおいて実行される処理ステップの例を示す。

以下では、変換される前のプログラム・コードがＪａｖａ（商標）言語で記載されたプログラム・コードの場合を例として説明する。

[0081] コンピュータ・システム（１０１）は、変換される前のプログラム・コード（以下、ＣＬ）を入力データとする。コンピュータ・システムは、該ＣＬに加えて、上記ＣＬからリンクされる別のプログラム・コード（以下、別のＣＬ）をさらに入力データとしてもよい。ここで、入力データとは、上記変換プログラムから参照可能なデータである。

[0082] 図３は、本発明の実施態様における、図１に示す変換プログラムにおいて実行される処理ステップのうち、変換に必要な情報を取得する処理ステップの例を示す。

[0083] ステップ３０１は、上記変換する処理の開始である。ステップ３０１では、コンピュータ・システムは、上記入力データを、例えばメモリに読み込む。該読み込みが終了することに応じて、該処理はステップ３０２に進む。

[0084] ステップ３０２では、コンピュータ・システムは、上記変換のための情報を格納するための変数ｉ、集合変数Ｍ、集合変数Ｓ及び集合変数Ｃを用意する。

変数ｉは、下記ステップ３１５で宣言される変数（以下、第２の変数）を識別するための値が格納される変数である。変数ｉは、例えば、上記第２の変数に振られる連番に使用される。なお、変数ｉは用意されなくてもよい。変数ｉが用意されない場合、コンピュータ・システムは、第２の変数が宣言される毎に、第２の変数に一意に識別できるような変数名を付ける。上記一意に識別できるような変数名は、例えばユーザによって決定されてもよいし、重複しないようにランダムで決定されてもよい。

集合変数は、異なる意味を持つデータをまとめて扱うためのグループである。集合変数は、例えば、配列又は構造体で表される。

集合変数Ｍは、上記ＣＬが実行されたときに処理が到達しうるメソッドについての情報が、上記メソッド毎に格納される集合変数である。上記メソッドについての情報は、対応するメソッドの実装を記載する文を、プログラム

・コードから特定できる情報であればよい。上記メソッドについての情報は、例えばメソッドの実装を記載する文の開始位置、メソッドの実装を記載する文の終了位置、又はメソッドの実装を記載する文であるがこれらに限られない。

集合変数Ｓは、上記メソッドそれぞれが実行されたときに処理が到達しうる同期ブロックについての情報が、上記同期ブロック毎に設定される集合変数である。上記同期ブロックについての情報とは、同期ブロックのロックオブジェクト（以下、Ｓの第１の情報）、同期ブロックのプログラム・コード上の開始終了位置（以下、Ｓの第２の情報）及び同期ブロック内の文（以下、Ｓの第３の情報）である。

集合変数Ｃは、上記同期ブロックそれぞれに含まれる所定の条件を満たす実行パスについての情報が、上記実行パス毎に設定される集合変数である。上記実行パスについての情報とは、上記実行パスに対応する上記同期ブロックのロックオブジェクト（以下、Ｃの第１の情報）、上記実行パスに対応する上記同期ブロックのプログラム・コード上の開始終了位置（以下、Ｃの第２の情報）及び上記実行パスをプログラム・コードから特定できる情報（以下、Ｃの第３の情報）である。上記Ｃの第３の情報は、例えば実行パスの開始位置、実行パスの終了位置、又は実行パスを記載する文であるがこれらに限られない。

コンピュータ・システムは、変数ｉ、集合変数Ｓ及び集合変数Ｃの上記用意において、該変数ｉ、該集合変数Ｓ及び該集合変数Ｃに初期値を設定する。上記初期値は、例えば、何も入っていないことを表す値である。コンピュータ・システムは、変数ｉの初期値として、例えば０又は１を設定する。また、集合変数Ｓ及びＣが、例えば、数値型変数と文字型変数とを含む構造体の場合、コンピュータ・システムは、集合変数Ｓ及びＣの初期値として、例えば、数値型変数の初期値には０を、文字型変数の初期値にはＮＵＬＬを設定する。

コンピュータ・システムはまた、集合変数Ｍの上記用意において、該集合

変数Mに上記メソッドについての情報を設定する。コンピュータ・システムは、ＣＬに含まれるメソッドを検索し、検索されたメソッドについての情報を集合変数Mに設定する。上記検索では、対象とするメソッドが予めユーザに指定されてもよいし、ＣＬに含まれる全てのメソッドが検索されてもよい。ＣＬに含まれるメソッドが実行されることによって到達しうるメソッドが、例えば上記別のＣＬから検索されてもよい。また、上記検索の代わりに、ユーザが、集合変数Mに上記メソッドについての情報を設定してもよい。

上記用意が終了することに応じて、該処理はステップ３０３に進む。

[0085] ステップ３０３は、集合変数Mについての繰り返しの開始を表す。

ステップ３０３では、コンピュータ・システムは、集合変数Mからメソッドについての情報（以下、ｍ）を１件取り出す。該取り出しが終了することに応じて、該処理はステップ３０４に進む。

[0086] ステップ３０４では、コンピュータ・システムは、集合変数Ｓにデータを設定する。

コンピュータ・システムは、上記取り出されたmに対応するメソッドが実行されたときに処理が到達しうる同期ブロックを検索する。なお、上記メソッド自身が同期ブロックの場合、上記メソッド自身も検索される同期ブロックとなる。ＣＬが、例えばＪａｖａ（商標）で記載されたプログラム・コードの場合、コンピュータ・システムは、例えば「synchronized」を上記ＣＬから検索する。なお、mに対応するメソッドが「synchronized」で修飾されている場合、上記修飾する「synchronized」もまた、検索される「synchronized」である。

コンピュータ・システムは、上記検索された同期ブロックそれぞれについての、ロックオブジェクト（Ｓの第１の情報）、プログラム・コード上の開始終了位置（Ｓの第２の情報）及び同期ブロック内の文（Ｓの第３の情報）を該検索された同期ブロックから取得し、該取得したＳの第１～第３の情報を、同期ブロックごとに、集合変数Ｓに１組の情報として設定する。なお、検索された同期ブロックのうち、既に検索され情報が設定されている同期ブ

ロックについては、上記設定は不要である。上記設定が終了することに応じて、該処理はステップ３０５に進む。

[0087] ステップ３０５は、集合変数Ｍについての繰り返しの終了を表す。

全てのｍが取り出し済みの場合、該処理はステップ３０６に進む。取り出していないｍがある場合、該処理はステップ３０３に戻る。

[0088] ステップ３０６は、集合変数Ｓについての繰り返しの開始を示す。

ステップ３０６では、コンピュータ・システムは、集合変数Ｓから同期ブロックについての上記１組の情報（以下、ｓ）を１件取り出す。該取り出しが終了することに応じて、該処理はステップ３０７に進む。

[0089] ステップ３０７では、コンピュータ・システムは、上記取り出されたｓに対応する集合変数Ｓに設定されている同期ブロック内の文（以下、ｓの第３の情報）に、副作用が発生しないパスが含まれるかどうかを判定する。該判定は、同期ブロック内の文（ｓの第３の情報）に含まれるパスであって、実行される経路それぞれについてのパス（以下、実行パス）全てに実行される。該判定では、コンピュータ・システムは、例えば、下記副作用リストに記載された副作用を起こす原因が、上記実行パスに含まれているかどうかを判定する。

副作用リストは、例えば、ユーザが予め作成しておく。ユーザは、副作用を起こす原因、例えばメソッド又は変数を、副作用リスト予めに登録しておく。メソッドは、例えばＩ／Ｏに関するメソッド又はＪＮＩのメソッドである。変数は、例えばインスタンス変数又はクラス変数である。コンピュータ・システムは、上記判定において、上記実行パスに、例えば上記メソッド又は変数への書き込みが見つかった場合、上記実行パスを副作用が発生するパスと判定し、見つからなかった場合、上記実行パスを副作用の発生しないパスと判定する。なお、上記登録は、コンピュータ・システムによって、自動的に実行されてもよい。コンピュータ・システムは、例えばステップ３０２において、上記ＣＬ又は上記別のＣＬの少なくとも１つから、副作用を起こす原因を取得し、該取得した副作用を起こす原因を副作用リストに登録しう

る。

また、コンピュータ・システムは、実行パスに含まれる変数の種類を調べることによって、上記判定を実行しうる。コンピュータ・システムは、例えば、上記変数の宣言が上記ＣＬのどこに記載されているか又は上記変数のスコープがＣＬ上のどの範囲に設定されているかを調べることによって、上記判定を実行しうる。

上記実行パスに副作用が発生しないパスがある場合、該処理はステップ３０８に進む。上記実行パスが全て副作用の発生するパスの場合、該処理はステップ３１０に進む。

[0090] ステップ３０８では、コンピュータ・システムは、上記副作用が発生しない実行パスと該実行パスを実行させるための条件文とが、以下の要件を満たすかどうかを判定する。上記要件とは、上記実行パスが、インスタンス変数又はクラス変数の高々１つの読み出しを含み、上記条件文が、該読み出しと定数とを評価することである。

要件を満たす条件文が１つでも存在する場合、該処理はステップ３０９に進む。要件を満たす条件文が存在しない場合、該処理はステップ３１０に進む。

[0091] ステップ３０９では、コンピュータ・システムは、上記要件を満たす条件文によって実行されるパスそれぞれについて、上記Ｓの第１の情報であるロックオブジェクト（Ｃの第１の情報）、上記Ｓの第２の情報であるプログラム・コード上の開始終了位置（Ｃの第２の情報）及び上記要件を満たす条件文によって実行される実行パスをプログラム・コードから特定できる情報（Ｃの第３の情報）を、集合変数Ｃに１組の情報として設定する。

上記設定の終了に応じて、該処理はステップ３１０に進む。

[0092] ステップ３１０は、集合変数Ｓについての繰り返しの終了を表す。

全てのｓが取り出し済みの場合、該処理はステップ３１１に進む。取り出していないｓがある場合、該処理はステップ３０６に戻る。

[0093] 図４は、本発明の実施態様における、図１に示す変換プログラムにおいて

実行される処理ステップのうち、変換を行う処理ステップの例を示す。

[0094] ステップ３１１は、集合変数Ｃについての繰り返しの開始を示す。

ステップ３１１では、コンピュータ・システムは、集合変数Ｃから実行パスについての上記１組の情報（以下、ｃ）を１件取り出す。なお、コンピュータ・システムは、プログラム・コードで早く実行される実行パスについての情報から順に上記取り出しを行う。該取り出しが終了することに応じて、該処理はステップ３１２に進む。

[0095] ステップ３１２では、コンピュータ・システムは、上記取り出されたｃに対応する集合変数Ｃに設定されているロックオブジェクト（以下、ｃの第１の情報）が、ＣＬ内のみで使用されているかどうかを判定する。該判定では、例えばＣＬがＪａｖａ（商標）言語で記載されている場合、ロックオブジェクト（ｃの第１の情報）の定義にｐｒｉｖａｔｅ修飾子が使用されていること又は、上記別のＣＬに上記ロックオブジェクト（ｃの第１の情報）を使用する記載が含まれていないことがチェックされる。該チェックにより、ロックオブジェクト（ｃの第１の情報）の定義にｐｒｉｖａｔｅ修飾子が使用されているか又は、上記別のＣＬに上記ロックオブジェクト（ｃの第１の情報）を使用する記載が含まれていない場合、コンピュータ・システムは、ロックオブジェクト（ｃの第１の情報）が、ＣＬ内のみで使用されていると判定する。ここで、例えば、上記別のＣＬが入力データとして与えられていない、又は、上記別のＣＬがバイナリデータとして提供されているためにソースコードの記載が不明であるという理由により、上記チェックが実行できない場合がある。上記チェックが実行できない場合、コンピュータ・システムは、例えば、ロックオブジェクト（ｃの第１の情報）が、ＣＬ内のみで使用されていないと判定してもよい。また、コンピュータ・システムは、例えば、モニタに上記判定をさせるためのメッセージを表示して、ユーザに上記判定をさせてもよい。ＣＬ内でのみ使用されている場合、該処理はステップ３１４に進む。ＣＬ外でも使用されている場合、該処理はステップ３１３に進む。

