(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
2015 年 2 月 12 日(12.02.2015)

WIPO | PCT

(10) 国際公開番号
WO 2015/019504 A1

(51) 国際特許分類:
G06F 11/28 (2006.01)

(21) 国際出願番号: PCT/JP2013/071724

(22) 国際出願日: 2013 年 8 月 9 日(09.08.2013)

(25) 国際出願の言語: 日本語

(26) 国際公開の言語: 日本語

(71) 出願人: 富士通株式会社 (FUJITSU LIMITED) [JP/JP]; 〒2118588 神奈川県川崎市中原区上小田中４丁目１番１号 Kanagawa (JP).

(72) 発明者: 清水　智弘 (SHIMIZU, Toshihiro); 〒2118588 神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内 Kanagawa (JP).

(74) 代理人: 服部　毅巖(HATTORI, Kiyoshi); 〒1920082 東京都八王子市東町９番８号　八王子東町センタービル　服部特許事務所 Tokyo (JP).

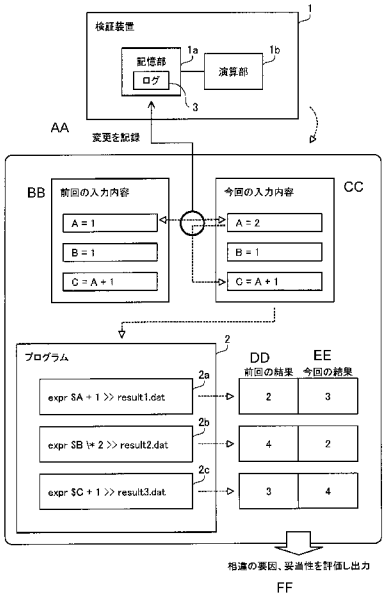
(81) 指定国 (表示のない限り、全ての種類の国内保護が可能): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, 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, RW, SD, SL, SZ, TZ, UG, ZM, ZW), ユーラシア (AM, AZ, BY, KG, KZ, RU, TJ, TM), ヨーロッパ (AL, 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, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

[続葉有]

(54) **Title**: VERIFICATION METHOD, VERIFICATION DEVICE, AND VERIFICATION PROGRAM

(54) 発明の名称 : 検証方法、検証装置および検証プログラム



1 Verification device
1a Storage unit
1b Computation unit
2 Program
3 Log
AA Record change
BB Previous input
CC Current input
DD Previous result
EE Current result
FF Evaluate and output difference factor and validity

(57) **Abstract**: An objective of the present invention is to allow assisting appropriately comprehending a factor whereupon execution results differ. When executing a program (2), a computation unit (1b) acquires a log (3) which includes information which identifies variables (A, C) which have had a change in the input content thereof with respect to a past execution and information which denotes a correspondence between program elements (2a, 2b, 2c) and each execution result which is an output of each of the program elements. If any of the results of the execution of the program (2) differ from the result of the past execution thereof, the computation unit (1b) queries the log (3) and searches for the program element (2a) corresponding to the said result of the execution and the variable (A) which is used in the program element (2a). On the basis of a change circumstance in the input content with respect to the variable (A), the computation unit (1b) evaluates the factor whereupon the result of the execution differs and the validity of the difference. The computation unit (1b) outputs information indicating the site whereupon the results of the execution differ, the factor of the difference, and the validity of the difference.

(57) 要約:　実行結果が相違する要因の適切な把握を支援できる。　演算部（１ｂ）は、プログラム（２）の実行時に、過去に実行したときに対して入力内容が変更された変数（Ａ，Ｃ）を識別する情報、および、プログラム要素（２ａ，２ｂ，２ｃ）と各プログラム要素の出力である各実行結果との対応を示す情報を含むログ（３）を取得する。演算部（１ｂ）は、プログラム（２）の何れかの実行結果が過去の実行結果と相違するとき、ログ（３）を参照して、当該実行結果に対応するプログラム要素（２ａ）およびプログラム要素（２ａ）に用いられる変数（Ａ）を検索する。演算部（１ｂ）は、変数（Ａ）に対する入力内容の変更状況に基づいて、当該実行結果が相違する要因と当該相違の妥当性とを評価する。演算部（１ｂ）は、実行結果の相違する箇所と相違の要因と相違の妥当性とを示す情報を出力する。

WO 2015/019504 A1

添付公開書類:

— 国際調査報告（条約第 21 条(3)）

明　細　書

発明の名称 ： 検証方法、検証装置および検証プログラム

技術分野

[0001] 本発明は検証方法、検証装置および検証プログラムに関する。

背景技術

[0002] 現在、種々のプログラムが利用されている。ユーザは、プログラムにより、一連の手順をコンピュータに実行させることで、所定の機能をコンピュータに発揮させることができる。例えば、業務用の機能を提供するソフトウェアのプログラムや、他のソフトウェアの動作を支援するソフトウェアのプログラム等が利用されている。

[0003] ここで、プログラムの実行結果がユーザの想定する結果になることを確かめたいことがある。想定外の結果になってしまうと、システムの仕様や制約に則った運用に支障を来すおそれがあるからである。そこで、プログラムを検証することが考えられている。

[0004] 例えば、データの更新を行う更新関数がオリジナルのデータを更新するときに更新関数の引数に順次与えられた値の履歴を記録しておき、更新関数の検証に用いる提案がある。この提案では、検証時に、記録された複数の値の少なくとも１つを当該複数の値に追加した数値群の各値を更新関数に順次与える。検証時のデータの最終的な更新結果が、更新後のオリジナルのデータと一致すれば、当該更新関数に可換性（データに対する更新の適用順序を変えても結果が変わらないこと）および冪等性（データに対して同じ更新を複数回適用しても、一度のみ適用した場合とその結果が変わらないこと）があると判断する。

[0005] なお、ネットワーク内の構成要素（例えば、パーソナルコンピュータやサーバ）に設定する設定パラメータおよび型定義を、設定パラメータおよび型定義のマスタとなる組と照合することで、設定内容の妥当性を検証する提案もある。

先行技術文献

特許文献

[0006] 特許文献1：特開２０１０－１２３０３０号公報

特許文献2：特表２００５－５１２１９６号公報

発明の概要

発明が解決しようとする課題

[0007] 変数に応じた処理を示す複数のプログラム要素（例えば、命令や関数等）を含むプログラムを検証することが考えられる。例えば、複数のコンピュータ上への仮想マシン群やソフトウェア群等の配備を支援する配備プログラムが利用されることがある。配備プログラムは、仮想マシン等を動作させるための各種の設定を行う複数のプログラム要素を含み得る。このようなプログラムでは、複数のプログラム要素により複数の実行結果が生成され得る。この場合、各実行結果の妥当性をどのようにして検証するかが問題となる。

[0008] 例えば、システムの部分的な仕様変更に応じて、配備プログラムにおける一部の変数の設定値が変更されたり、一部のプログラム要素自体のコードが変更されたりする。このため、配備プログラムが異なるタイミングで実行されると、一部のプログラム要素で両タイミングの実行結果が相違することがある。ところが、この場合、実行結果の相違が、両タイミング間で発生した一部の変数の設定値の変更が要因なのか、何れかのプログラム要素自体の処理が要因なのか、実行結果のみからでは適切に把握することが難しい。

[0009] １つの側面では、本発明は、実行結果が相違する要因の適切な把握を支援する検証方法、検証装置および検証プログラムを提供することを目的とする。

課題を解決するための手段

[0010] １つの態様では、変数に応じた処理を示す複数のプログラム要素を含むプログラムの検証方法が提供される。この検証方法では、コンピュータが、プログラムの実行時に、過去に当該プログラムを実行したときに対して入力内

容が変更された変数を識別する情報、および、プログラム要素と当該プログラム要素が実行されることで出力される実行結果との対応を示す情報を含むログを取得し、プログラムの何れかの実行結果が過去の実行結果と相違するとき、ログを参照して、当該実行結果に対応するプログラム要素および当該プログラム要素に用いられる変数を検索し、当該変数に対する入力内容の変更状況に基づいて、実行結果が相違する要因と当該相違の妥当性とを評価し、実行結果の相違する箇所と相違の要因と相違の妥当性とを示す情報を出力する。

[0011] また、１つの態様では、変数に応じた処理を示す複数のプログラム要素を含むプログラムの検証に用いられる検証装置が提供される。この検証装置は、記憶部と演算部とを有する。記憶部は、プログラムの実行状況を示すログを記憶する。演算部は、プログラムの実行時に、過去に当該プログラムを実行したときに対して入力内容が変更された変数を識別する情報、および、プログラム要素と当該プログラム要素が実行されることで出力される実行結果との対応を示す情報を含むログを取得し、プログラムの何れかの実行結果が過去の実行結果と相違するとき、ログを参照して、当該実行結果に対応するプログラム要素および当該プログラム要素に用いられる変数を検索し、当該変数に対する入力内容の変更状況に基づいて、実行結果が相違する要因と当該相違の妥当性とを評価し、実行結果の相違する箇所と相違の要因と相違の妥当性とを示す情報を出力する。

[0012] また、１つの態様では、コンピュータによって実行される検証プログラムであって、変数に応じた処理を示す複数のプログラム要素を含むプログラムの検証に用いられる検証プログラムが提供される。この検証プログラムは、コンピュータに、プログラムの実行時に、過去に当該プログラムを実行したときに対して入力内容が変更された変数を識別する情報、および、プログラム要素と当該プログラム要素が実行されることで出力される実行結果との対応を示す情報を含むログを取得し、プログラムの何れかの実行結果が過去の実行結果と相違するとき、ログを参照して、当該実行結果に対応するプログ

ラム要素および当該プログラム要素に用いられる変数を検索し、当該変数に対する入力内容の変更状況に基づいて、実行結果が相違する要因と当該相違の妥当性とを評価し、実行結果の相違する箇所と相違の要因と相違の妥当性とを示す情報を出力する、処理を実行させる。

発明の効果

[0013] １つの側面では、実行結果が相違する要因の適切な把握を支援できる。

本発明の上記および他の目的、特徴および利点は本発明の例として好ましい実施の形態を表す添付の図面と関連した以下の説明により明らかになるであろう。

図面の簡単な説明

[0014] [図1]第１の実施の形態の検証装置を示す図である。

[図2]第２の実施の形態の情報処理システムを示す図である。

[図3]配備サーバのハードウェア例を示す図である。

[図4]配備サーバの機能例を示す図である。

[図5]記憶部に格納されるデータの例を示す図である。

[図6]配備プログラムに含まれるブロックの例を示す図である。

[図7]配備プログラムの例を示す図である。

[図8]変換後配備プログラムの例を示す図である。

[図9]自由変数テーブルの例を示す図である。

[図10]システムコールログの例を示す図である。

[図11]システムコールログの例（続き）を示す図である。

[図12]ログテーブルの例を示す図である。

[図13]検証処理の全体を示すフローチャートである。

[図14]配備プログラムの変換例を示すフローチャートである。

[図15]配備プログラムの冪等性の検証例を示すフローチャートである。

[図16]ログに基づく変数の分析例を示すフローチャートである。

[図17]検証結果の出力例（その１）を示す図である。

[図18]検証結果の出力例（その２）を示す図である。

[図19]検証結果の出力例（その３）を示す図である。

[図20]配備プログラムの他の例を示す図である。

[図21]ＢＮＦ記法で記したプログラムの例を示す図である。

[図22]プログラム変換用の関数の記述例（その１）を示す図である。

[図23]プログラム変換用の関数の記述例（その２）を示す図である。

[図24]プログラム変換用の関数の記述例（その３）を示す図である。

[図25]配備プログラムの他の変換例を示す図である。

[図26]ログの出力フォーマットの例を示す図である。

発明を実施するための形態

[0015] 以下、本実施の形態を図面を参照して説明する。

［第１の実施の形態］

図１は、第１の実施の形態の検証装置を示す図である。検証装置１は、プログラム２の検証に用いられる。プログラム２は、プログラム要素２ａ，２ｂ，２ｃを含む。ここで、プログラム要素とは、所定の変数に応じた処理（の実行）を示す命令や関数（あるいは命令の集合）等である。プログラム要素２ａは変数Ａに応じた処理を示す。プログラム要素２ｂは変数Ｂに応じた処理を示す。プログラム要素２ｃは変数Ｃに応じた処理を示す。

[0016] 検証装置１は、記憶部１ａおよび演算部１ｂを有する。記憶部１ａは、ＲＡＭ（Random Access Memory）等の揮発性記憶装置でもよいし、ＨＤＤ（Hard Disk Drive）やフラッシュメモリ等の不揮発性記憶装置でもよい。演算部１ｂは、ＣＰＵ（Central Processing Unit）、ＤＳＰ（Digital Signal Processor）、ＡＳＩＣ（Application Specific Integrated Circuit）、ＦＰＧＡ（Field Programmable Gate Array）等を含み得る。演算部１ｂはプログラムを実行するプロセッサであってもよい。ここでいう「プロセッサ」には、複数のプロセッサの集合（マルチプロセッサ）も含まれ得る。

[0017] 記憶部１ａは、プログラム２の実行状況を示すログ３を記憶する。プログラム２は、検証装置１で実行されるものでもよいし、他の情報処理装置で実行されるものでもよい。検証装置１で実行される場合、演算部１ｂ自身がロ

グ３を記録できる。他の情報処理装置で実行される場合、演算部１ｂは、他の情報処理装置からネットワークを介して、ログ３の内容を取得し、記録できる。このように、演算部１ｂは検証装置１および他の情報処理装置の何れでプログラム２が実行されても、ログ３を取得し、記憶部１ａに格納できる。

[0018] 演算部１ｂは、プログラム２の実行時に、過去にプログラム２を実行したときに対して入力内容が変更された変数を識別する情報を含むログ３を取得する。変数の入力内容の情報はプログラム２に含まれてもよいし、プログラム２とは別個のデータで用意されてもよい。

[0019] 例えば、演算部１ｂは、変数の入力内容の情報がプログラム２自体に含まれていれば、過去にプログラム２が実行されたときのプログラム２の内容と、今回実行されるプログラム２の内容とを対比することで、入力内容の変更された変数を検出し得る。あるいは、変数の入力内容の情報がプログラム２とは別個のデータとして用意されていれば、演算部１ｂは、当該データの過去の内容と今回の内容とを対比することで、入力内容の変更された変数を検出し得る。両方の方法を用いて、入力内容の変更された変数を検出してもよい。なお、過去に実行されたプログラム２や入力内容の情報は記憶部１ａに記憶される。

[0020] 例えば、過去の実行時の一例として、今回の実行時の直前の実行時（前回の実行時）が考えられる。例えば、この場合、演算部１ｂは、前回の実行時と今回の実行時とで入力内容が変更された変数を識別する情報をログ３に記録する。例えば、前回の各変数の入力内容は次の通りであるとする。Ａ＝１。Ｂ＝１。Ｃ＝Ａ＋１。これに対し、今回の各変数の入力内容は次の通りであるとする。Ａ＝２。Ｂ＝１。Ｃ＝Ａ＋１。

[0021] この場合、変数Ａの入力内容は、前回と今回とで変更されている。変数Ｂの入力内容は、前回と今回とで変更されていない。変数Ｃの定義自体は前回と今回とで変更されていないが、変数Ｃは変数Ａに依存する。変数Ａの入力内容は前回と今回とで変更されているから、変数Ｃの入力内容も前回と今回

とで変更されている。よって、この場合、演算部１ｂは、変数Ａ，Ｃが変更されていることをログ３に記録する。

[0022] また、演算部１ｂは、プログラム２の実行時に、プログラム要素と当該プログラム要素が実行されることで出力される実行結果との対応を示す情報をログ３に記録する。例えば、プログラム２の実行に伴って、プログラム要素２ａでは“＄Ａ＋１”（＄Ａは変数Ａに代入されている値を示す。以下、同様）を計算し、その実行結果として“ｒｅｓｕｌｔ１．ｄａｔ”を出力する。プログラム要素２ｂでは“＄Ｂ＼＊２”を計算し（“＼＊”は乗算を示す）、その実行結果として“ｒｅｓｕｌｔ２．ｄａｔ”を出力する。プログラム要素２ｃでは“＄Ｃ＋１”を計算し、その実行結果として“ｒｅｓｕｌｔ３．ｄａｔ”を出力する。

[0023] この場合、演算部１ｂは、プログラム要素２ａと実行結果“ｒｅｓｕｌｔ１．ｄａｔ”との対応をログ３に記録する。プログラム要素２ｂと実行結果“ｒｅｓｕｌｔ２．ｄａｔ”との対応をログ３に記録する。プログラム要素２ｃと実行結果“ｒｅｓｕｌｔ３．ｄａｔ”との対応をログ３に記録する。演算部１ｂは、プログラム２の各実行結果を、過去の分についても記憶部１ａに格納しておく。

[0024] 演算部１ｂは、プログラム２の何れかの実行結果が過去の実行結果と相違するとき、記憶部１ａに記憶されたログ３を参照して、当該実行結果に対応するプログラム要素および当該プログラム要素に用いられる変数を検索する。

[0025] 例えば、実行結果“ｒｅｓｕｌｔ１．ｄａｔ”の内容は前回（“２”）と今回（“３”）とで相違する。よって、演算部１ｂは、ログ３を参照して、当該実行結果に対応するプログラム要素２ａとプログラム要素２ａに用いられる変数Ａとを検索する。また、実行結果“ｒｅｓｕｌｔ２．ｄａｔ”の内容は前回（“４”）と今回（“２”）とで相違する。よって、演算部１ｂは、ログ３を参照して、当該実行結果に対応するプログラム要素２ｂとプログラム要素２ｂに用いられる変数Ｂとを検索する。また、実行結果“ｒｅｓｕ

ｌｔ３．ｄａｔ”の内容は前回（“３”）と今回（“４”）とで相違する。よって、演算部１ｂは、ログ３を参照して、当該実行結果に対応するプログラム要素２ｃとプログラム要素２ｃに用いられる変数Ｃとを検索する。演算部１ｂは、各プログラム要素に用いられる各変数をプログラム２の記述から検索してもよい。または、プログラム要素と変数との一覧を予め用意しておき、当該一覧からプログラム要素に用いられる変数を検索してもよい。

[0026] 演算部１ｂは、記憶部１ａに記憶されたログ３を参照し、検索された変数に対する入力内容の変更状況に基づいて、実行結果が相違する要因と当該相違の妥当性とを評価し、評価結果と実行結果が異なる箇所とを示す情報を出力する。ここで、結果の妥当性の基準は、プログラム２に対して期待される性質による。例えば、プログラム２に冪等性（同じ入力に対しては、何回実行しても同じ結果が得られる性質）を求めるなら、同じ入力に対して同じ結果が得られるときに、その結果は妥当である。一方、同じ入力に対して同じ結果が得られなければ、その結果は妥当でない。プログラム２に冪等性を求める場合を考えると、演算部１ｂの出力は次のようになる。

[0027] 例えば、ログ３には、前述のように変数Ａの入力内容に変更があったことが記録されている。この場合、プログラム要素２ａの変数Ａの入力内容の変更が、前回と今回との実行結果の相違の要因であると考えられる。したがって、演算部１ｂは、（ａ１）プログラム要素２ａの変数Ａに対する入力内容の変更があったことを相違の要因として出力する。また、（ａ２）プログラム要素２ａに対する入力が変わっているから実行結果の相違は妥当である旨を出力する。更に、（ａ３）相違する実行結果“ｒｅｓｕｌｔ１．ｄａｔ”の情報を出力する（当該データ内の相違箇所を出力してもよい。以下、同様）。

[0028] また、ログ３には、変数Ｂの入力内容に変更があったことが記録されていない。この場合、プログラム要素２ｂに対する入力が変更されていないにも関わらず、その実行結果が前回と今回とで相違していることになる。したがって、プログラム要素２ｂは、冪等性がないと考えられる。よって、演算部

１ｂは、（ｂ１）プログラム要素２ｂを相違の要因として出力する。また、（ｂ２）プログラム要素２ｂは冪等性がないから実行結果の相違は妥当ではない旨を出力する。更に、（ｂ３）相違する実行結果“ｒｅｓｕｌｔ２．ｄａｔ”の情報を出力する。

[0029] また、ログ３には、前述のように変数Ｃの入力内容に変更があったことが記録されている。この場合、プログラム要素２ｃの変数Ｃの入力内容の変更が、前回と今回との実行結果の相違の要因であると考えられる。したがって、演算部１ｂは、（ｃ１）プログラム要素２ｃの変数Ｃに対する入力内容の変更があったことを相違の要因として出力する。また、（ｃ２）プログラム要素２ｃに対する入力が変わっているから実行結果の相違は妥当である旨を出力する。更に、（ｃ３）相違する実行結果“ｒｅｓｕｌｔ３．ｄａｔ”の情報を出力する。