[0096] ステップ３１３では、コンピュータ・システムは、上記ｃに対応する集合変数Ｃの第３の情報（以下、ｃの第３の情報）に対応するパスを実行させる条件文に含まれる全ての変数がＣＬ内でのみ定義されているかどうかを判定する。該判定では、上記別のＣＬに上記参照される変数の定義が含まれていないことがチェックされる。該チェックにより、上記別のＣＬに上記参照される変数の定義が含まれていない場合、コンピュータ・システムは、参照される変数がＣＬ内でのみ定義されていると判定する。ここで、例えば、上記別のＣＬが入力データとして与えられていない又は、上記別のＣＬがバイナリデータとして提供されているためにソースコードの記載が不明であるという理由により、上記チェックが実行できない場合がある。上記チェックが実行できない場合、コンピュータ・システムは、例えば、上記参照される変数がＣＬ内でのみ定義されていないと判定してもよい。また、コンピュータ・システムは、例えば、モニタに上記判定をさせるためのメッセージを表示して、ユーザに上記判定をさせてもよい。上記参照される変数がＣＬ内でのみ定義されている場合、該処理はステップ３１４に進む。上記参照される変数がＣＬ内でのみ定義されていない場合、該処理はステップ３１８に進む。

[0097] ステップ３１４では、コンピュータ・システムは、ＣＬ内に、変数（以下、第２の変数）を宣言する文を追加する。上記追加する場所は、ＣＬを記述するプログラム言語の仕様によって異なる。なお、第２の変数の宣言は、原子性が保証されているデータ型であって、スタティックであること及び最適化を抑止することが指定可能なデータ型の第２の変数の宣言である。ここで、原子性が保証されているとは、メモリへの変数の読み込みが１回の操作で実行されることが言語仕様によって保証されていることである。Ｊａｖａ（商標）言語であれば、例えばboolean型、byte型、short型、int型又はfloat型は、サイズが３２ビット以下のデータ型であり、例えば３２ビット以上のＣＰＵ上で上記Ｊａｖａ（商標）言語で記載されたプログラムが実行される場合、メモリへの変数の読み込みが１回の操作で実行されることが保証されているデータ型である。スタティックであるとは、ＣＬ内で実体を１つしか

持たないことである。例えば、Ｊａｖａ（商標）言語の場合、変数がスタティックであることは、修飾子staticで指定可能である。また、最適化を抑止するとは、例えば、コンパイラにおいてプログラム・コードがメモリのアクセス順序に関する最適化されることを抑止することである。例えば、Ｊａｖａ（商標）言語の場合、変数の最適化を抑止することは、修飾子volatileで指定可能である。volatile宣言を導入することにより、メモリ・オーダと同期ブロック外への変更との可視性が保証される。

第２の変数の変数名は、他の変数と重複しない名前であればどのような名前であってもよい。該名前は、例えば、変数ｉを含ませることで識別されてもよい。ＣＬが、第２の変数として配列が使用可能な言語で記載されたプログラム・コードであれば、上記名前は、変数ｉを添え字とする配列であってもよい。

上記第２の変数の宣言では、第２の変数がスタティック変数であることを宣言させる。また、上記ｃの第３の情報に対応するパスに含まれる共有変数が、Ｓに情報が設定されている同期ブロック内の文以外からアクセスされる場合、上記第２の変数の宣言では、上記第２の変数が最適化が抑止される対象の変数であることをさらに宣言させる。ここで、共有変数とは、複数のスレッドからアクセスされる可能性のある変数である。ＣＬを記述するプログラム言語がＪａｖａ（商標）言語の場合、上記共有変数は、インスタンス変数及びクラス変数である。なお、上記アクセスされることは、例えばＣＬ又は上記別のＣＬが精査されることで求められる。ここで、例えば、上記別のＣＬが入力データとして与えられていない又は、上記別のＣＬがバイナリデータとして提供されているためにソースコードの記載が不明であるという理由により、上記精査が実行できない場合がある。上記精査が実行できない場合、コンピュータ・システムは、共有変数が同期ブロックの外側からアクセスされるとしてもよい。また、コンピュータ・システムは、例えば、モニタに共有変数が同期ブロックの外側からアクセスされるかどうかをユーザに判断させるメッセージを表示して、ユーザに上記判断をさせてもよい。

上記追加が終了することに応じて、該処理はステップ３１５に進む。

[0098] ステップ３１５では、コンピュータ・システムは、上記ｃの第３の情報に対応する実行パスを実行させる条件文から文を作成する。上記作成される文は、上記実行パスが実行されるための条件が成立した場合と、条件が成立しなかった場合とでそれぞれ違う値を上記第２の変数に代入する文である。

例えば、上記第２の変数がboolean型の変数b1であり、上記条件文の条件式が(x!=null)の場合、上記作成される文は「b1=(x!=null);」とすることができる。別の例では、上記第２の変数がint型の変数i1であり、上記条件文の条件式が(y==0)の場合、上記作成される文は、例えば「if(y==0){i1=0;}else{i1=1;};」とすることができる。

また、上記条件文が、例えば、３通り以上のパスに分岐させる条件文「if(x==0){if(y==0){x++;y--;}else{c2++;}}else{c1++;}（c1及びc2はともにローカル変数）」の場合、上記作成される文は、例えば、「b=(x!==0)?1:(y!=0)?2:0;」とすることができる。ここで、変数bに代入される値は、副作用を持たないパスの中からどれを選ぶかを示す。上記例では、bが1の場合「c1++;」が実行されるパスが選択され、bが2の場合「c2++;」が実行されるパスが選択される文が作成される。

次にコンピュータ・システムは、上記条件式にインスタンス変数が含まれる場合、上記第２の変数の宣言から、上記スタティック変数であることの宣言を削除する。例えばＪａｖａ（商標）言語の場合、上記削除は、修飾子staticを削除することである。

上記削除が終了することに応じて、該処理はステップ３１６に進む。

[0099] ステップ３１６では、コンピュータ・システムは、上記ｃの第３の情報に対応するパスを、上記ｃの第２の情報で示されるＣＬ上の同期ブロックの開始位置の直前に複製する。次に、コンピュータ・システムは、上記第２の変数の値が成立値の場合に上記追加された実行パスからの処理を開始させ、一方上記第２の変数の値が不成立値の場合に上記同期ブロックからの処理を開始させる条件文をＣＬ上に追加する。なお、上記条件文の終端は、値が成立

値の場合及び値が不成立値の場合ともに、上記ｃの第２の情報で示されるＣＬ上の同期ブロックの終了位置の直後になるようにする。

上記条件文の追加が終了することに応じて、該処理はステップ３１７に進む。

[0100] ステップ３１７では、コンピュータ・システムは、上記ｃの第３の情報に対応する実行パスと関係する同期ブロックについての処理を実行する。該処理については、下記図５で詳しく述べる。なお、上記同期ブロックについての処理では、ＣＬに新たな文が追加されうる。また、上記同期ブロックについての処理では、ステップ３１４及び３１６でＣＬに追加された文が削除されうる。上記同期ブロックについての処理が終了することに応じて、該処理はステップ３１８に進む。

[0101] ステップ３１８は、集合変数Ｃについての繰り返しの終了を表す。

全てのｃが取り出し済みの場合、該処理はステップ３１９に進む。取り出していないｃがある場合、該処理はステップ３１１に戻る。

[0102] ステップ３１９では、コンピュータ・システムは、第２の変数について、上記第２の変数を宣言する文と、上記第２の変数に対応する上記ステップ３１４及び３１６でＣＬに追加された文又は上記第２の変数に対応する同期ブロックの文であって最も早く実行される文との間に、上記第２の変数に初期値を設定する文を追加する。上記初期値を設定する文は、ステップ３１５で生成された文である。ここで、上記初期値は、上記不成立値であってもよい。なお、上記文の追加は、定義された全ての第２の変数について実行される。

上記追加が終了することに応じて、該処理はステップ３２０に進む。

[0103] ステップ３２０は、上記変換する処理の終了である。

[0104] 図５は、本発明の実施態様における、図４に記載のステップ３１７の処理の詳細を示す。

[0105] ステップ３２１は、ステップ３１７の処理の開始を示す。

[0106] ステップ３２２では、コンピュータ・システムは、集合変数Ｓからメソッ

ドについての情報（ｓ）を１件取り出す。該取り出しが終了することに応じて、該処理はステップ３２３に進む。

[0107] ステップ３２３では、コンピュータ・システムは、上記ｃの第１の情報であるロックオブジェクトと、上記ｓの第１の情報であるロックオブジェクトが同じかどうかを判定する。同じである場合、該処理はステップ３２４に進む。同じでない場合、該処理はステップ３２９に進む。

[0108] ステップ３２４では、コンピュータ・システムは、上記ｓの第３の情報で表される同期ブロックに、ステップ３１５で作成された文に含まれる変数を更新させる文が含まれるかどうかを判定する。上記更新させる文が含まれる場合、該処理はステップ３２５に進む。上記更新させる文が含まれない場合、該処理はステップ３２９に進む。

[0109] ステップ３２５では、コンピュータ・システムは、上記ｓの第３の情報で表される同期ブロックに、上記ｃの第１の情報であるロックオブジェクトを使用してロックを取得するスレッドを起動する文が含まれるかどうかを判定する。概判定は、コンピュータ・システムが、上記同期ブロックを精査することで実行される。なお、上記精査において、上記ロックオブジェクトを、上記同期ブロック内で使用させる文（以下、第１の文）が見つかる場合がある。上記第１の文は、例えば、上記ロックオブジェクトを引数とするメソッドである。上記第１の文が見つかった場合、コンピュータ・システムは、例えば、上記別のＣＬを精査して、上記第１の文によって実行される実行パスを求める。コンピュータ・システムは、上記求められた実行パスを精査することで、上記スレッドを起動する文を見つけうる。

上記精査により、上記スレッドを起動する文が見つからなかった場合、該処理はステップ３２６に進む。見つかった場合、該処理はステップ３２８に進む。

ここで、例えば、上記別のＣＬが入力データとして与えられていない又は、上記別のＣＬがバイナリデータとして提供されているためにソースコードの記載が不明であるという理由により、上記精査が実行できない場合がある

。上記精査が実行できない場合、コンピュータ・システムは、上記スレッドを起動する文が、上記ｃの第３の情報に対応する実行パスに含まれると判定してもよい。また、コンピュータ・システムは、例えば、モニタに上記スレッドを起動する文が含まれるかどうかを入力させるメッセージを表示して、ユーザに上記判定をさせてもよい。

[0110] ステップ３２６では、コンピュータ・システムは、上記ｓの第３の情報に対応する同期ブロック内で定義されている変数が、上記同期ブロック外で参照され且つスレッド間の実行順序付けに使用されているかを判定する。概判定は、コンピュータ・システムが、上記プログラム・コードを精査することで実行される。上記精査により、上記変数が見つからなかった場合、該処理はステップ３２７に進む。見つかった場合、該処理はステップ３２８に進む。

ここで、例えば、上記別のＣＬが入力データとして与えられていない、又は、上記別のＣＬがバイナリデータとして提供されているためにソースコードの記載が不明であるという理由により、上記精査が実行できない場合がある。上記精査が実行できない場合、コンピュータ・システムは、上記変数が、上記ｃの第３の情報に対応する実行パスに含まれると判定してもよい。また、コンピュータ・システムは、例えば、モニタに上記変数が含まれるかどうかを入力させるメッセージを表示して、ユーザに上記判定をさせてもよい。