[0030] 検証装置１によれば、演算部１ｂにより、プログラム２の実行時に、過去にプログラム２を実行したときに対して入力内容が変更された変数Ａ，Ｃを識別する情報、および、プログラム要素２ａ，２ｂ，２ｃとプログラム要素２ａ，２ｂ，２ｃが実行されることで出力される各実行結果との対応を示す情報がログ３に記録される。演算部１ｂにより、プログラム２の何れかの実行結果が過去の実行結果と相違するとき、ログ３が参照されて、当該実行結果に対応するプログラム要素および当該プログラム要素に用いられる変数が検索される。演算部１ｂにより、ログ３が参照されて、検索された当該変数に対する入力内容の変更状況に基づいて、実行結果が相違する要因と当該相違の妥当性とが評価され、評価結果と実行結果の異なる箇所とを示す情報が出力される。

[0031] これにより、実行結果が相違する要因の適切な把握を支援できる。例えば、ユーザは、上記出力（ａ１）～（ａ３）および（ｃ１）～（ｃ３）を参照することで、プログラム要素２ａ，２ｃの実行結果に相違がみられるものの、変数Ａ，Ｃに対する入力内容が変更されたことに起因しており妥当であると適切かつ迅速に判断できる。

[0032] ここで、単に、プログラム要素２ａ，２ｃ自体の内容が過去の実行時と書き換わっているかを確認すること（静的解析）も考えられる。しかし、プログラム要素２ａ，２ｃ自体の内容が過去の実行時と同一であっても、実行結果が相違することがあり、静的解析のみでは当該相違の要因や妥当性の判断を行うのは難しい。これに対し、検証装置１を用いることで、プログラム要素２ａ，２ｃ自体の内容が書き換わっていない場合でも、上記のように実行結果の相違の要因と妥当性の判断を適切に行える。

[0033] また、ユーザは、上記の出力（ｂ１）～（ｂ３）を参照することで、プログラム要素２ｂの実行結果の相違は妥当でないことを適切かつ迅速に判断できる。具体的には、当該相違は入力の変更によるものではなく、プログラム要素２ｂが求められる性質（例えば、冪等性）をもっていないことに起因していることを容易に把握できる。このため、ユーザは、プログラム要素２ｂの見直しを迅速に開始できる。例えば、プログラム要素２ｂ自体の処理に問題があったのか、プログラム要素２ｂが過去の記述から書き換えられたのか等の判断作業を迅速に行える。

[0034] このように、過去の実行結果と相違する実行結果に対して、その妥当性と要因とを出力することで、ユーザによる要因分析の作業やプログラムの修正作業の省力化を図れる。その結果、プログラム開発における作業コストの軽減にも寄与し得る。

[0035] 演算部１ｂは、プログラム２を実行する検証装置１または他の情報処理装置がログ３を生成するように、プログラム２を変換してもよい。あるいは、そのような変換後のプログラムがプログラム２であると考えてもよい。そうすれば、ログ３を出力するためのコードのプログラム２への挿入をユーザに強いずに済み、効率的な検証作業を支援できる。

[0036] ここで、変数に対する入力内容が変更されているか否かの判断には種々の方法が考えられる。上記のように、（１）今回と前回との入力内容を直接対比して変更の有無を判断してもよい。（２）変数Ｚのように、第１の変数の代入文の中に含まれる第２の変数を検出して、当該第２の変数に対する入力

内容の変更の有無から、第１の変数に対する入力内容の変更の有無を判断してもよい。

[0037] また、（３）１つの変数に対する代入文が条件式の複数の節（ｔｈｅｎ節、ｅｌｓｅ節等）それぞれに記述されている場合には、当該変数の入力内容に変更があるとみなしてログ３に出力するようにしてもよい。条件式の実行結果に応じて当該変数の代入値が変更される可能性が高いからである。（１）～（３）のような方法を採れば、各変数の入力内容の変更の有無を適切に検出して、ログ３に出力するようプログラム２を変換できる。

[0038] このとき、演算部１ｂは、各プログラム要素の記述を過去に実行したときと対比することで、各プログラム要素自体のコード変更の有無を判断して、ログ３に出力するようにしてもよい。このようにすれば、プログラム要素自体のコード変更を加味して、結果データの相違の妥当性を検証できる。例えば、前回と今回とで、プログラム要素２ｂにコード変更があるなら、前回と今回とで結果が相違しても当該相違を妥当と評価することが考えられる。これにより、ユーザの作業負担を一層軽減でき、効率的な検証作業を支援できる。

[0039] プログラム２の具体例として、仮想マシンやソフトウェア等を情報処理装置に配備する配備プログラムが挙げられる。以下では、配備プログラムの検証を行う場合を想定して更に詳細に説明する。ただし、他の種類のプログラムに対する適用を妨げるものではない。

[0040] ［第２の実施の形態］

図２は、第２の実施の形態の情報処理システムを示す図である。第２の実施の形態の情報処理システムは、配備サーバ１００および配備先サーバ２００，３００，４００を含む。配備サーバ１００および配備先サーバ２００，３００，４００は、ネットワーク１０に接続されている。ネットワーク１０は、ＬＡＮ（Local Area Network）でもよいし、ＷＡＮ（Wide Area Network）やインターネット等の広域ネットワークでもよい。

[0041] 配備サーバ１００は、配備先サーバ２００，３００，４００に対する仮想

マシンや所定のソフトウェアの配備（以下、単に仮想マシンの配備という）を制御するサーバコンピュータである。配備サーバ１００は、配備プログラムを用いて仮想マシンの配備を行う。配備プログラムの例としては、ｃｈｅｆ、ｐｕｐｐｅｔおよびｒｕｎｄｅｃｋ等が考えられる。

[0042] 例えば、ユーザは、情報処理システムの仕様に応じて、配備プログラムを作成し、配備サーバ１００に実行させる。配備プログラムには、ユーザにより、ディスク、ネットワークおよびユーザアカウント等の設定を行うための一連の命令（スクリプト）を示すコードが記述される。配備プログラムを配備用のスクリプトと呼んでもよい。配備サーバ１００は、配備プログラムに記述された内容に従って、ディスク、ネットワークおよびユーザアカウント等の設定を配備先サーバ２００，３００，４００に実行させる。

[0043] 配備先サーバ２００，３００，４００は、複数の仮想マシンを実行可能なサーバコンピュータである。例えば、配備先サーバ２００，３００，４００は、ハイパーバイザと呼ばれるソフトウェアを実行する。ハイパーバイザは、配備先サーバ２００，３００，４００のＣＰＵやＲＡＭ等のリソースを仮想マシンに割り当てる。例えば、配備先サーバ２００では、割り当てられたリソースを用いて仮想マシンＶ１，Ｖ２が実行される。配備先サーバ２００，３００，４００は、配備サーバ１００の指示に従って、仮想マシンの動作環境を設定する。例えば、配備先サーバ２００，３００，４００には、配備サーバ１００と連携する配備用のクライアントアプリケーションが実行される。

[0044] ここで、配備プログラムでは、設定内容が同じである複数の仮想マシンの配備手順が１つの配備プログラム中に記述されることが多い。また、多数の仮想マシンが１つの配備プログラムによって配備先サーバ２００，３００，４００に配備され得る。このような情報処理システムでは、何れかの仮想マシンで配備の失敗が起こったとしても、何れの仮想マシンの何れの設定で失敗が起こったかを把握して、設定内容を個別に訂正するのは作業に時間がかかり非効率的である。そこで、そのような場合は、全ての仮想マシンの配備

を配備プログラムにより再実行する。設定内容を個別に検査して訂正するよりも全ての仮想マシンの配備をやり直した方が効率的だからである。

[0045] 例えば、複数の仮想マシンの配備を配備プログラムにより行う場合、配備手順で一部失敗があった仮想マシンが存在しても、当該仮想マシンを指定することなく、全ての仮想マシンの再配備を行う。また、例えば、当該仮想マシンにアカウント追加を行おうとする場合、アカウント設定のプログラムを別個に用意せずに、配備プログラムに入力するパラメータを変更し、配備プログラム全体を再実行する。

[0046] このような作業を行うに当たり、配備プログラムには冪等性が求められる。具体的には、配備の手順に失敗していない同一の仮想マシンの再配備を行っても結果が変わらないことが望まれる。また、例えば、変更された設定（例えば、アカウント設定）とは関係ない設定（例えば、ネットワーク設定）に対して、再配備後も設定結果が変わらないことが望まれる。

[0047] そこで、第２の実施の形態では配備プログラムの冪等性の検証を支援する機能を提供する。当該検証は、例えば、１台の配備先サーバ（例えば、配備先サーバ２００）を用いて行える。以下では、配備先サーバ２００を用いて検証を行うことを想定する。

[0048] 図３は、配備サーバのハードウェア例を示す図である。配備サーバ１００は、プロセッサ１０１、ＲＡＭ（Random Access Memory）１０２、ＨＤＤ（Hard Disk Drive）１０３、通信部１０４、画像信号処理部１０５、入力信号処理部１０６、ディスクドライブ１０７および機器接続部１０８を有する。各ユニットが配備サーバ１００のバスに接続されている。配備先サーバ２００，３００，４００も配備サーバ１００と同様のユニットを用いて実現できる。

[0049] プロセッサ１０１は、配備サーバ１００の情報処理を制御する。プロセッサ１０１は、マルチプロセッサであってもよい。プロセッサ１０１は、例えばＣＰＵ、ＭＰＵ（Micro Processing Unit）、ＤＳＰ、ＡＳＩＣ、ＦＰＧＡまたはＰＬＤ（Programmable Logic Device）等である。プロセッサ１０１は

、ＣＰＵ、ＭＰＵ、ＤＳＰ、ＡＳＩＣ、ＦＰＧＡ、ＰＬＤのうちの２以上の要素の組み合わせであってもよい。

[0050] ＲＡＭ１０２は、配備サーバ１００の主記憶装置である。ＲＡＭ１０２は、プロセッサ１０１に実行させるＯＳ（Operating System）のプログラムやアプリケーションプログラムの少なくとも一部を一時的に記憶する。また、ＲＡＭ１０２は、プロセッサ１０１による処理に用いる各種データを記憶する。

[0051] ＨＤＤ１０３は、配備サーバ１００の補助記憶装置である。ＨＤＤ１０３は、内蔵した磁気ディスクに対して、磁気的にデータの書き込みおよび読み出しを行う。ＨＤＤ１０３には、ＯＳのプログラム、アプリケーションプログラム、および各種データが格納される。配備サーバ１００は、フラッシュメモリやＳＳＤ（Solid State Drive）等の他の種類の補助記憶装置を備えてもよく、複数の補助記憶装置を備えてもよい。