[0111] ステップ３２７では、コンピュータ・システムは、ステップ３１５において作成した文を、ＣＬに追加する。ここで、第２の変数が、変数の最適化を抑止することが指定されている変数である場合、上記追加は、ｓに対応する同期ブロックについて実行される。また、上記追加されるＣＬ上の位置は、上記ｓに対応する同期ブロックから抜けるための文の直前である。

第２の変数が、変数の最適化を抑止することが指定されていない変数である場合、上記追加されるＣＬ上の位置は、上記ステップ３２４で見つかった上記更新させる文の後であって、上記ｓに対応する同期ブロックを抜ける処

理の前である。

上記追加が終了することに応じて、該処理はステップ３２９に進む。

[0112] ステップ３２８では、コンピュータ・システムは、ステップ３１４及び３１６において、ＣＬに追加された文を削除する。

上記削除が終了することに応じて、ステップ３２２についての繰り返しは終了し、該処理はステップ３３０に進む。

[0113] ステップ３２９は、集合変数Ｓについての繰り返しの終了を表す。

全てのｓが取り出し済みの場合、コンピュータ・システムは、変数ｉをインクリメントする。該インクリメントが終了することに応じて、該処理はステップ３３０に進む。取り出していないｓがある場合、該処理はステップ３２２に戻る。

[0114] ステップ３３０は、ステップ３１７の処理の終了を示す。

[0115] 図３～図５の各ステップの動作の具体的な例を下記図６～図８及び図９～図１１に示す。

[0116] 図６は、本発明の実施態様における、図３～図５に記載の各ステップの動作を具体的に説明するために使用する入力データＣＬを示す。

なお、図６～図８で示される例は、ステップ３２５及びステップ３２６において、ユーザに問い合わせを行う場合を示す。

[0117] コンピュータ・プログラムは、入力データ（４０１）を入力ＣＬとして、変換プログラムを起動する。

ステップ３０２では、コンピュータ・システムは、変数に以下の値を設定する。

集合変数Ｍ = {void enqueue(W w) public W dequeue()}

変数ｉ = 1

集合変数Ｓ= {φ}

集合変数Ｃ = {φ}

上記設定が終了することに応じて、該処理はステップ３０３に進む。

ステップ３０３～ステップ３０５の繰り返しでは、コンピュータ・システ

ムは、m=“boolean enqueue(W w)”及びm=“public W dequeue()”それぞれについて、ステップ３０４を実行する。該実行により、集合変数Ｓには、

Ｓ = {

<w、(S01,S08)、(S02,S03,S04,S05,S06,S07)>（以下、ｓ１），

<lock、(S03,S07)、(S04,S05,S06)> （以下、ｓ２），

<lock、(T01,T09)、(T02,T03,T04,T05,T06,T07,T08)>（以下、ｓ３）

}

が保存される。

上記保存が終了することに応じて、該処理はステップ３０６に進む。

ステップ３０６～ステップ３１０の繰り返しでは、コンピュータ・システムは、上記集合変数Ｓに含まれるｓ１～ｓ３それぞれについて、ステップ３０７～ステップ３０９を実行する。該実行の結果、集合変数Ｃには、

Ｃ = {

<lock、(T01,T09)、(T02,T07,T08)>（以下、ｃ）

}

が保存される。

ここで、S03行目を含むパスには、同期という副作用がある。また、S04行目及びT05行目を含むパスには、インスタンス変数への代入という副作用が存在する。よって、S03行目、S04行目及びT05行目を含むパスは、ステップ３０７の条件を満たさない。従って、S03行目、S04行目及びT05行目を含むパスは、ステップ３０９において、集合変数Ｃに保存されることはない。また、例えば、S02行目では、wというメソッドの引数から渡されるローカル変数が参照されている。よって、S02行目は、ステップ３０８の条件を満たさない。従って、S02行目は、ステップ３０９において、集合変数Ｃに保存されることはない。　次に、該処理はステップ３１１に進む。

[0118] ステップ３１１～ステップ３１８の繰り返しでは、コンピュータ・システムは、上記集合変数Ｃに含まれるｃについて、ステップ３１２～３１７を実行する。

[0119] ステップ３１２及びステップ３１３では、上記ｃについての判定が実行される。ここで、ｃに含まれる“lock”は、privateで宣言された変数であり、ＣＬ外からは参照されない変数である。よって、ｃはステップ３１２及びステップ３１３の条件を満たすので、該処理はステップ３１４に進む。

[0120] ステップ３１４では、コンピュータ・システムは、変数を宣言する文をＣＬに追加する。ここで、共有変数headは、ｓ１内でのみアクセスされる。よって、コンピュータ・システムは、インスタンス変数の宣言“private static boolean b1”をＣＬに追加する。

[0121] ステップ３１５では、コンピュータ・システムは、T02行目の条件文(head != null)を用いて、“b1 = (head != null)”という文を生成する。また、headはインスタンス変数であるので、コンピュータ・システムは、ステップ３１４で追加された宣言“private static boolean b1”から、“static”を取り除く。

[0122] ステップ３１６では、コンピュータ・システムは、T02行目、T07行目及びT08行目をT01行目の前に生成する。また、コンピュータ・システムは、ｃに含まれる条件文を“(b1)”で置き換える。コンピュータ・システムは、T02行目のthen節が副作用を持つパスへの分岐であるので、上記then節を同期ブロックへのコードに繋がるようにする。

[0123] 図７は、本発明の実施態様における、図４に記載のステップ３１６の処理が実行された後のメソッドdequeue()を示す。

ステップ３１６の処理によって、ＣＬのメソッドdequeue()には、文（４１２）が追加されている。

[0124] 上記ステップ３１６において、上記追加が実行されることに応じて、該処理はステップ３１７に進む。

[0125] ステップ３１７では、コンピュータ・システムは、ステップ３２２～ステップ３２９の繰り返しを実行する。

[0126] ステップ３２２～ステップ３２９の繰り返しでは、コンピュータ・システムは、上記集合変数Ｓに含まれるｓ１～ｓ３それぞれについて、ステップ３

２３～ステップ３２８を実行する。

ステップ３２２の条件を満たすｓ１～ｓ３は、

〈lock、(S03,S07)、(S04,S05,S06)〉（ｓ２）及び

〈lock、(T01,T09)、(T02,T03,T04,T05,T06,T07,T08)〉（ｓ３）である。

[0127] ステップ３２３では、コンピュータ・システムは、上記ステップ３２２の条件を満たすｓ２又はｓ３に関するパスに、b0の代入文の右辺で参照される変数への代入が１つでもあるかを調べる。ｓ２又はｓ３においては、headへの代入をする処理が検索される。上記検索の結果、ｓ２又はｓ３は、ステップ３２３の条件を満たす。

[0128] ステップ３２４では、コンピュータ・システムは、上記ｓ２又はｓ３で表される同期ブロックについて、新たなスレッドが生成されて、生成されたスレッドが、上記ｓ２又はｓ３で表される同期ブロックと同じロックオブジェクトを獲得しようとしていないかどうかをチェックする。該チェックにより、ｓ２で表される同期ブロックのS06行目に、L0を引数としてメソッドbar(lock)を呼び出す処理が含まれていることが発見される。

ここで、入力がＣＬのみの場合、上記メソッドbar(lock)を実装するプログラムのプログラム・コードは不明である。よって、コンピュータ・システムは、ユーザに上記獲得しようとしていないかどうかを判定させる。コンピュータ・システムは、例えばモニタに「foo.bar(lock)は、スレッドを起動してlockに関してsynchronized block又はsynchronized methodを呼んでいますか？」と表示し、ユーザに対して問い合わせを行う。ユーザが「N」と答えた場合、該処理はステップ３２６に進み、ユーザが「Y」と答えた場合、該処理はステップ３２８に進む。

[0129] ステップ３２６では、コンピュータ・システムは、ｓ２又はｓ３で表される同期ブロック内で定義された変数が、同期ブロック外で参照され且つ少なくとも２のスレッド間の実行順序を決定するために用いられていないかをチェックする。ここで、S06行目の共有変数lockは、foo.bar(lock)からエスケープして他のスレッドからアクセスされる可能性がある。よって、コンピュ

ータ・システムは、例えばモニタに「lockは他のスレッドと実行順序を決定するために用いられますか？」と表示し、ユーザに対して問い合わせを行う。ユーザが「N」と答えた場合、該処理はステップ３２７に進み、ユーザが「Yと答えた場合、該処理はステップ３２８に進み、上記繰り返しを抜ける。

[0130] ステップ３２７では、コンピュータ・システムは、ステップ３１５において生成された文“b1 = (head != null)”を追加する位置を決定する。なお、b1はvolatile宣言がされていないため、コンピュータ・システムは、上記生成された文をheadを更新させる文より後ろに追加する。該追加する場所は、例えば、ｓ２については、S06行目の後に、ｓ３については、T06行目の前とT08行目の前である。

[0131] 上記ステップ３２２～ステップ３２９の繰り返しが終了することに応じて、該処理はステップ３１９に進む。

[0132] ステップ３１９では、コンピュータ・システムは、ステップ３１４において追加された宣言に“b1 = (head != null);”を追加する。

上記追加されることに応じて、該処理は終了する。

[0133] 図８は、本発明の実施態様における、図６について、図３～図５に記載のステップの処理が実行された後の入力データ（以下、変換後のプログラム・コード（４２１））を示す。

図３～図５のステップの処理によって、変換後のプログラム・コード（４２１）には、文（４２２）が追加されている。

変換後のプログラム・コード（４２１）は、headがnullの場合、b1はfalseとなる。よって、変換後のプログラム・コード（４２１）が実行され、headがnullのときにdequeue()メソッドが呼び出されると、該処理は同期ブロックに突入することなくnullを返す。従って、ロック衝突の問題が解決されている。

[0134] 図９は、本発明の実施態様における、図３～図５に記載の各ステップの動作を具体的に説明するために使用する入力データＣＬを示す。

なお、図９～図１１で示される例は、上記ＣＬの他に別のＣＬ（図示せず

）が存在し、ステップ３２５及びステップ３２６において、ユーザに問い合わせを行わない場合の例である。

[0135] コンピュータ・プログラムは、入力データ（５０１）を入力ＣＬとして、変換プログラムを起動する。

ステップ３０２では、コンピュータ・システムは、変数に以下の値を設定する。

集合変数Ｍ = {boolean enqueue(W w) public W dequeue()}

変数ｉ = 1

集合変数Ｓ= {φ}

集合変数Ｃ = {φ}

上記設定が終了することに応じて、該処理はステップ３０３に進む。

[0136] ステップ３０３～ステップ３０５の繰り返しでは、コンピュータ・システムは、m=“boolean enqueue(W w)”及びm=“public W dequeue()”それぞれについて、ステップ３０４を実行する。該実行により、集合変数Ｓには、

Ｓ = {

<w、(S01,S08)、(S02,S03,S04,S05,S06,S07)>（以下、ｓ１），

<lock、(S03,S07)、(S04,S05,S06)> （以下、ｓ２），

<lock、(T01,T09)、(T02,T03,T04,T05,T06,T07,T08)>（以下、ｓ３）

}

が保存される。

上記設定が終了することに応じて、該処理はステップ３０６に進む。

[0137] ステップ３０６～ステップ３１０の繰り返しでは、コンピュータ・システムは、上記集合変数Ｓに含まれるｓ１～ｓ３それぞれについて、ステップ３０７～ステップ３０９を実行する。該実行の結果、集合変数Ｃには、

Ｃ = {

<lock, (T01, T09), (T02, T07, T08)>（以下、ｃ）

}

が保存される。

ここで、S03行目を含むパスには、同期という副作用がある。また、S04行目及びT05行目を含むパスには、インスタンス変数への代入という副作用が存在する。よって、S03行目、S04行目及びT05行目を含むパスは、ステップ３０７の条件を満たさない。従って、S03行目、S04行目及びT05行目を含むパスは、ステップ３０９において、集合変数Ｃに保存されることはない。また、例えば、S02行目では、wというメソッドの引数から渡されるローカル変数が参照されている。よって、S02行目は、ステップ３０８の条件を満たさない。従って、S02行目は、ステップ３０９において、集合変数Ｃに保存されることはない。　次に、該処理はステップ３１１に進む。

[0138] ステップ３１１～ステップ３１８の繰り返しでは、コンピュータ・システムは、上記集合変数Ｃに含まれるｃについて、ステップ３１２～３１７を実行する。

[0139] ステップ３１２及びステップ３１３では、上記ｃについての判定が実行される。ここで、ｃに含まれる“lock”は、privateで宣言された変数であり、ＣＬ外からは参照されない変数である。よって、ｃはステップ３１２及びステップ３１３の条件を満たすので、該処理はステップ３１４に進む。

[0140] ステップ３１４では、コンピュータ・システムは、変数を宣言する文をＣＬに追加する。ここで、共有変数headは、ｓ１内でのみアクセスされる。よって、コンピュータ・システムは、インスタンス変数の宣言“private static boolean b1”をＣＬに追加する。

[0141] ステップ３１５では、コンピュータ・システムは、T02行目の条件文(head != null)を用いて、“b1 = (head != null)”という文を生成する。また、headはインスタンス変数であるので、コンピュータ・システムは、ステップ３１４で追加された宣言“private static boolean b1”から、“static”を取り除く。

[0142] ステップ３１６では、コンピュータ・システムは、T02行目、T07行目及びT08行目をT01行目の前に生成する。また、コンピュータ・システムは、ｃに含まれる条件文を“(b1)”で置き換える。コンピュータ・システムは、T02行目

のthen節が副作用を持つパスへの分岐であるので、上記then節を同期ブロックへのコードに繋がるようにする。

[0143] 図１０は、本発明の実施態様における、図４に記載のステップ３１６の処理が実行された後のメソッドdequeue()を示す。

ステップ３１６の処理によって、ＣＬのメソッドdequeue()には、文（５１２）が追加されている。

[0144] 上記ステップ３１６において、上記追加が実行されることに応じて、該処理はステップ３１７に進む。

[0145] ステップ３１７では、コンピュータ・システムは、ステップ３２２～ステップ３２９の繰り返しを実行する。

[0146] ステップ３２２～ステップ３２９の繰り返しでは、コンピュータ・システムは、上記集合変数Ｓに含まれるｓ１～ｓ３それぞれについて、ステップ３２３～ステップ３２８を実行する。

ステップ３２２の条件を満たすｓ１～ｓ３は、

〈lock、(S03,S07)、(S04,S05,S06)〉（ｓ２）及び

〈lock、(T01,T09)、(T02,T03,T04,T05,T06,T07,T08)〉（ｓ３）である。

ステップ３２３では、コンピュータ・システムは、上記ステップ３２２の条件を満たすｓ２又はｓ３に関するパスに、b0の代入文の右辺で参照される変数への代入が１つでもあるかを調べる。ｓ２又はｓ３において、headへの代入をする処理が検索される。上記検索の結果、ｓ２又はｓ３は、ステップ３２３の条件を満たす。

[0147] ステップ３２４では、コンピュータ・システムは、上記ｓ２又はｓ３で表される同期ブロックについて、新たなスレッドが生成され、該生成されたスレッドが、上記ｓ２又はｓ３で表される同期ブロックと同じロックオブジェクトを獲得しようとしていないかどうかをチェックする。上記ｓ２又はｓ３で表される同期ブロックには、新たなスレッドを生成する文は含まれない。よって、コンピュータ・システムは、該チェックにおいて、上記ｓ２又はｓ３で表される同期ブロックと同じロックオブジェクトを獲得しようとしてい

ないと判断する。該獲得しようとしていないことに応じて、該処理はステップ３２６に進む。

[0148] ステップ３２６では、コンピュータ・システムは、ｓ２又はｓ３で表される同期ブロック内で定義された変数が、同期ブロック外で参照され且つ少なくとも２のスレッド間の実行順序を決定するために用いられていないかをチェックする。ここで、共有変数lockは、private宣言されているためにＣＬ外からはアクセスされない。また、共有変数lockは、ＣＬ内では、同期ブロック内でのみアクセスされる。よって、コンピュータ・システムは、該チェックにおいて、共有変数lockは実行順序を決定するために用いられていないと判断する。該用いられていないことに応じて、該処理はステップ３２７に進む。

[0149] ステップ３２７では、コンピュータ・システムは、ステップ３１５において生成された文“b1 = (head != null)”を追加する位置を決定する。なお、b1はvolatile宣言がされていないため、コンピュータ・システムは、上記生成された文をheadを更新させる文より後ろに追加する。該追加する場所は、例えば、ｓ２については、S06行目の前に、ｓ３については、T06行目の前とT08行目の前である。

[0150] 上記ステップ３２２～ステップ３２９の繰り返しが終了することに応じて、該処理はステップ３１９に進む。

[0151] ステップ３１９では、コンピュータ・システムは、ステップ３１４において追加された宣言に“b1 = (head != null);”を追加する。

上記追加されることに応じて、該処理は終了する。

[0152] 図１１は、本発明の実施態様における、図９について、図３～図５に記載の各ステップの処理が実行された後の入力データ（以下、変換後のプログラム・コード（５２１））を示す。

図３～図５のステップの処理によって、変換後のプログラム・コード（５２１）には、文（５２２）が追加されている。

変換後のプログラム・コード（５２１）は、headがnullの場合、b1はfalse

となる。よって、変換後のプログラム・コード（４０１）が実行され、headがnullのときにdequeue()メソッドが呼び出されると、該処理は同期ブロックに突入することなくnullを返す。従って、ロック衝突の問題が解決されている。

[0153] 図１２は、本発明の実施態様における、第１の条件式によって選択されるパスが３通り以上である場合の、図３～図５に記載の各ステップの処理が実行される前のプログラム・コードの一部及び実行された後のプログラム・コードの一部を示す。

プログラム・コード（６０１）の同期ブロックは、２つの第１の条件式「（x==0）」及び「（y==0）」を含む。また、上記２つの第１の条件式により、３通りのパス（６０２～６０４）が選択される。ここで、c1及びc2がローカル変数の場合、パス（６０３）及びパス（６０４）は、副作用のないパスである。なお、パス（６０２）は、第１の変数に対応するx及びyの２つの変数を更新させる文を含むため、副作用のあるパスである。

[0154] プログラム・コード（６１１）は、プログラム・コード（６０１）に対して、図３～図５に記載の各ステップの処理が実行された結果を示す。

プログラム・コード（６１１）には、プログラム・コード（６０１）の同期ブロックに、第２の変数を宣言する宣言文（６１２）、第２の条件文（６１３）及びステップ３１５で作成される文（６１４）が追加されている。

上記各ステップの処理において、第２の変数には、副作用のないパス（６０３及び６０４）の中からどのパスを選ぶかを示す値をとることができる変数が用意される。よって、プログラム・コード（６１１）には、パス（６０３）を選ぶ値として2を、パス（６０４）を示す値として1を取ることができる変数bを宣言する宣言文（６１２）が追加されている。さらに、プログラム・コード（６１１）には、上記第２の変数に応じて副作用のないパス（６０３及び６０４）を実行させる条件文として、第２の条件文（６１３）が追加されている。

プログラム・コード（６１１）にはまた、第１の変数であるx及びyを更新

させる更新文の直後に、変数ｂの値を求める文であるステップ３１５で作成される文（６１４）が追加されている。

[0155] 本発明の実施態様によれば、図３～図５の各ステップを実行することによって、java.lang.ref.ReferenceQueueにおけるロック衝突の問題が解決可能である。

[0156] 図１３は、本発明の実施態様における、図１のシステムが有するコンピュータ・ハードウェアのブロック図を示す。

本発明の実施形態に係るコンピュータ・システム（７０１）は、ＣＰＵ（７０２）とメイン・メモリ（７０３）と含み、これらはバス（７０４）に接続されている。ＣＰＵ（７０２）は好ましくは、３２ビット又は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Ｘｅｏｎ（商標）シリーズ、Ｃｏｒｅ（商標）シリーズ、Ａｔｏｍ（商標）シリーズ、Ｐｅｎｔｉｕｍ（商標）シリーズ、Ｃｅｌｅｒｏｎ（商標）シリーズ、ＡＭＤ社のＰｈｅｎｏｍ（商標）シリーズ、Ａｔｈｌｏｎ（商標）シリーズ、Ｔｕｒｉｏｎ（商標）シリーズ又はＳｅｍｐｒｏｎ（商標）などを使用することができる。バス（７０４）には、ディスプレイ・コントローラ（７０５）を介して、ＬＣＤモニタなどのディスプレイ（７０６）が接続されている。ディスプレイ（７０６）は、コンピュータ・システムの管理のために、通信回線を介してネットワークに接続されたコンピュータ・システムについての情報と、そのコンピュータ・システム上で動作中のソフトウェアについての情報を、適当なグラフィック・インターフェースで表示するために使用される。バス（７０４）にはまた、ＩＤＥ又はＳＡＴＡコントローラ（７０７）を介して、ハードディスク又はシリコン・ディスク（７０８）と、ＣＤ－ＲＯＭ、ＤＶＤドライブ又はＢｌｕ－ｒａｙドライブ（７０９）が接続されている。

[0157] ハードディスク（７０８）には、オペレーティング・システム、Ｊ２ＥＥなどのＪａｖａ（商標）処理環境を提供するプログラム、その他のプログラム及びデータが、メイン・メモリ（７０３）にロード可能なように記憶され

ている。

[0158] ＣＤ－ＲＯＭ、ＤＶＤ又はＢｌｕ－ｒａｙドライブ（７０９）は、必要に応じて、ＣＤ－ＲＯＭ、ＤＶＤ－ＲＯＭ又はＢｌｕ－ｒａｙディスクからプログラムをハードディスクに追加導入するために使用される。バス（７０４）には更に、キーボード・マウスコントローラ（７１０）を介して、キーボード（７１１）及びマウス（７１２）が接続されている。

[0159] 通信インタフェース（７１４）は、例えばイーサネット（商標）・プロトコルに従う。通信インタフェース（７１４）は、通信コントローラ（７１３）を介してバス（７０４）に接続され、コンピュータ・システム及び通信回線（７１５）を物理的に接続する役割を担い、コンピュータ・システムのオペレーティング・システムの通信機能のＴＣＰ／ＩＰ通信プロトコルに対して、ネットワーク・インターフェース層を提供する。なお、通信回線は、有線ＬＡＮ環境、或いは例えばＩＥＥＥ８０２．１１ａ／ｂ／ｇ／ｎなどの無線ＬＡＮ接続規格に基づく無線ＬＡＮ環境であってもよい。

請求の範囲

[請求項1] マルチスレッド上で動作するプログラムのプログラム・コードをロック衝突が少ないプログラム・コードに変換する方法であって、

前記プログラム・コードをメモリ内に読み出し、該読み出したプログラム・コードから、同期ブロック内にあり且つ該同期ブロックに対する副作用がないパスへ分岐させる第１の条件文を検索するステップであって、前記副作用がないとは前記同期ブロック内の文による処理中の値が前記同期ブロック外の文によって参照できないことである、前記検索するステップと、

前記同期ブロック外に、前記検索された第１の条件文によって分岐される副作用のないパスを複製するステップと、

前記複製に応じて、前記プログラム・コード内に第２の条件文を追加するステップであって、前記第２の条件文は前記複製された副作用のないパスへ分岐させる条件文である、前記追加するステップと