[0052] 通信部１０４は、ネットワーク１０を介して他のコンピュータと通信を行えるインタフェースである。通信部１０４は、有線インタフェースでもよいし、無線インタフェースでもよい。

[0053] 画像信号処理部１０５は、プロセッサ１０１からの命令に従って、配備サーバ１００に接続されたディスプレイ１１に画像を出力する。ディスプレイ１１としては、ＣＲＴ（Cathode Ray Tube）ディスプレイや液晶ディスプレイ等を用いることができる。

[0054] 入力信号処理部１０６は、配備サーバ１００に接続された入力デバイス１２から入力信号を取得し、プロセッサ１０１に出力する。入力デバイス１２としては、例えば、マウスやタッチパネル等のポインティングデバイス、キーボード等を用いることができる。

[0055] ディスクドライブ１０７は、レーザ光等を利用して、光ディスク１３に記録されたプログラムやデータを読み取る駆動装置である。光ディスク１３として、例えば、ＤＶＤ（Digital Versatile Disc）、ＤＶＤ－ＲＡＭ、ＣＤ－ＲＯＭ（Compact Disc Read Only Memory）、ＣＤ－Ｒ（Recordable）／Ｒ

W（ReWritable）等を使用できる。ディスクドライブ１０７は、例えば、プロセッサ１０１からの命令に従って、光ディスク１３から読み取ったプログラムやデータをＲＡＭ１０２またはＨＤＤ１０３に格納する。

[0056] 機器接続部１０８は、配備サーバ１００に周辺機器を接続するための通信インタフェースである。例えば、機器接続部１０８にはメモリ装置１４やリーダライタ装置１５を接続できる。メモリ装置１４は、機器接続部１０８との通信機能を搭載した記録媒体である。リーダライタ装置１５は、メモリカード１６へのデータの書き込み、またはメモリカード１６からのデータの読み出しを行う装置である。メモリカード１６は、カード型の記録媒体である。機器接続部１０８は、例えば、プロセッサ１０１からの命令に従って、メモリ装置１４またはメモリカード１６から読み取ったプログラムやデータをＲＡＭ１０２またはＨＤＤ１０３に格納する。

[0057] 図４は、配備サーバの機能例を示す図である。配備サーバ１００は、記憶部１１０、プログラム変換部１２０、配備実行部１３０および検証部１４０を有する。記憶部１１０は、ＲＡＭ１０２またはＨＤＤ１０３に確保した記憶領域として実現できる。プログラム変換部１２０、配備実行部１３０および検証部１４０は、プロセッサ１０１が実行するソフトウェアのモジュールとして実現できる。

[0058] 記憶部１１０は、プログラム変換部１２０、配備実行部１３０および検証部１４０の処理に用いられる各種のデータを記憶する。例えば、記憶部１１０が記憶するデータには、配備プログラムや配備プログラムの実行に伴って生成されたログ等が含まれる。配備プログラムは、ユーザによって作成され、記憶部１１０に予め格納される。

[0059] プログラム変換部１２０は、記憶部１１０に記憶された配備プログラムを変換することで、変換後配備プログラムを生成する。プログラム変換部１２０は、変換後配備プログラムを記憶部１１０に格納する。具体的には、プログラム変換部１２０は次の変換を行う。

[0060] 第１に、プログラム変換部１２０は、配備プログラムの変数に対する前回

の入力内容と今回の入力内容とを比較し、入力内容が変更されている変数を検出する。そして、プログラム変換部１２０は、当該変数について入力内容が変更されている旨を示すログを出力するように配備プログラムを変換する。

[0061] 第２に、プログラム変換部１２０は、配備プログラム内のコードを設定内容で区分したブロック（ブロックＩＤ（IDentifier）で識別される）という単位でラベリングする。プログラム変換部１２０は、各ブロックの実行を示すログを出力するように配備プログラムを変換する。すると、後述するように、ブロックと、ブロック内のコードが実行されることで出力された結果データ（実行結果）との対応をログにより容易に識別可能となる。また、ブロックと変数との対応関係を事前に把握しておけば、ログにより、結果データに関与した変数を容易に識別できる。ここで、ブロックは、第１の実施の形態のプログラム要素の一例である。

[0062] 配備実行部１３０は、プログラム変換部１２０により生成された変換後配備プログラムの内容を配備先サーバ２００に提供することで、配備先サーバ２００に仮想マシン群を配備する。配備先サーバ２００は、変換後配備プログラムの記述に従って、当該変換後配備プログラムの実行状況を示すログを生成し、配備実行部１３０に提供する。配備実行部１３０は、当該ログを記憶部１１０に格納する。また、配備実行部１３０は、配備先サーバ２００から仮想マシンの配備に伴って作成された複数の結果データを取得し、記憶部１１０に格納する。記憶部１１０には、配備プログラムの実行毎に、各結果データが取得され保持される。

[0063] 検証部１４０は、記憶部１１０に記憶されたログおよび結果データを分析することで、配備プログラムの冪等性を検証する。検証部１４０による検証の対象となる場合は、前回とは異なる内容の結果データが得られた場合である。第２の実施の形態の情報処理システムでは、仮想マシンの動作環境の仕様は適宜変更され得る。それに伴い、ユーザにより、ブロックに対する変数の入力内容が変更されたり、ブロック内のコードが書き換えられたりする。

このため、結果データの相違を調べるのみでは、冪等性の検証には不十分である。結果データを出力したブロックへの入力が変更されている可能性があるからである。

[0064] そこで、検証部１４０は、上記ログを参照して、前回と相違する結果データに対応するブロックと、当該ブロックに用いられる変数とを検索する。更に、検証部１４０は、当該ログを参照して、検索した変数に対する入力内容の変更状況を把握する。検証部１４０は、ブロックの変数に対する入力内容の変更状況に基づいて、結果データの相違が相違する要因と相違の妥当性を評価する。検証部１４０は、当該評価結果と前回とは異なる内容となった結果データとを出力する。例えば、検証部１４０は、ディスプレイ１１に出力内容を表示させる。

[0065] 図５は、記憶部に格納されるデータの例を示す図である。記憶部１１０は、旧配備プログラム１１１、配備プログラム１１２、変換後配備プログラム１１３、自由変数テーブル１１４、システムコールログ１１５、ログテーブル１１６、旧結果データ群１１７および結果データ群１１８を記憶する。

[0066] 旧配備プログラム１１１は、前回の仮想マシンの配備を行うために、ユーザにより作成された配備プログラムである。配備プログラム１１２は、今回の仮想マシンの配備を行うために、ユーザにより作成された配備プログラムである。

[0067] 変換後配備プログラム１１３は、プログラム変換部１２０により配備プログラム１１２が変換されることで生成された変換後の配備プログラムである。前述のように、プログラム変換部１２０は、変数およびブロックに関する所定のログを記録できるように、配備プログラム１１２を変換し、変換後配備プログラム１１３を生成する。

[0068] 自由変数テーブル１１４は、配備プログラムに用いられている自由変数と、当該自由変数が用いられているブロックとの対応関係を登録した情報である。自由変数とは、当該ブロック内で束縛されない（ブロック内のコードによって代入値が限定されない）変数である。自由変数は、ブロック外部から

与えられる入力であるということもできる。以下の説明では、単に変数という場合は、自由変数（ただし、自由変数を強調したい場合には、自由変数と明示することもある）を指すものとする。

[0069] システムコールログ１１５は、配備先サーバ２００で仮想マシンの配備に伴って生成されたシステムコールログである。システムコールログ１１５には、変換後配備プログラム１１３に基づく所定のログ（変数やブロックに関するログ）が含まれる。例えば、配備先サーバ２００のＯＳにＵｎｉｘ（登録商標）を用いるならｔｒｕｓｓコマンドによりシステムコールログを得ることができる。あるいは、配備先サーバ２００のＯＳにＬｉｎｕｘ（登録商標）を用いるならｓｔｒａｃｅコマンドによりシステムコールログを得ることができる。例えば、配備実行部１３０は、変換後配備プログラム１１３を用いた配備を行う前に、配備先サーバ２００により、これらのコマンドを実行させ、システムコールログを取得可能な状態にしておく。

[0070] ログテーブル１１６は、システムコールログ１１５を変換した情報である。ログテーブル１１６には、入力内容の変更された変数を識別する情報が含まれる。ログテーブル１１６には、ブロックと当該ブロックの処理により出力された結果データとの対応が含まれる。

[0071] 旧結果データ群１１７は、前回の配備により配備先サーバ２００で生成された旧結果データの集合である。例えば、旧結果データには、仮想マシンの配備に伴って生成されたネットワーク、アカウントおよび各種ソフトウェアの動作用のパラメータ等を設定したデータが含まれる。

[0072] 結果データ群１１８は、今回の配備により配備先サーバ２００で生成された結果データの集合である。旧結果データ群１１７と同様に、当該結果データには、仮想マシンの配備に伴って生成されたネットワーク、アカウントおよび各種ソフトウェアの動作用のパラメータ等を設定したデータが含まれる。

[0073] ここで、システムコールログ１１５、旧結果データ群１１７および結果データ群１１８は、配備先サーバ２００が備える記憶装置に格納されてもよい

。その場合、配備サーバ１００は、配備サーバ１００の処理に応じて、配備先サーバ２００からこれらの情報を取得することができる。

[0074] 図６は、配備プログラムに含まれるブロックの例を示す図である。配備プログラム１１２は、仮想マシンの動作環境を設定するための各種のブロック（プログラム要素）を含む。通常、１つの設定項目は、配備プログラムの一部分に連続して記述される。具体的には、配備プログラム１１２は、ディスク設定部分Ｂ１、ディレクトリ設定部分Ｂ２、ネットワーク設定部分Ｂ３、アカウント設定部分Ｂ４、ＮＦＳ（Network File System）設定部分Ｂ５およびＤＢ（DataBase）設定部分Ｂ６を含む。

[0075] ディスク設定部分Ｂ１は、ハードディスクに対するデバイスファイルの作成を行うブロックである。ディレクトリ設定部分Ｂ２は、ディレクトリの作成を行うブロックである。ネットワーク設定部分Ｂ３は、ネットワークに関する設定を行うブロックである。アカウント設定部分Ｂ４は、ユーザのアカウント設定を行うブロックである。ＮＦＳ設定部分Ｂ５は、ＮＦＳサーバの設定を行うブロックである。ＤＢ設定部分Ｂ６は、ＤＢＭＳ（DataBase Management System）の設定を行うブロックである。