を含む、前記方法。

[請求項2] 前記第１の条件文が第１の条件式を含み、該第１の条件式は、第１の変数の読み出し、定数又はそれらの組み合わせに基づいてパスを選択させるように定義されており、前記第１の変数は、前記第１の条件文を含むサブルーチンを呼び出すルーチンの中だけが有効範囲である変数ではない、請求項１に記載の方法。

[請求項3] 前記第２の条件文が、前記第１の条件式の結果を格納するための変数に基づいて、前記第１の条件式の結果に応じてパスを選択させる第２の条件式を含む、請求項２に記載の方法。

[請求項4] 前記検索するステップが、前記読み出したプログラム・コードから、更新文をさらに検索するステップを含み、前記更新文は、前記同期ブロック内にあり且つ前記第１の変数を更新させ、

前記追加するステップが、前記検索された更新文の直後から前記同期ブロックの終了文までの間に、前記第１の条件式の結果を格納する

ための変数と、該変数に代入される前記第１の条件式の結果とを含む文をさらに追加するステップを含む、

請求項３に記載の方法。

[請求項5] 前記検索するステップが、前記読み出したプログラム・コード及び他のプログラム・コードの少なくとも１つから、更新文をさらに検索するステップを含み、前記更新文は、前記同期ブロックと同期する他の同期ブロック内にあり且つ前記第１の変数を更新させ、

前記追加するステップが、前記検索された更新文の直後から前記他の同期ブロックの終了文までの間に、前記第１の条件式の結果を格納するための変数と、該同じ変数に代入される前記第１の条件式の結果とを含む文をさらに追加するステップを含む、

請求項３に記載の方法。

[請求項6] 前記追加するステップが、前記第１の条件式の結果を格納するための変数が初めて参照される文より前に、前記第２の条件文が前記同期ブロックを含むパスを実行させる値を用いて又は前記第１の条件式の結果を用いて前記第１の条件式の結果を格納するための変数を初期化する文をさらに追加するステップを含む、請求項３に記載の方法。

[請求項7] 前記第１の条件式の数が２以上である場合、該２以上の第１の条件式の結果が、前記２以上の第１の条件式の組み合わせで求められる、請求項２に記載の方法。

[請求項8] 前記第１の条件式によって選択されるパスの数が３通り以上である場合、

前記第１の条件式の結果を格納するための変数が、前記３通り以上のパスのうち副作用のないパスそれぞれに対応する、請求項３に記載の方法。

[請求項9] 前記第１の条件式の結果を格納するための変数が、原子性が保証されたデータ型の変数である、請求項３に記載の方法。

[請求項10] 前記検索するステップが、前記プログラム・コード及び他のプログ

ラム・コードの少なくとも１つから、前記同期ブロックと同期する他の同期ブロックをさらに検索するステップを含み、

前記検索の終了に応じて、前記複製するステップを実行する、請求項１に記載の方法。

[請求項11] 前記検索するステップが、前記プログラム・コード及び他のプログラム・コードの少なくとも１つに含まれており且つ前記同期ブロック外の文から前記第１の変数を参照する文をさらに検索するステップを含み、

前記検索の終了に応じて、前記複製するステップを実行する、請求項２に記載の方法。

[請求項12] 前記同期ブロック内にスレッドを起動させる文が含まれる場合、

前記検索するステップが、前記プログラム・コード及び他のプログラム・コードの少なくとも１つに含まれており且つ前記スレッドの実装を記載する文から、前記同期ブロック内の文と同期をとって実行される文をさらに検索するステップを含み、

前記同期をとって実行される文が検索されることに応じて、前記スレッドを起動させる文を含むパスを副作用のあるパスと判定するステップ

をさらに含む、請求項１に記載の方法。

[請求項13] 前記同期ブロック内にスレッドを起動させる文が含まれる場合、

前記検索するステップが、前記スレッドを起動させる文を含むパスが副作用のないパスかどうかをユーザに判断することを求めるステップをさらに含む、

請求項１記載の方法。

[請求項14] 前記同期ブロックが実行されるスレッドの実行順序を決定するために使用される変数が前記同期ブロック内に定義されている場合、

前記検索するステップが、前記プログラム・コード及び他のプログラム・コードの少なくとも１つに含まれており且つ前記使用される変

数を参照する文を前記同期ブロック外からさらに検索するステップを含み、

前記使用される変数を参照する文が検索されることに応じて、前記使用される変数を含むパスを副作用のあるパスと判定するステップをさらに含む、

請求項１に記載の方法。

[請求項15] 前記同期ブロックが実行されるスレッドの実行順序を決定するために使用される変数が前記同期ブロック内に定義されている場合、

前記検索するステップが、前記使用される変数を含むパスが副作用のないパスかどうかをユーザに判断することを求めるステップをさらに含む、

請求項１に記載の方法。

[請求項16] 前記追加するステップが、前記読み出したプログラム・コードに、前記第１の条件式の結果を格納するための変数を宣言する文をさらに追加するステップを含み、

共有変数が前記プログラム・コード及び前記他のプログラム・コードの少なくとも１つに含まれており且つ前記同期ブロック外の文からアクセスされる場合に、前記宣言がメモリのアクセス順序に関する最適化を抑止することの指定及び原子性を保証することの指定を含み、前記共有変数は前記副作用のないパスに含まれており、且つ複数のスレッドからアクセスされることができ、

上記アクセスされない場合に、前記宣言が原子性を保証することの指定を含む、

請求項３に記載の方法。

[請求項17] 前記検索するステップが、前記同期ブロックと同期する他の同期ブロックがあるかどうかをユーザに判断させることを許すステップをさらに含み、

該許すステップの後に、前記複製するステップを実行する、請求項

１に記載の方法。

[請求項18] 前記検索するステップが、前記第１の変数を参照する文があるかどうかをユーザに判断させることを許すステップをさらに含み、

該許すステップの後に、前記複製するステップを実行する、請求項２に記載の方法。

[請求項19] マルチスレッド上で動作するプログラムのプログラム・コードをロック衝突が少ないプログラム・コードに変換するコンピュータ・システムであって、

前記プログラム・コードをメモリ内に読み出し、該読み出したプログラム・コードから、同期ブロック内にあり且つ該同期ブロックに対する副作用がないパスへ分岐させる第１の条件文を検索する検索部であって、前記副作用がないとは前記同期ブロック内の文による処理中の値が前記同期ブロック外の文によって参照できないことである、前記検索部と、

前記同期ブロック外に、前記検索された第１の条件文によって分岐される副作用のないパスを複製する複製部と、

前記複製に応じて、前記プログラム・コード内に第２の条件文を追加する追加部であって、前記第２の条件文は前記複製された副作用のないパスへ分岐させる条件文である、前記追加部と

を含む、前記コンピュータ・システム。

[請求項20] マルチスレッド上で動作するプログラムのプログラム・コードをロック衝突が少ないプログラム・コードに変換する方法であって、

前記プログラム・コードをメモリ内に読み出し、該読み出したプログラム・コードから、同期ブロック内にあり且つ該同期ブロックに対する副作用がないパスへ分岐させる第１の条件文を検索するステップであって、前記副作用がないとは前記同期ブロック内の文による処理中の値が前記同期ブロック外の文によって参照できないことである、前記検索するステップと、

前記同期ブロック外に、前記検索された第１の条件文によって分岐される副作用のないパスを複製するステップと、

前記複製に応じて、前記プログラム・コード内に第２の条件文を追加するステップであって、前記第２の条件文は前記複製された副作用のないパスへ分岐させる条件文である、前記追加するステップと

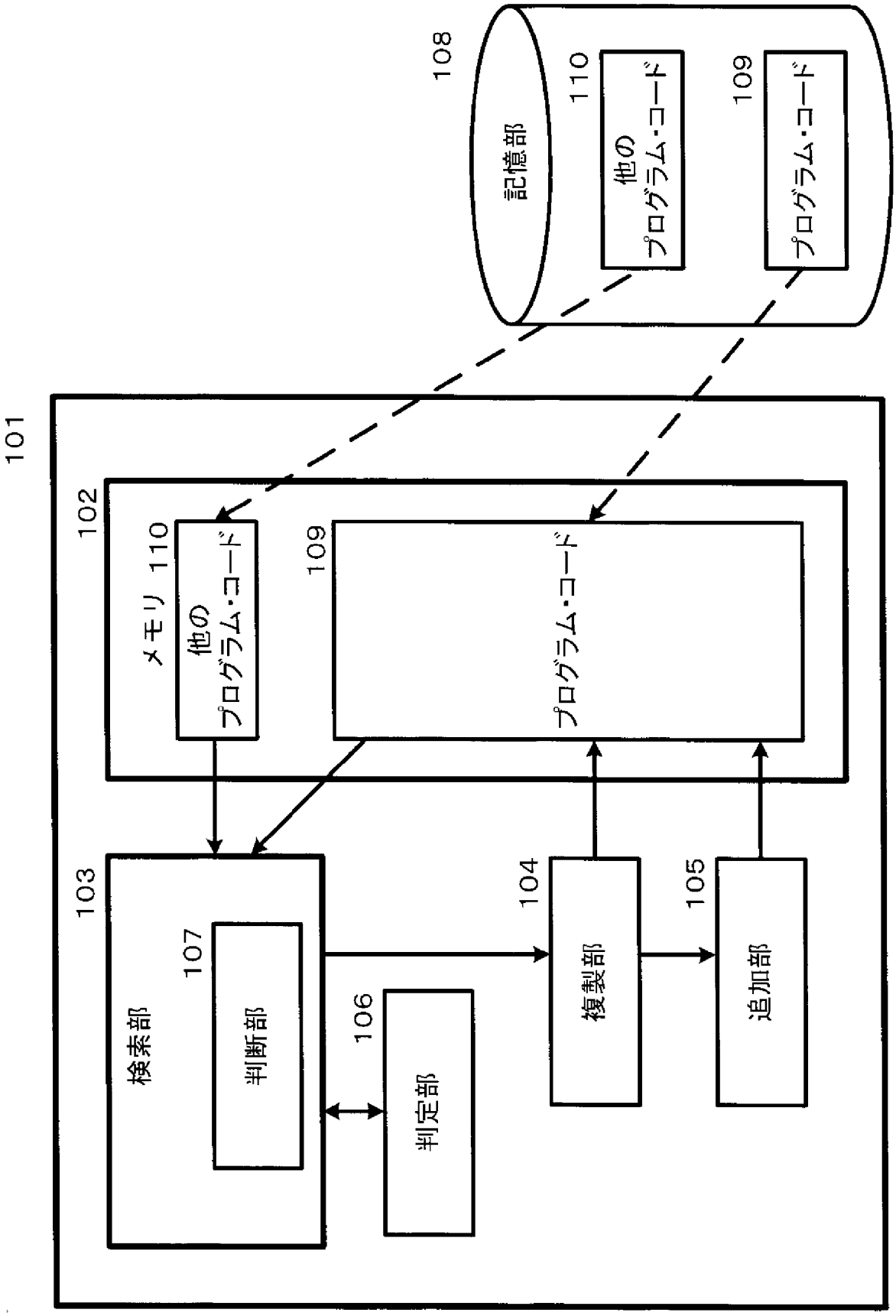
を含み、

前記第１の条件文が第１の条件式を含み、該第１の条件式は、インスタンス変数若しくはクラス変数の読み出し、定数又はそれらの組み合わせに基づいて、副作用のないパスに分岐させる場合にその結果がtrueとなるように定義されており、一方副作用のあるパスに分岐させる場合にその結果がfalseとなるように定義されており、

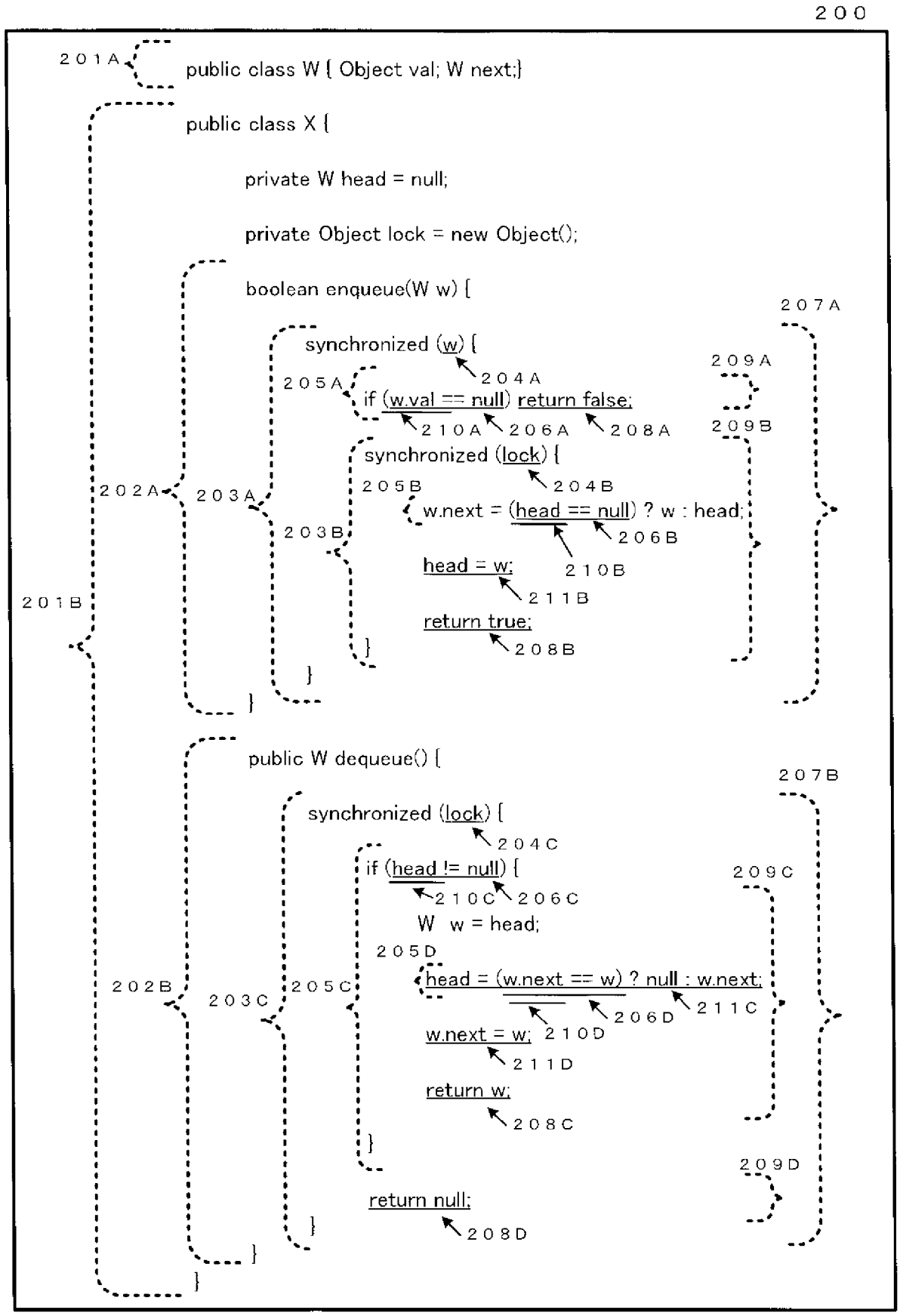
前記第２の条件文が第２の条件式を含み、該第２の条件式は、前記第１の条件式の結果を格納するためのboolean型の変数に基づいて、前記第１の条件式の結果に応じてパスを選択させるように定義されている、前記方法。

[請求項21] コンピュータ・システムに、請求項１～１８のいずれか１項に記載の方法の各ステップを実行させるコンピュータ・プログラム。

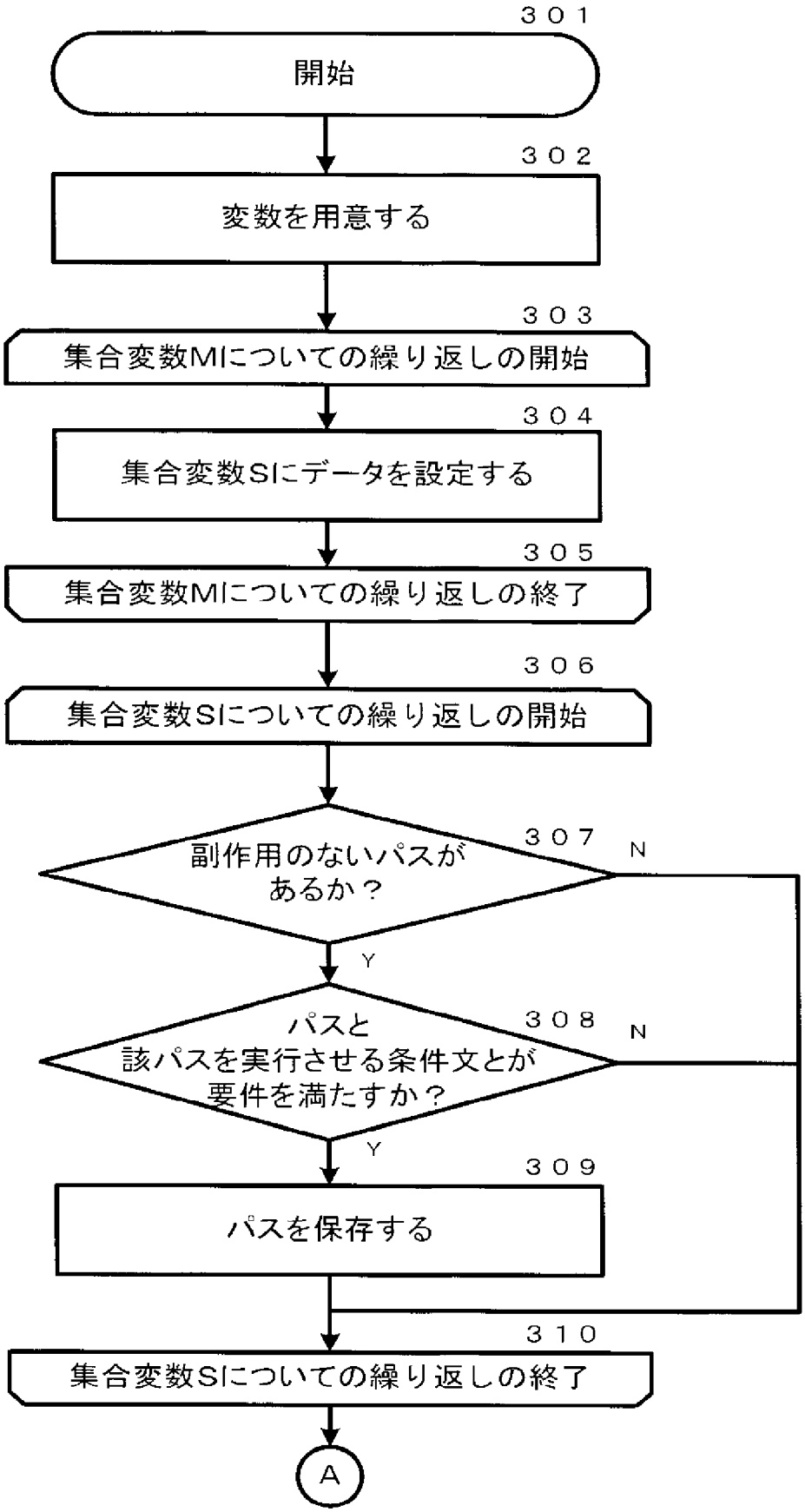
[図1]


101
102
メモリ
110
他の
プログラム・コード
109
プログラム・コード
103
検索部
107
判断部
106
判定部
104
複製部
105
追加部
108
記憶部
110
他の
プログラム・コード
109
プログラム・コード

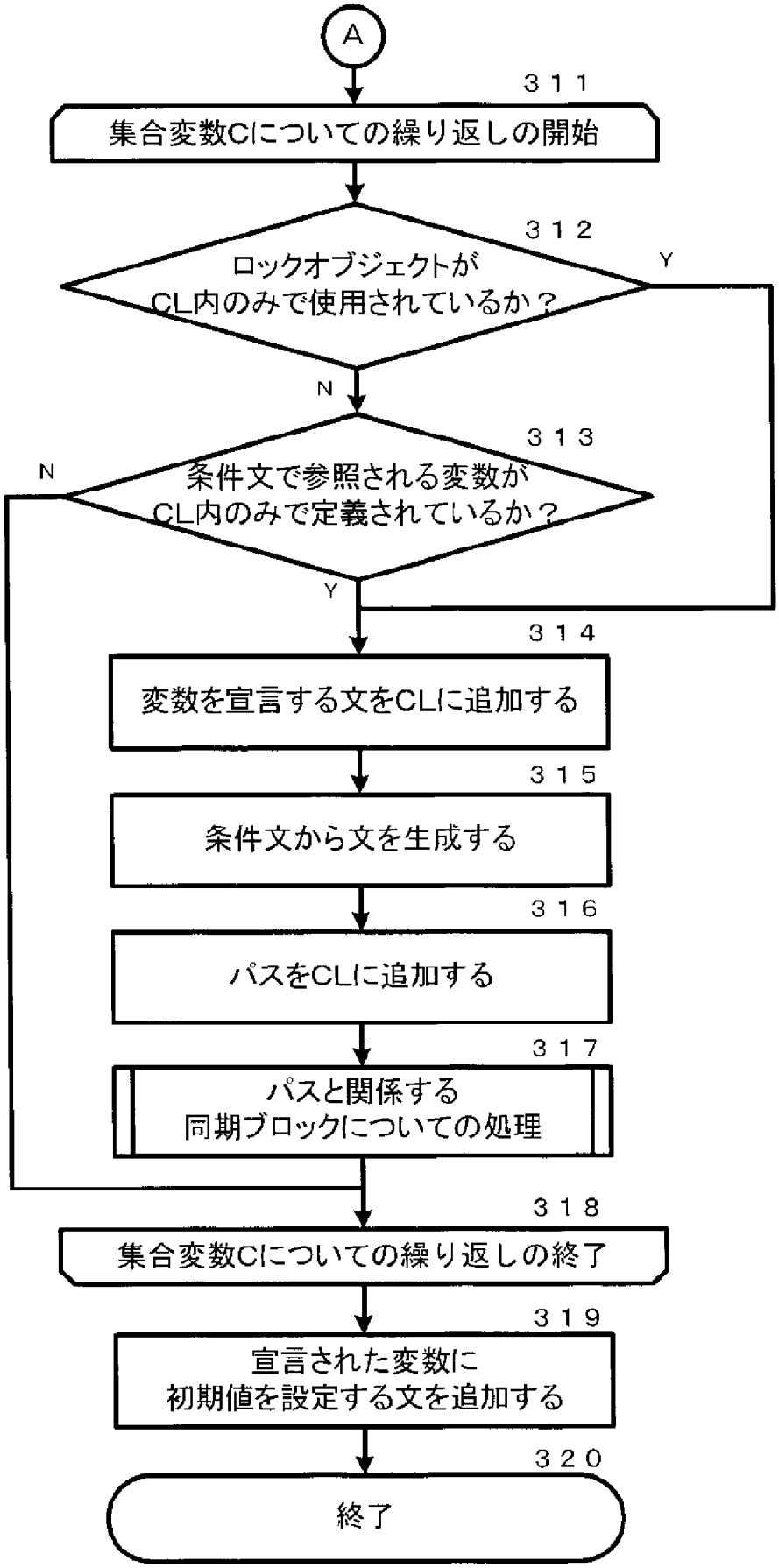
[図2]