[0076] 配備プログラム１１２の記述に従えば、ディスク設定、ディレクトリ設定、ネットワーク設定、アカウント設定、ＮＦＳ設定およびＤＢ設定が上側から下側へ順番に行われることになる。なお、仮想マシンの各項目の設定を上記順序で行うことは一例である。例示した項目以外の設定が行われてもよいし、任意の順序で行える設定については、順番を入れ替えてもよい。例示した項目の代わりに他の項目の設定が行われてもよい。

[0077] ここで、後述するように、プログラム変換部１２０は、配備プログラム１１２の各ブロックを識別してブロックＩＤでラベリングする。具体的には、ディスク設定部分Ｂ１のブロックＩＤを“１”とする。ディレクトリ設定部分Ｂ２のブロックＩＤを“２”とする。ネットワーク設定部分Ｂ３のブロックＩＤを“３”とする。アカウント設定部分Ｂ４のブロックＩＤを“４”とする。ＮＦＳ設定部分Ｂ５のブロックＩＤを“５”とする。ＤＢ設定部分Ｂ

６のブロックＩＤを“６”とする。

[0078] 図７は、配備プログラムの例を示す図である。図７では、旧配備プログラム１１１および配備プログラム１１２の一部の内容を例示している。以下、旧配備プログラム１１１および配備プログラム１１２の内容を図７に付した行番号により指し示す（なお、省略された行については、省略が１行か複数行かに関わらず、１つの行番号（例えば、“１”）を付している。以下、同様）。

[0079] 旧配備プログラム１１１について、各行には次のような記述がある。２行目は“ＤＩＳＫ＝“／ｄｅｖ／ｈｄａ１””である。４行目は“Ｗ＝“ｐｏｃｈｉ””である。５行目は“Ｘ＝“１９２．１６８．０．１””である。６行目は“Ｙ＝“ｔａｒｏ””である。７行目は“Ｚ＝ｆ（Ｘ）”である。９行目は“ｅｘｅｃｕｔｅ “ｐｖｃｒｅａｔｅ ＃{ＤＩＳＫ}” ｄｏ ｃｏｍｍａｎｄ “ｐｖｃｒｅａｔｅ ＃{ＤＩＳＫ}””である。１０行目は“ｅｎｄ”である。ここで、変数Ｚの定義に含まれる“ｆ（Ｘ）”は、変数Ｘを用いた所定の関数を示している（変数Ｚの定義の記述の中に変数Ｘが含まれている）。すなわち、変数Ｚは変数Ｘに依存する。例えば、“ｆ（ｘ）”は変数Ｘで示されるＩＰ（Internet Protocol）アドレスのホストアドレス部に対して、１を加算したＩＰアドレスを得る演算である（ただし、変数Ｘに対する別の演算でもよい）。

[0080] 配備プログラム１１２について、各行には次のような記述がある。２行目は“ＤＩＳＫ＝“／ｄｅｖ／ｈｄａ０””である。４行目は“Ｗ＝“ｐｏｃｈｉ””である。５行目は“Ｘ＝“１９２．１６８．１０．１””である。６行目は“Ｙ＝“ｊｉｒｏ””である。７行目は“Ｚ＝“ｆ（Ｘ）””である。９行目は“ｅｘｅｃｕｔｅ “ｐｖｃｒｅａｔｅ ＃{ＤＩＳＫ}” ｄｏ ｃｏｍｍａｎｄ “ｐｖｃｒｅａｔｅ ＃{ＤＩＳＫ}””である。１０行目は“ｅｎｄ”である。

[0081] 旧配備プログラム１１１および配備プログラム１１２を参照すると、プログラムの構文の特徴から次のことが分かる。配備プログラムの構文の規約で

は、等号“＝”を用いて変数に数値や文字列等の値の代入を記述し、等号の左側に変数を記述することになっている。よって、２行目および４行目～７行目は、それぞれ、変数ＤＩＳＫ，Ｗ，Ｘ，Ｙ，Ｚに対する代入文である。

[0082] また、配備プログラムの構文の規約では、コマンド“ｅｘｅｃｕｔｅ”および“ｅｎｄ”を用い、これら２つのコマンドの間に１つの設定内容を記述することになっている。９行目には、設定の実行を示すコマンド“ｅｘｅｃｕｔｅ”が記述されている。更に、続く１０行目には、１つの設定の終わりを示す“ｅｎｄ”が記述されている。したがって、９行目～１０行目で１つの設定内容となる。この場合、９行目～１０行目は１つのブロックである。

[0083] また、配備プログラムの構文の規約では、ブロック内で変数に代入された値を利用する場合、シャープ記号“＃”に続けて中括弧“｛｝”で変数名（“ＤＩＳＫ”や“Ｘ”等）を括ることになっている。９行目には文字列“＃｛ＤＩＳＫ｝”が含まれている。したがって、９行目～１０行目のブロックは、変数ＤＩＳＫを用いるブロックである。

[0084] 更に、旧配備プログラム１１１と配備プログラム１１２とを対比すると、配備プログラム１１２の上記各行について次のことが分かる。２行目の記述は変更されている。４行目の記述は変更されていない。５行目の記述は変更されている。６行目の記述は変更されている。７行目の記述は変更されていない（ただし、後述するように変数Ｚへの代入値に関しては別の判断となる）。９行目の記述は変更されていない。１０行目の記述は変更されていない。

[0085] プログラム変換部１２０は、上記のように構文上の規約に従って、旧配備プログラム１１１や配備プログラム１１２から、変数やブロックを抽出することができる。その際に用いる構文上の規約は、プログラム変換部１２０に予め与えられる。

[0086] 図８は、変換後配備プログラムの例を示す図である。変換後配備プログラム１１３は、プログラム変換部１２０により配備プログラム１１２が変換された結果である。以下、変換後配備プログラム１１３の内容を図８に付した

行番号により指し示す。変換後配備プログラム１１３には例えば次の記述がある。

[0087] ２行目は“ＤＩＳＫ＝“／ｄｅｖ／ｈｄａ０””である。３行目は“ｌｏｇ＿ｗｒｉｔｅ（“ＤＩＳＫ　ｃｈａｎｇｅｄ”）”である。５行目は“Ｗ＝“ｐｏｃｈｉ””である。６行目は“Ｘ＝“１９２．１６８．１０．１””である。７行目は“ｌｏｇ＿ｗｒｉｔｅ（“Ｘ　ｃｈａｎｇｅｄ”）”である。

[0088] ８行目は“Ｙ＝ｊｉｒｏ”である。９行目は“ｌｏｇ＿ｗｒｉｔｅ（“Ｙ　ｃｈａｎｇｅｄ”）”である。１０行目は“Ｚ＝ｆ（Ｘ）”である。１１行目は“ｌｏｇ＿ｗｒｉｔｅ（“Ｚ　ｃｈａｎｇｅｄ；ｕｓｅ　Ｘ”）”である。

[0089] １３行目は“ｌｏｇ＿ｗｒｉｔｅ（“ｂｌｏｃｋ　１　ｅｎｔｅｒ”）”である。１４行目は“ｅｘｅｃｕｔｅ　“ｐｖｃｒｅａｔｅ　＃｛ＤＩＳＫ｝”　ｄｏ　ｃｏｍｍａｎｄ　“ｐｖｃｒｅａｔｅ　＃｛ＤＩＳＫ｝””である。１５行目は“ｅｎｄ”である。１６行目は“ｌｏｇ＿ｗｒｉｔｅ（“ｂｌｏｃｋ　１　ｅｘｉｔ”）”である。

[0090] このうち、３，７，９，１１，１３，１６行目は、プログラム変換部１２０により挿入されたコードである。

ここで、関数“ｌｏｇ＿ｗｒｉｔｅ（“＜文字列＞”）”は、“＜文字列＞”部分に記述された文字列を、配備先サーバ２００により、システムコールログ１１５に出力させるものである。当該関数は配備先サーバ２００の関数ライブラリに予め登録されている。

[0091] 例えば、“ｌｏｇ＿ｗｒｉｔｅ（“ＤＩＳＫ　ｃｈａｎｇｅｄ”）”であれば、配備先サーバ２００は“ＤＩＳＫ　ｃｈａｎｇｅｄ”の文字列をシステムコールログ１１５に出力する。これは、変数ＤＩＳＫの入力内容が変更されていることを示している。

[0092] また、１１行目の記述“ｌｏｇ＿ｗｒｉｔｅ（“Ｚ　ｃｈａｎｇｅｄ；ｕｓｅ　Ｘ”）”は、変数Ｚに対する入力内容が変更されていることを示して

いる。特に、当該記述において、“ｕｓｅ　Ｘ”の部分は、変数Ｚの入力内容が変数Ｘの入力内容に依存していることを示す。図７の例示では変数Ｚの定義自体は変更されていない。しかし、変数Ｘの入力内容が変更されていれば、プログラム変換部１２０は変数Ｚの入力内容も変更されているとみなす。

[0093] 更に、１３行目の記述“ｌｏｇ＿ｗｒｉｔｅ（“ｂｌｏｃｋ　１　ｅｎｔｅｒ”）”および１６行目の記述“ｌｏｇ＿ｗｒｉｔｅ（“ｂｌｏｃｋ　１　ｅｘｉｔ”）”は、１４行目～１５行目で記述されたブロックの実行開始および終了を示す。変換後配備プログラム１１３の１４行目～１５行目は、配備プログラム１１２の９行目～１０行目に対応している。プログラム変換部１２０は、これらのコードを挿入することで、当該ブロックをブロックＩＤ“１”としてラベリングしている。

[0094] 図９は、自由変数テーブルの例を示す図である。自由変数テーブル１１４は、プログラム変換部１２０により生成される。自由変数テーブル１１４には、ブロックＩＤおよび自由変数集合の項目が含まれる。ブロックＩＤの項目には、ブロックＩＤが登録される。自由変数集合の項目には、当該ブロックＩＤで示されるブロックで利用される自由変数の集合が登録される。

[0095] 例えば、自由変数テーブル１１４には、ブロックＩＤが“１”、自由変数集合が“ＤＩＳＫ”という情報が登録される。これは、ブロックＩＤ“１”のブロックにおいて、自由変数ＤＩＳＫが用いられていることを示す。

[0096] また、自由変数テーブル１１４には、ブロックＩＤが“２”、自由変数集合が設定なし“－（ハイフン）”という情報が登録されている。これは、ブロックＩＤ“２”のブロックにおいて、利用される自由変数がないことを示す。

[0097] また、自由変数テーブル１１４には、ブロックＩＤが“３”、自由変数集合が“Ｗ，Ｘ”という情報が登録されている。これは、ブロックＩＤ“３”のブロックにおいて、自由変数Ｗ，Ｘが用いられていることを示す。