200
201A
public class W { Object val; W next;}
public class X {
private W head = null;
private Object lock = new Object();
boolean enqueue(W w) {
synchronized (w) {
204A
205A
if (w.val == null) return false;
210A
206A
208A
synchronized (lock) {
204B
205B
w.next = (head == null) ? w : head;
206B
210B
head = w;
211B
return true;
208B
}
}
}
public W dequeue() {
synchronized (lock) {
204C
if (head != null) {
210C
206C
W w = head;
205D
head = (w.next == w) ? null : w.next;
206D
211C
w.next = w;
210D
211D
return w;
208C
}
return null;
208D
}
}
}
201B
202A
203A
203B
202B
203C
205C
207A
209A
209B
207B
209C
209D

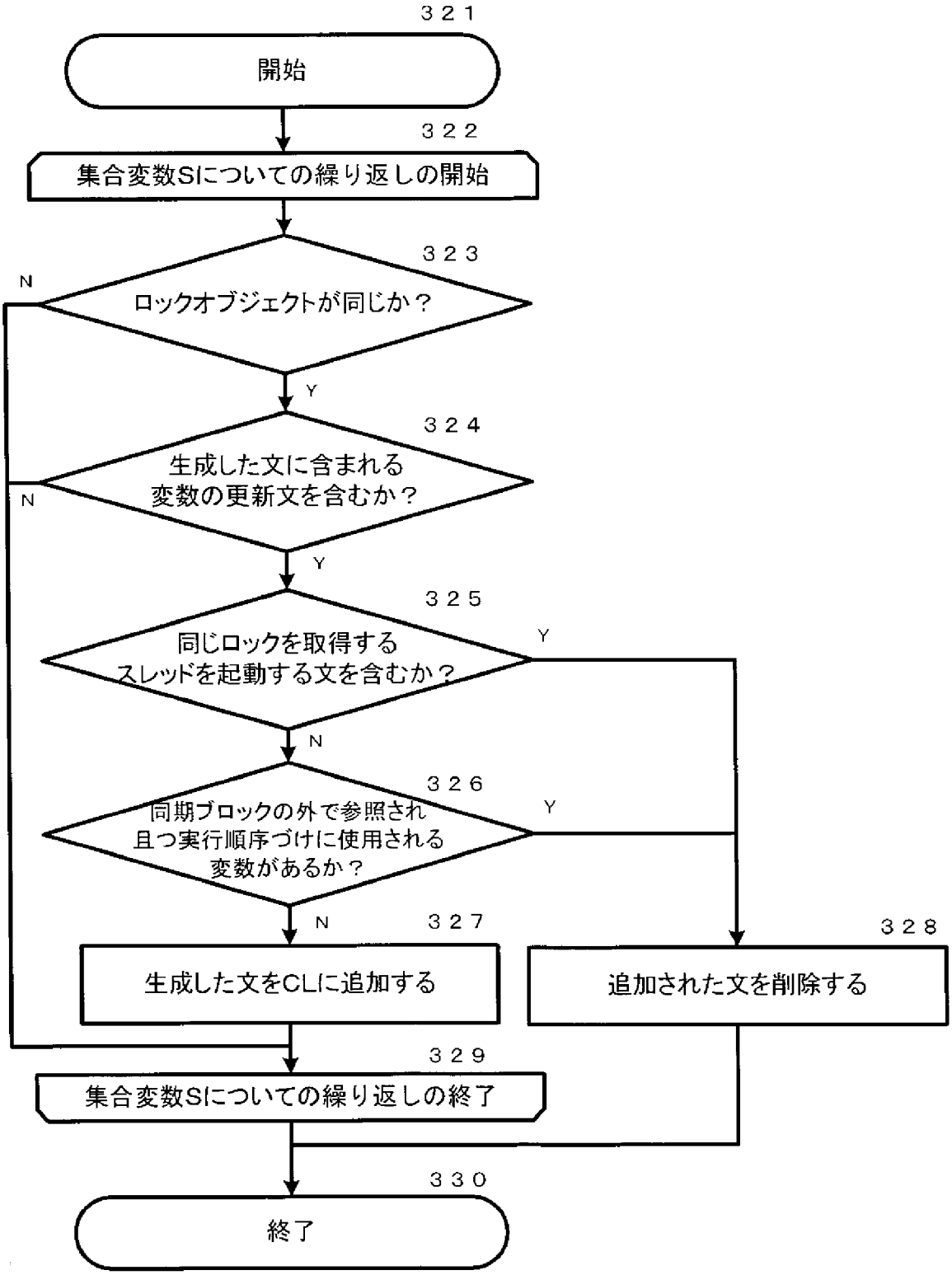
[図3]


３０１
開始
３０２
変数を用意する
３０３
集合変数MIについての繰り返しの開始
３０４
集合変数SIにデータを設定する
３０５
集合変数MIについての繰り返しの終了
３０６
集合変数SIについての繰り返しの開始
３０７
副作用のないパスが
あるか？
N
Y
３０８
パスと
該パスを実行させる条件文とが
要件を満たすか？
N
Y
３０９
パスを保存する
３１０
集合変数SIについての繰り返しの終了
A

[図4]


A
311
集合変数Cについての繰り返しの開始
312
ロックオブジェクトが
CL内のみで使用されているか？
Y
N
313
条件文で参照される変数が
CL内のみで定義されているか？
N
Y
314
変数を宣言する文をCLに追加する
315
条件文から文を生成する
316
パスをCLに追加する
317
パスと関係する
同期ブロックについての処理
318
集合変数Cについての繰り返しの終了
319
宣言された変数に
初期値を設定する文を追加する
320
終了

[図5]


321
開始
322
集合変数Sについての繰り返しの開始
323
ロックオブジェクトが同じか？
N
Y
324
生成した文に含まれる
変数の更新文を含むか？
N
Y
325
同じロックを取得する
スレッドを起動する文を含むか？
Y
N
326
同期ブロックの外で参照され
且つ実行順序づけに使用される
変数があるか？
Y
N
327
生成した文をCLに追加する
328
追加された文を削除する
329
集合変数Sについての繰り返しの終了
330
終了

[図6]

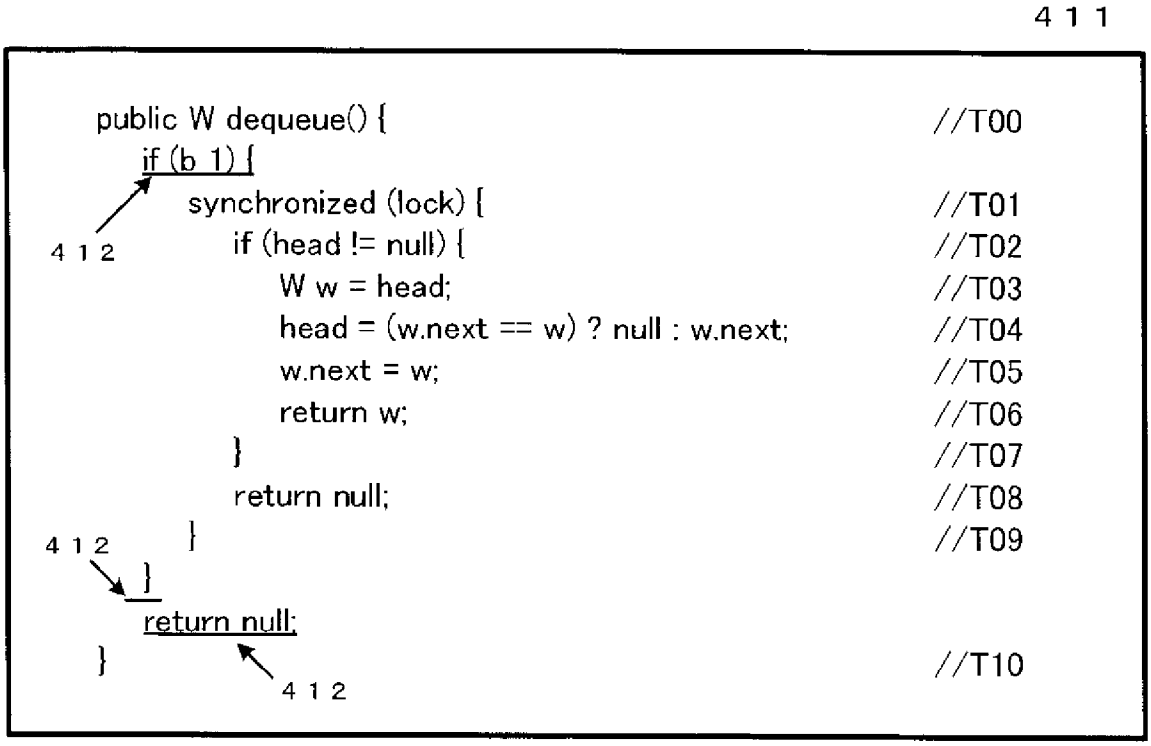
401

```
public class W { Object val; W next; }
public class X {
    private W head = null;                                  //D01
    private Object lock = new Object();                     //D02

    void enqueue(W w, Foo foo) {                            //S00
        synchronized (w) {                                  //S01
            if (w.val == null) return;                      //S02
            synchronized (lock) {                           //S03
                w.next = (head == null) ? W : head;         //S04
                head = w;                                   //S05
                foo.bar(lock);                              //S06
            }                                               //S07
        }                                                   //S08
    }                                                       //S09

    public W dequeue() {                                    //T00
        synchronized (lock) {                               //T01
            if (head != null) {                             //T02
                W w = head;                                 //T03
                head = (w.next == w) ? null : w.next;       //T04
                w.next = w;                                 //T05
                return w;                                   //T06
            }                                               //T07
            return null;                                    //T08
        }                                                   //T09
    }                                                       //T10
}
```

[図7]


411
public W dequeue() { //T00
if (b 1) {
412
synchronized (lock) { //T01
if (head != null) { //T02
W w = head; //T03
head = (w.next == w) ? null : w.next; //T04
w.next = w; //T05
return w; //T06
} //T07
return null; //T08
} //T09
412
}
return null;
412
} //T10

[図8]

421

```
public class W { Object val; W next; }
public class R {
    private W head = null;                                  //D01
    private Object lock = new Object();                     //D02
    private boolean b_1= (head != null);
                                                  422
    void enqueue(W w, Foo foo) {                            //S00
        synchronized (w) {                                  //S01
            if (w.val == null) return;                      //S02
            synchronized (lock) {                           //S03
                w.next = (head == null) ? W : head;         //S04
                head = w;                                   //S05
                foo.bar(lock);                              //S06
                b_1= (head != null);
            }                                               //S07
                                          422
        }                                                   //S08
    }                                                       //S09

    public W dequeue() {                                    //T00
        if (b_1) {
  422       synchronized (lock) {                           //T01
                If (head != null) {                         //T02
                    W w = head;                             //T03
                    head = (w.next == w) ? null : w.next;   //T04
                    w.next = w;                             //T05
                    b_1= (head != null) ;
                    return w;                               //T06
                                      422
                }                                           //T07
                b_1= (head != null) ;
                return null;                                //T08
                                  422
  422       }                                               //T09
        }
        return null;
    }                                                       //T10
                  422
}
```

[図9]

501

```
public class W { Object val; W next; }
public class X {
    private W head = null;                                  //D01
    private Object lock = new Object();                     //D02

    boolean enqueue(W w) {                                  //S00
        synchronized (w) {                                  //S01
            if (w.val == null) return false;                //S02
            synchronized (lock) {                           //S03
                w.next = (head == null) ? W : head;         //S04
                head = w;                                   //S05
                return true;                                //S06
            }                                               //S07
        }                                                   //S08
    }                                                       //S09

    public W dequeue() {                                    //T00
        synchronized (lock) {                               //T01
            if (head != null) {                             //T02
                W w = head;                                 //T03
                head = (w.next == w) ? null : w.next;       //T04
                w.next = w;                                 //T05
                return w;                                   //T06
            }                                               //T07
            return null;                                    //T08
        }                                                   //T09
    }                                                       //T10
}
```