[0098] 図１０は、システムコールログの例を示す図である。システムコールログ

１１５は、配備先サーバ２００によって生成される。以下、システムコールログ１１５の内容を図１０に付した行番号により指し示す。システムコールログ１１５には例えば次の記述がある。

[0099] ２行目は“ｏｐｅｎ　ｆ”である。“ｆ”は所定のファイルを示す。２行目は“ｆ”で示されるファイルのオープン処理である。３行目は“ｗｒｉｔｅ　ｆ　“ＤＩＳＫ　ｃｈａｎｇｅｄ””である。“ｆ”で示されるファイルに“ＤＩＳＫ　ｃｈａｎｇｅｄ”を書き込んだことを示す。４行目は“ｃｌｏｓｅ　ｆ”である。“ｆ”で示されるファイルのクローズ処理である。

[0100] ６行目～８行目（“ｆ”への“Ｘ　ｃｈａｎｇｅ”の書き込み）、１０行目～１２行目（“ｆ”への“Ｙ　ｃｈａｎｇｅｄ”の書き込み）、１４行目～１６行目（“ｆ”への“Ｚ　ｃｈａｎｇｅｄ；ｕｓｅ　Ｘ”）も同様の処理を示している。また、１８行目～２０行目（“ｆ”への“ｂｌｏｃｋ　１　ｅｎｔｅｒ”の書き込み）、２６行目～２８行目（“ｆ”への“ｂｌｏｃｋ　１　ｅｘｉｔ”の書き込み）も同様の処理を示している。

[0101] ２２行目は、“ｏｐｅｎ　／ｖａｒ／ｌｏｇ／ｍｅｓｓａｇｅｓ”である。“／ｖａｒ／ｌｏｇ／ｍｅｓｓａｇｅｓ”ファイルのオープン処理である。２３行目は“ｗｒｉｔｅ　／ｖａｒ／ｌｏｇ／ｍｅｓｓａｇｅｓ　“／ｄｅｖ／ｈｄａ０　ｃｒｅａｔｅｄ””である。同ファイルに“／ｄｅｖ／ｈｄａ０　ｃｒｅａｔｅｄ”を書き込んだことを示す。２４行目は“ｃｌｏｓｅ　／ｖａｒ／ｌｏｇ／ｍｅｓｓａｇｅｓ”である。同ファイルのクローズ処理である。

[0102] 図１１は、システムコールログの例（続き）を示す図である。更に、システムコールログ１１５には例えば次の記述がある。３１行目～３３行目、４０行目～４２行目、４４行目～４６行目、５２行目～５４行目、５６行目～５８行目、６４行目～６６行目は、前述の１８行目～２０行目および２６行目～２８行目と同様である（記録されているブロックＩＤが異なる）。

[0103] また、３５行目は、“ｏｐｅｎ　／ｅｔｃ／ｈｏｓｔｓ”である。“／ｅｔｃ／ｈｏｓｔｓ”ファイルのオープン処理である。３６行目は、“ｗｒｉ

ｔｅ　／ｅｔｃ／ｈｏｓｔｓ　“１２７．０．０．１　ｌｏｃａｌｈｏｓｔ””である。同ファイルに“１２７．０．０．１　ｌｏｃａｌｈｏｓｔ”を書き込んだことを示す。３７行目は、“ｗｒｉｔｅ　／ｅｔｃ／ｈｏｓｔｓ　“１９２．１６８．１０．１　ｐｏｃｈｉ””である。同ファイルに“１９２．１６８．１０．１　ｐｏｃｈｉ”を書き込んだことを示す。３８行目は、“ｃｌｏｓｅ　／ｅｔｃ／ｈｏｓｔｓ”である。同ファイルのクローズ処理である。

[0104] また、４８行目は、“ｏｐｅｎ　／ｅｔｃ／ｐａｓｓｗｄ”である。“／ｅｔｃ／ｐａｓｓｗｄ”ファイルのオープン処理である。４９行目は、“ｗｒｉｔｅ　／ｅｔｃ／ｐａｓｓｗｄ　“ｊｉｒｏ　．．．””である。同ファイルに“ｊｉｒｏ　．．．”を書き込んだことを示す。５０行目は“ｃｌｏｓｅ　／ｅｔｃ／ｐａｓｓｗｄ”である。同ファイルのクローズ処理である。

[0105] また、６０行目は、“ｏｐｅｎ　／ｅｔｃ／ｅｘｐｏｒｔｓ”である。“／ｅｔｃ／ｅｘｐｏｒｔｓ”ファイルのオープン処理である。６１行目は、“ｗｒｉｔｅ　／ｅｔｃ／ｅｘｐｏｒｔｓ　“／ｈｏｍｅ／ｎｆｓ　１９２．１６８．１０．２／２４　（ｒｗ）””である。同ファイルに“／ｈｏｍｅ／ｎｆｓ　１９２．１６８．１０．２／２４　（ｒｗ）”を書き込んだことを示す。６２行目は“ｃｌｏｓｅ　／ｅｔｃ／ｅｘｐｏｒｔｓ”である。同ファイルのクローズ処理である。

[0106] 図１２は、ログテーブルの例を示す図である。ログテーブル１１６は、システムコールログ１１５に基づいて検証部１４０により生成される。ログテーブル１１６は、ブロックＩＤおよびログの項目を含む。ブロックＩＤの項目にはブロックＩＤが登録される。ただし、ブロックＩＤの登録がないレコードもある。ブロックＩＤの登録がない場合、当該レコードは変数に対する入力内容の変更に関するものである。ログの項目には、ログの記述内容が登録される。

[0107] 例えば、ログテーブル１１６には、ブロックＩＤが設定なし“－”、ログ

が“ＤＩＳＫ　ｃｈａｎｇｅｄ”という情報が登録されている。これは、図１０で例示したシステムコールログ１１５の３行目の記述に対応している（“ｗｒｉｔｅ　ｆ”の記述は後の処理で用いないため省いている。以下、同様）。当該レコードは変数ＤＩＳＫが変更されたことを示す。他の変数Ｘ，Ｙ，Ｚに関するレコードも同様の意味である。

[0108] また、ログテーブル１１６には、ブロックＩＤが“１”、ログが“ｂｌｏｃｋ　１　ｅｎｔｅｒ”という情報が登録されている。これは、図１０で例示したシステムコールログ１１５の１９行目の記述に対応している。当該レコードは、ブロックＩＤ“１”のブロックに対して、ブロックの実行開始を示すログ“ｂｌｏｃｋ　１　ｅｎｔｅｒ”が記録されたことを示す。また、以降の記述がブロックＩＤ“１”のブロックに関するものであることを示す。

[0109] また、ログテーブル１１６には、ブロックＩＤが“１”、ログが“ｗｒｉｔｅ　／ｖａｒ／ｌｏｇ／ｍｅｓｓａｇｅｓ　“／ｄｅｖ／ｈｄａ０　ｃｒｅａｔｅｄ”””という情報が登録されている。これは、図１０で例示したシステムコールログ１１５の２３行目の記述に対応している。当該レコードは、ブロックＩＤ“１”のブロックの処理により、“／ｄｅｖ／ｈｄａ０”というデバイスファイルが作成され、その結果が“／ｖａｒ／ｌｏｇ／ｍｅｓｓａｇｅｓ”ファイルに書き込まれたことを示す。

[0110] 更に、ログテーブル１１６には、ブロックＩＤが“１”、ログが“ｂｌｏｃｋ　１　ｅｘｉｔ”という情報が登録されている。これは、図１０で例示したシステムコールログ１１５の２７行目の記述に対応している。当該レコードは、ブロックＩＤ“１”のブロックに対して、ブロックの実行終了を示すログ“ｂｌｏｃｋ　１　ｅｘｉｔ”が記録されたことを示す。また、ブロックＩＤ“１”のブロックの処理が完了したことを示す。他のブロックＩＤ“２”、“３”等に関するレコードも同様の意味である。

[0111] 図１３は、検証処理の全体を示すフローチャートである。以下、図１３に示す処理をステップ番号に沿って説明する。

（Ｓ１）プログラム変換部１２０は、検証の対象となる配備プログラム１１２を変換することで、変換後配備プログラム１１３を生成する。当該変換処理の詳細は後述する。

[0112] （Ｓ２）配備実行部１３０は、変換後配備プログラム１１３を用いて、配備先サーバ２００に対する仮想マシンの配備を行う。具体的には、配備実行部１３０は、変換後配備プログラム１１３に含まれる各ブロックのコマンドを配備先サーバ２００に提供することで、仮想マシンの動作環境の設定を行わせる。配備先サーバ２００では、配備の実行に伴ってシステムコールログ１１５が生成される。また、配備先サーバ２００では、変換後配備プログラム１１３の各ブロックのコマンドに従って、結果データ群１１８が生成される。

[0113] （Ｓ３）配備実行部１３０は、変換後配備プログラム１１３による配備が完了すると、システムコールログ１１５および結果データ群１１８を配備先サーバ２００から取得し、記憶部１１０に格納する。結果データ群１１８には、前述した“ｍｅｓｓａｇｅｓ”（ログファイル）、“ｈｏｓｔｓ”（ホスト名の静的定義ファイル）、“ｐａｓｓｗｄ”（アカウント設定ファイル）および“ｅｘｐｏｒｔｓ”（ＮＦＳ設定ファイル）等の結果データが含まれる。

[0114] （Ｓ４）検証部１４０は、記憶部１１０に記憶されたシステムコールログ１１５に基づいて、ログテーブル１１６を作成する。具体的には、システムコールログ１１５のうち、変数に関するファイル“ｆ”への出力内容をブロックＩＤなし（“－”）でログテーブル１１６に登録する。ログテーブル１１６の例では、変数のログに対しては“ｗｒｉｔｅ　ｆ”の記述も省いている。また、検証部１４０は、システムコールログ１１５のうち、ブロックに関する各ファイルへの出力内容を、当該ブロックのブロックＩＤと出力先のファイルとが分かるように対応付けてログテーブル１１６に登録する。ログテーブル１１６の具体的な内容は、図１２で例示した通りである。

[0115] （Ｓ５）検証部１４０は、ログテーブル１１６および結果データ群１１８

を分析することで、配備プログラム１１２の冪等性を検証する。検証部１４０は、検証結果を出力する。例えば、検証部１４０は、検証結果を示す画像をディスプレイ１１に表示させることで、ユーザに検証結果を提示する。

[0116] 図１４は、配備プログラムの変換例を示すフローチャートである。以下、図１４に示す処理をステップ番号に沿って説明する。以下に示す処理は、図１３のステップＳ１に相当する。

[0117] （Ｓ１１）プログラム変換部１２０は、検証の比較対象として、旧配備プログラム１１１のユーザによる指定を受け付ける。また、プログラム変換部１２０は、検証対象として、配備プログラム１１２のユーザによる指定を受け付ける。

[0118] （Ｓ１２）プログラム変換部１２０は、旧配備プログラム１１１と配備プログラム１１２とを比較して、配備プログラム１１２の変数のうち、旧配備プログラム１１１と代入値の異なる変数を１つ検索する。以降のステップＳ１４～Ｓ１６を実行済の変数は検索の対象外となる。図７で例示したように、プログラム変換部１２０は、旧配備プログラム１１１および配備プログラム１１２の構文から変数および代入値を特定できるし、同じ変数に対する代入値の相違も把握できる。

[0119] ここで、プログラム変換部１２０は、他の変数に依存する変数について、当該他の変数の代入値の変更があるとき、当該変数の代入値にも変更があると判断する。当該他の変数の代入値の変更がないとき、当該変数の代入値にも変更がないと判断する。図７で例示したように、プログラム変換部１２０は、検索された変数の定義の記述に、他の変数名が含まれているか否かを判定することで、検索した変数が他の変数に依存するか否かを判定できる。配備プログラム１１２の例でいえば、変数Ｘに関する“Ｘ＝“１９２．１６８．１０．１””の記述は他の変数を含まないから、変数Ｘは他の変数に依存しない。変数Ｚに関する“Ｚ＝ｆ（Ｘ）”は他の変数Ｘを含むから、変数Ｚは変数Ｘに依存する。

[0120] （Ｓ１３）プログラム変換部１２０は、何れかの変数を検索できたか否か

を判定する。検索できた場合、処理をステップＳ１４に進める。検索できなかった場合、処理をステップＳ１７に進める。

[0121] （Ｓ１４）プログラム変換部１２０は、配備プログラム１１２を参照して、検索した変数が他の変数に依存するか否かを判定する。依存する場合、処理をステップＳ１５に進める。依存しない場合、処理をステップＳ１６に進める。

[0122] （Ｓ１５）プログラム変換部１２０は、当該変数の直後に、“ｌｏｇ＿ｗｒｉｔｅ（“＜変数名＞　ｃｈａｎｇｅｄ：ｕｓｅ　＜他の変数名＞”）”の記述を挿入する。例えば、ステップＳ１３で検索された変数が変数Ｚであれば、“ｌｏｇ＿ｗｒｉｔｅ（“Ｚ　ｃｈａｎｇｅｄ：ｕｓｅ　Ｘ”）”という記述を“Ｚ＝ｆ（Ｘ）”の直後に挿入する。これは、図８で例示した変換後配備プログラム１１３の１１行目に相当する。そして、処理をステップＳ１２に進める。

[0123] （Ｓ１６）プログラム変換部１２０は、当該変数の直後に、“ｌｏｇ＿ｗｒｉｔｅ（“＜変数名＞　ｃｈａｎｇｅｄ”）”の記述を挿入する。例えば、ステップＳ１３で検索された変数が変数Ｘであれば、“ｌｏｇ＿ｗｒｉｔｅ（“Ｘ　ｃｈａｎｇｅｄ”）”という記述を“Ｘ＝“１９２．１６８．１０．１””の直後に挿入する。これは、図８で例示した変換後配備プログラム１１３の７行目に相当する。そして、処理をステップＳ１２に進める。

[0124] （Ｓ１７）プログラム変換部１２０は、配備プログラム１１２を参照して、ブロックを１つ検索する。以降のステップＳ１９を実行済のブロックは検索の対象外となる。図７で例示したように、プログラム変換部１２０は、配備プログラム１１２の構文からブロックを特定できる。

[0125] （Ｓ１８）プログラム変換部１２０は、何れかのブロックを検索できたか否かを判定する。検索できた場合、処理をステップＳ１９に進める。検索できなかった場合、処理を終了する（変換後配備プログラム１１３の生成が完了する）。

[0126] （Ｓ１９）プログラム変換部１２０は、検索したブロックに対してブロッ

クＩＤを付与する。例えば、ブロックＩＤとして番号を昇順に付与することが考えられる。プログラム変換部１２０は、当該ブロックの直前に、“ｌｏｇ＿ｗｒｉｔｅ（“ｂｌｏｃｋ　＜ブロックＩＤ＞　ｅｎｔｅｒ”）”の記述を挿入する。例えば、ブロックＩＤ“１”のブロックであれば、“ｌｏｇ＿ｗｒｉｔｅ（“ｂｌｏｃｋ　１　ｅｎｔｅｒ”）”という記述を当該ブロックの直前に挿入する。プログラム変換部１２０は、当該ブロックの直後に、“ｌｏｇ＿ｗｒｉｔｅ（“ｂｌｏｃｋ　＜ブロックＩＤ＞　ｅｘｉｔ”）”の記述を挿入する。例えば、ブロックＩＤ“１”のブロックであれば、“ｌｏｇ＿ｗｒｉｔｅ（“ｂｌｏｃｋ　１　ｅｘｉｔ”）”という記述を当該ブロックの直後に挿入する。

[0127] （Ｓ２０）プログラム変換部１２０は、検索したブロック内の自由変数を、ステップＳ１９で付与したブロックＩＤに対応付けて自由変数テーブル１１４に登録する。図７で例示したように、プログラム変換部１２０は、配備プログラム１１２の構文によってブロックで用いられる変数を特定し、ブロック内で代入式の定義がない変数（あるいは、ブロック内の束縛変数以外の変数）を自由変数として特定できる。そして、処理をステップＳ１７に進める。

[0128] このようにして、プログラム変換部１２０は、配備プログラム１１２を変換することで、変換後配備プログラム１１３を生成する。なお、旧配備プログラム１１１および配備プログラム１１２の例では、同プログラム内に変数の入力値が含まれるものとしたが、別個のファイルを用意して各変数の入力値を与えてもよい。その場合、プログラム変換部１２０は、旧配備プログラム１１１と配備プログラム１１２と各配備プログラムに対する入力内容のデータとを比較することで、ステップＳ１２の処理を実行できる。

[0129] なお、配備サーバ１００は、旧配備プログラム１１１についても上記と同様に変換し、変換後の旧配備プログラム１１１による配備を配備先サーバ２００で事前に行っている。配備サーバ１００は、当該事前の配備により旧結果データ群１１７を予め取得し、記憶部１１０に格納している。旧結果デー

タ群１１７には、結果データ群１１８の各結果データに対応する旧結果データが含まれる。

[0130] 図１５は、配備プログラムの冪等性の検証例を示すフローチャートである。以下、図１５に示す処理をステップ番号に沿って説明する。以下に示す処理は、図１３のステップＳ５に相当する。

[0131] （Ｓ２１）検証部１４０は、旧結果データ群１１７に含まれる各旧結果データと、結果データ群１１８に含まれる今回の各結果データとを比較する。検証部１４０は、同じファイル名の結果データ同士を比較する。例えば、今回の結果データとして“ｈｏｓｔｓ”ファイルがあれば、当該結果データに対応する前回の“ｈｏｓｔｓ”ファイルと今回の“ｈｏｓｔｓ”ファイルとを比較する。

[0132] （Ｓ２２）検証部１４０は、旧結果データと内容の相違する結果データがあるか否かを判定する。ある場合、処理をステップＳ２３に進める。ない場合、処理をステップＳ２９に進める。“ｈｏｓｔｓ”ファイルの例でいえば、前回の“ｈｏｓｔｓ”ファイルに“１９２．１６８．０．１　ｐｏｃｈｉ”が記述され、今回の“ｈｏｓｔｓ”ファイルに“１９２．１６８．１０．１　ｐｏｃｈｉ”が記述されている。この場合、今回の“ｈｏｓｔｓ”ファイルの当該レコードは、前回の“ｈｏｓｔｓ”ファイルには含まれていないから、内容が相違していることになる。

[0133] （Ｓ２３）検証部１４０は、ステップＳ２２で旧結果データと相違すると判定された結果データのうち、未分析（ステップＳ２４～Ｓ２７の処理を行っていないもの）のものを１つ抽出する。

[0134] （Ｓ２４）検証部１４０は、当該結果データを出力したブロックの変数の分析処理を行う。検証部１４０は、当該分析により、旧結果データが得られたとき（旧配備プログラム１１１または旧配備プログラム１１１をプログラム変換部１２０により変換したプログラムで配備を行ったとき）に対して、当該変数の入力内容に変更があったかを確認する。変数の分析処理の詳細は後述する。

[0135] （Ｓ２５）検証部１４０は、ステップＳ２４の分析結果に基づいて、当該結果データを出力したブロックの変数の代入値に変更があったか否かを判定する。変更があった場合、処理をステップＳ２６に進める。変更がなかった場合、処理をステップＳ２７に進める。

[0136] （Ｓ２６）検証部１４０は、結果データの相違を妥当と判断する。検証部１４０は、当該相違が妥当である旨を出力する。検証部１４０は、相違箇所（当該結果データ）と当該結果データを出力したブロックと入力変更のあった変数箇所とを当該相違の要因として出力する。検証部１４０は、結果データ内の相違箇所（相違するレコード）を出力することもできる。具体的には、ステップＳ２２で例示した“ｈｏｓｔｓ”ファイルに対して、“１９２．１６８．１０．１　ｐｏｃｈｉ”のレコードを相違箇所として出力してもよい。そして、処理をステップＳ２８に進める。

[0137] （Ｓ２７）検証部１４０は、結果データの相違を非妥当と判断する。検証部１４０は、当該相違が非妥当である旨を出力する。検証部１４０は、相違箇所（当該結果データ）と当該結果データを出力したブロックとを当該相違の要因として出力する。検証部１４０は、結果データ内の相違箇所（相違するレコード）を出力することもできる。具体例は、ステップＳ２６と同様である。

[0138] （Ｓ２８）検証部１４０は、ステップＳ２２で旧結果データと相違があると判定された全ての結果データを分析済であるか否かを判定する。全て分析済であれば、処理を終了する。全て分析済でなければ、処理をステップＳ２３に進める。

[0139] （Ｓ２９）検証部１４０は、旧結果データ群１１７と結果データ群１１８との間に差異がないことを出力する。この場合、ユーザは、配備プログラム１１２の冪等性は確保されていると判断できる。

[0140] 図１６は、ログに基づく変数の分析例を示すフローチャートである。以下、図１６に示す処理をステップ番号に沿って説明する。以下に示す処理は、図１５のステップＳ２４に相当する。