[図10]

511

```
public W dequeue() {                                          //T00
    if (b1) {
        synchronized (lock) {                                 //T01
            if (head != null) {                               //T02
                W w = head;                                   //T03
                head = (w.next == w) ? null : w.next;         //T04
                w.next = w;                                   //T05
                return w;                                     //T06
            }                                                 //T07
            return null;                                      //T08
        }                                                     //T09
    }
    return null;
}                                                             //T10
```

512

512

[図11]

521

```
public class W { Object val; W next; }
public class R {
    private W head = null;                                  //D01
    private Object lock = new Object();                     //D02
    private boolean b_1 = (head != null);
                                                  522
    boolean enqueue(W w) {                                  //S00
        synchronized (w) {                                  //S01
            if (w.val == null) return false;                //S02
            synchronized (lock) {                           //S03
                w.next = (head == null) ? w : head;         //S04
                head = w;                                   //S05
                b_1= (head != null);
                return true;                                //S06
            }                                               //S07
        }                                                   //S08
    }                                                       //S09

    public W dequeue() {                                    //T00
        if (b_1) {
522                                                         //T01
            synchronized (lock) {
                if (head != null) {                         //T02
                    W w = head;                             //T03
                    head = (w.next == w) ? null : w.next;   //T04
                    w.next = w;                             //T05
                    b_1= (head != null) ;
                    return w;                               //T06
                }                                           //T07
                b_1= (head != null) ;
                return null;                                //T08
522         }                                               //T09
522     }
        return null;
    }                                                       //T10
                  522
}
```

[図12]

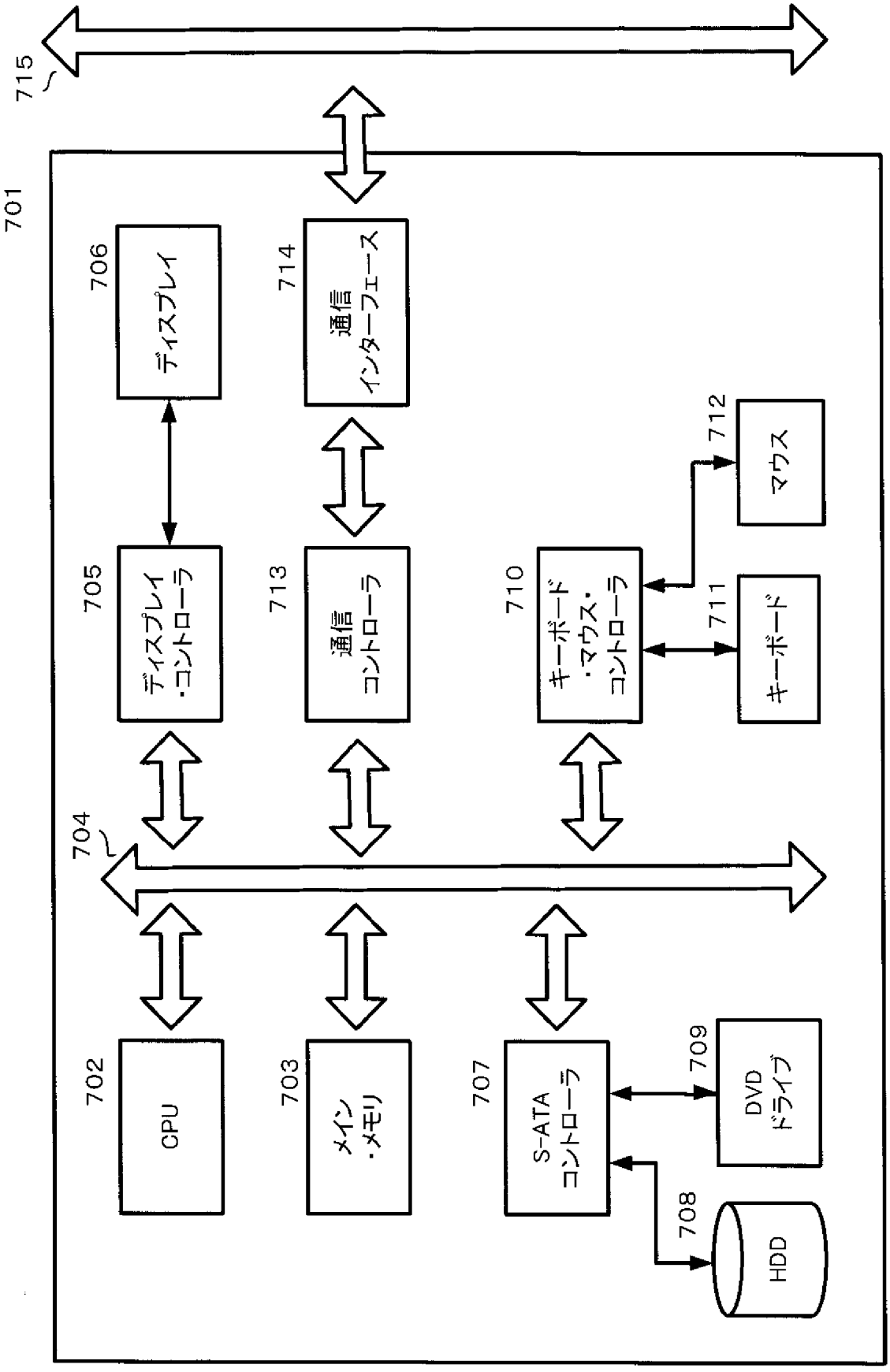
601

```
//c1とc2とはlocal変数
synchronized (obj) {
        if ((x ==0)) {
                if (y ==0) {          ┐
                        x++;          │ 602
                        y--;          ┘
                } else {              ┐
                        c2++;         │ 603
                }                     ┘
        } else {                      ┐
                c1++;                 │ 604
        }                             ┘
}
```

611

```
volatile int b = 0 ←── 612
if (b == 1) { c1++;}
else if (b == 2) { c2++;}  ←── 613
else { // b == 0
        synchronized (obj) {
                if ((x ==0)) {
                        if (y ==0) {
                                x++;
                                y--;
                                b = (x !== 0) ? 1 : (y != 0) ? 2 : 0;
                        } else {                          ←── 614
                                c2++;
                        }
                } else {
                        c1++;
                }
        }
613 ↘
    }
```

[図13]


715
701
706
ディスプレイ
714
通信
インターフェース
712
マウス
705
ディスプレイ
・コントローラ
713
通信
コントローラ
710
キーボード
・マウス・
コントローラ
711
キーボード
704
702
CPU
703
メイン
・メモリ
707
S-ATA
コントローラ
709
DVD
ドライブ
708
HDD

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP2010/057122

A. CLASSIFICATION OF SUBJECT MATTER
G06F9/52(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F9/52

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Jitsuyo Shinan Koho	1922-1996	Jitsuyo Shinan Toroku Koho	1996-2010
Kokai Jitsuyo Shinan Koho	1971-2010	Toroku Jitsuyo Shinan Koho	1994-2010

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	JP 2009-32233 A (Sharp Corp.), 12 February 2009 (12.02.2009), entire text; all drawings (Family: none)	1-21
A	JP 2000-276343 A (Mitsubishi Electric Corp.), 06 October 2000 (06.10.2000), entire text; all drawings (Family: none)	1-21
A	JP 11-288367 A (Hitachi, Ltd.), 19 October 1999 (19.10.1999), entire text; all drawings (Family: none)	1-21

☒ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search 05 July, 2010 (05.07.10)	Date of mailing of the international search report 13 July, 2010 (13.07.10)
Name and mailing address of the ISA/ Japanese Patent Office Facsimile No.	Authorized officer Telephone No.

Form PCT/ISA/210 (second sheet) (July 2009)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP2010/057122

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	JP 4-75136 A (Sanyo Electric Co., Ltd., Mitsubishi Electric Corp., Sharp Corp., Matsushita Electric Industrial Co., Ltd.), 10 March 1992 (10.03.1992), entire text; all drawings (Family: none)	1-21
A	JP 5-66972 A (Nippon Telegraph And Telephone Corp.), 19 March 1993 (19.03.1993), entire text; all drawings (Family: none)	1-21

国際調査報告

国際出願番号　ＰＣＴ／ＪＰ２０１０／０５７１２２

Ａ．　発明の属する分野の分類（国際特許分類（ＩＰＣ））

Int.Cl.　G06F9/52(2006.01)i

Ｂ．　調査を行った分野

調査を行った最小限資料（国際特許分類（ＩＰＣ））

Int.Cl.　G06F9/52

最小限資料以外の資料で調査を行った分野に含まれるもの

日本国実用新案公報	１９２２－１９９６年
日本国公開実用新案公報	１９７１－２０１０年
日本国実用新案登録公報	１９９６－２０１０年
日本国登録実用新案公報	１９９４－２０１０年

国際調査で使用した電子データベース（データベースの名称、調査に使用した用語）

Ｃ．　関連すると認められる文献

引用文献のカテゴリー＊	引用文献名　及び一部の箇所が関連するときは、その関連する箇所の表示	関連する請求項の番号
A	JP 2009-32233 A (シャープ株式会社) 2009.02.12, 全文, 全図 (ファミリーなし)	1-21
A	JP 2000-276343 A (三菱電機株式会社) 2000.10.06, 全文, 全図 (ファミリーなし)	1-21
A	JP 11-288367 A (株式会社日立製作所) 1999.10.19, 全文, 全図 (ファミリーなし)	1-21

☑　Ｃ欄の続きにも文献が列挙されている。

☐　パテントファミリーに関する別紙を参照。

＊　引用文献のカテゴリー
「Ａ」特に関連のある文献ではなく、一般的技術水準を示すもの
「Ｅ」国際出願日前の出願または特許であるが、国際出願日以後に公表されたもの
「Ｌ」優先権主張に疑義を提起する文献又は他の文献の発行日若しくは他の特別な理由を確立するために引用する文献（理由を付す）
「Ｏ」口頭による開示、使用、展示等に言及する文献
「Ｐ」国際出願日前で、かつ優先権の主張の基礎となる出願の日の後に公表された文献
「Ｔ」国際出願日又は優先日後に公表された文献であって出願と矛盾するものではなく、発明の原理又は理論の理解のために引用するもの
「Ｘ」特に関連のある文献であって、当該文献のみで発明の新規性又は進歩性がないと考えられるもの
「Ｙ」特に関連のある文献であって、当該文献と他の１以上の文献との、当業者にとって自明である組合せによって進歩性がないと考えられるもの
「＆」同一パテントファミリー文献

国際調査を完了した日	国際調査報告の発送日
０５．０７．２０１０	１３．０７．２０１０

国際調査機関の名称及びあて先
日本国特許庁（ＩＳＡ／ＪＰ）
郵便番号１００－８９１５
東京都千代田区霞が関三丁目４番３号

特許庁審査官（権限のある職員）　５Ｂ　３５６０
鈴木　修治
電話番号　０３－３５８１－１１０１　内線　３５４５

Ｃ（続き）．　関連すると認められる文献

引用文献のカテゴリー＊	引用文献名　及び一部の箇所が関連するときは、その関連する箇所の表示	関連する請求項の番号
A	JP 4-75136 A（三洋電機株式会社，三菱電機株式会社，シャープ株式会社，松下電器産業株式会社）1992.03.10，全文，全図（ファミリーなし）	1-21
A	JP 5-66972 A（日本電信電話株式会社）1993.03.19，全文，全図（ファミリーなし）	1-21