[0141] （Ｓ３１）検証部１４０は、ログテーブル１１６を参照して、旧結果データと内容の相違する結果データを出力したブロックを検索する。例えば、検証部１４０は、結果データ“ｈｏｓｔｓ”ファイルを出力したブロックのブロックＩＤ“３”を、ログテーブル１１６から取得できる。

[0142] （Ｓ３２）検証部１４０は、自由変数テーブル１１４を参照して、検索されたブロックで用いられる変数を検索する。例えば、検証部１４０は、ブロックＩＤ“３”のブロックに対して、当該ブロックで用いられる変数Ｗ，Ｘを自由変数テーブル１１４から取得できる。

[0143] （Ｓ３３）検証部１４０は、ステップＳ３２の検索の結果、該当のブロックで用いられる何れかの変数を取得できたか否かを判定する。取得できた場合、処理をステップＳ３４に進める。取得できない場合、処理を終了する。取得できない場合とは、自由変数テーブル１１４において、該当のブロックに対応する自由変数集合の設定がない（“－”）場合である。なお、ステップＳ３２で複数の変数が取得されている場合、以降のステップＳ３４～Ｓ４０の処理は、変数毎に実行される（例えば、変数Ｗについての処理を実行した後、変数Ｘについての処理を実行する等）。

[0144] （Ｓ３４）検証部１４０は、ログテーブル１１６を参照して、着目する変数（ステップＳ３２の検索で取得された何れかの変数）について、着目するブロックよりも前のログから変更ログ（“＜変数名＞　ｃｈａｎｇｅｄ”）を検索する。例えば、変数Ｗであれば着目するブロック（ブロックＩＤ“３”のブロック）よりも前のログに変更ログは存在していない。また、変数Ｘであれば、着目する同ブロックよりも前のログに変更ログ“Ｘ　ｃｈａｎｇｅｄ”が存在している。

[0145] （Ｓ３５）検証部１４０は、着目する変数に対する変更ログがあるか否かを判定する。変更ログがある場合、処理をステップＳ３６に進める。変更ログがない場合、処理を終了する。例えば、変数Ｗであれば前述のように変更ログがないので変数Ｗに関する処理は終了となる。変数Ｘであれば前述のように変更ログがあるので処理をステップＳ３６に進めることになる。

[0146] （Ｓ３６）検証部１４０は、着目する変数を、代入値の変更された変数として記憶部１１０に記録する。

（Ｓ３７）検証部１４０は、ログテーブル１１６を参照して、着目する変数が依存する他の変数があるか否かを判定する。依存する他の変数がある場合、処理をステップＳ３８に進める。依存する他の変数がない場合、処理を終了する。検証部１４０は、着目する変数の変更ログに“ｕｓｅ ＜他の変数名＞”の記述が含まれていれば、当該着目する変数が依存する他の変数があると判定できる。例えば、ログテーブル１１６には変数Ｚに対して“Ｚｃｈａｎｇｅｄ：ｕｓｅ Ｘ”という記述が含まれているから、着目する変数がＺであれば、変数Ｚが依存する他の変数として変数Ｘを特定できる。当該記述が含まれていなければ、当該着目する変数が依存する他の変数がないと判定できる。

[0147] （Ｓ３８）検証部１４０は、着目する変数と他の変数との依存関係を記憶部１１０に記録する。

（Ｓ３９）検証部１４０は、ログテーブル１１６を参照して、着目する変数の変更ログよりも前の箇所から、当該変数が依存する他の変数を検索する。検証部１４０は当該他の変数を着目する変数に切り替える。そして、処理をステップＳ３７に進める。

[0148] このようにして、検証部１４０は、入力内容の変更された変数および変数の依存関係を記憶部１１０に記録する。例えば、検証部１４０は、グラフ構造を用いて変数の依存関係を記録することができる。例えば、図１５のステップＳ２６において、検証部１４０は、結果データの相違の要因となった変数間の依存関係を、ユーザに提示することもできる。

[0149] 図１７は、検証結果の出力例（その１）を示す図である。ＧＵＩ（Graphical User Interface）５１０は、図１５のステップＳ２６の出力例である。ＧＵＩ５２０は、図１５のステップＳ２７の出力例である。ＧＵＩ５１０，５２０は、検証部１４０により生成され、ディスプレイ１１に表示される。検証部１４０は、ＧＵＩ５１０，５２０により、配備プログラム１１２の検証

結果をユーザに提示できる。

[0150] ＧＵＩ５１０は、旧結果データに対する相違が妥当であると判断された結果データの一覧を含む。表示内容には、当該相違が妥当であること、相違のある箇所（結果データ）、要因となったブロックおよび変数を示す文字列が含まれる。

[0151] 検証部１４０は、相違のある箇所として、結果データを示す情報（例えば、“／ｅｔｃ／ｈｏｓｔｓ”）を提示する。結果データとして、デバイスファイル（例えば、“／ｄｅｖ／ｈｄａ０”）や他の種類のファイル名を提示することもできる。

[0152] また、検証部１４０は、相違の要因となったブロックの配備プログラム１１２における記述箇所を提示する。具体的には、当該ブロックの記述箇所が配備プログラム１１２内の何行目に相当するか（例えば、“ｌｉｎｅｓ　ｘ２－ｙ２”）を提示する。なお、各ブロックが配備プログラム１１２の何行目に相当するかは、例えば、変換後配備プログラム１１３において、配備プログラム１１２のどの行に“ｌｏｇ＿ｗｒｉｔｅ（“ｂｌｏｃｋ　＜ブロックＩＤ＞　ｅｎｔｅｒ”）”を挿入したかにより把握できる。

[0153] また、検証部１４０は、要因となった変数の配備プログラム１１２における記述箇所（例えば、代入式やブロック内の箇所）を提示する。具体的には、当該変数の記述箇所が配備プログラム１１２内の何行目に相当するか（例えば、“ｌｉｎｅ　ｂ”）を提示する。

[0154] ＧＵＩ５２０は、旧結果データに対する相違が非妥当であると判断された結果データの一覧を含む。表示内容には、当該相違が非妥当であること、相違のある箇所（結果データ）、要因となったブロックおよび変数を示す文字列が含まれる。

[0155] 検証部１４０は、相違のある箇所として、結果データを示す情報（例えば、“／ｕｓｒ／ｌｉｂ／ｘｘｘ”）を提示する。また、検証部１４０は、相違の要因となったブロックの配備プログラム１１２における記述箇所を提示する。具体的な提示方法は、ＧＵＩ５１０と同様である。

[0156] また、検証部１４０は、ＧＵＩ５２０において、相違の要因となった変数を提示しない。結果の相違が非妥当な場合、変数に対する入力内容の変更はないと判断されているからである。したがって、この場合、ユーザは、ブロック自体の処理が要因となって配備プログラム１１２の冪等性が失われていると判断できる。

[0157] 図１８は、検証結果の出力例（その２）を示す図である。ＧＵＩ５１１は、“／ｅｔｃ／ｅｘｐｏｒｔｓ”が旧結果データ（例えば、前回配備時に得られた“／ｅｔｃ／ｅｘｐｏｒｔｓ”）と相違する各要因の関連を示している。例えば、検証部１４０は、ＧＵＩ５１０に表示された相違箇所（“／ｅｔｃ／ｅｘｐｏｒｔｓ”）、要因となるブロック箇所（“ｌｉｎｅｓ　ｘ４－ｙ４”）または変数箇所（“ｌｉｎｅｓ　ｄ，ｂ”）のユーザによる選択入力を受け付ける。すると、検証部１４０は、選択された箇所に応じて、ＧＵＩ５１１を生成し、ディスプレイ１１に表示させる。ユーザは、例えば、入力デバイス１２を用いて、当該選択入力を行える。

[0158] 例えば、ＧＵＩ５１１では、相違箇所とブロックと変数Ｚと変数Ｘとを関連線で結ぶことで、各要因の関連が示されている。検証部１４０は、図１６のステップＳ３７～Ｓ３９で記録された変数間の依存関係に基づいて、変数Ｚと変数Ｘとの関連をユーザに提示することができる。

[0159] また、ＧＵＩ５２１は、“／ｕｓｒ／ｌｉｂ／ｘｘｘ”が旧結果データ（例えば、前回配備時に得られた“／ｕｓｒ／ｌｉｂ／ｘｘｘ”）と相違する各要因の関連を示している。例えば、検証部１４０は、ＧＵＩ５２０に表示された相違箇所（“／ｕｓｒ／ｌｉｂ／ｘｘｘ”）または要因となるブロック箇所（“ｌｉｎｅｓ　ｘ５－ｙ５”）のユーザによる選択入力を受け付ける。すると、検証部１４０は、選択された箇所に応じて、ＧＵＩ５２１を生成し、ディスプレイ１１に表示させる。例えば、ＧＵＩ５２０では、相違箇所とブロックとを関連線で結ぶことで、各要因の関連が示されている。

[0160] 図１９は、検証結果の出力例（その３）を示す図である。ＧＵＩ５１２は、“／ｅｔｃ／ｅｘｐｏｒｔｓ”の旧結果データと相違する各要因の関連を

示す他の出力例である。例えば、ブロックＩＤ“５”のブロックが、前述の変数Ｚ以外の他の変数も用いるなら、当該ブロックに関連する変数の系統として、変数Ｚの系統と、当該他の変数の系統とを分離して、ユーザに提示することもできる。

[0161] ここで、第２の実施の形態の情報処理システムでは、仮想マシン自体の設定変更や実行させるソフトウェアの追加・変更等の仕様変更に伴って、配備プログラムもメンテナンスされる。例えば、配備プログラムにおいて、ユーザアカウントの追加といったパラメータの一部変更が行われることもある。また、例えば、ソフトウェアや設定内容の追加に伴ってブロック自体の記述の変更が行われることもある。

[0162] この場合、各ブロックによって生成された各結果データに対し、その妥当性（本例では、冪等であるか）をどのように検証するかが問題となる。結果データが旧結果データと相違する場合、その相違の要因が、一部の変数の設定値の変更なのか、何れかのブロック自体の処理なのか、を結果データのみからでは適切に把握することが難しいからである。

[0163] そこで、配備サーバ１００は、配備プログラム１１２から、変換後配備プログラム１１３を生成する。配備サーバ１００は、配備プログラム１１２に代えて、変換後配備プログラム１１３を用いて配備を行うことで、変数に対する入力内容の変更やブロックを識別するためのログを配備先サーバ２００に出力させる。そして、結果データが旧結果データと相違する場合には、当該結果データを出力したブロックおよび当該ブロックに用いられる変数をログから特定し、当該変数に対する入力内容の変更の有無を確認することで、当該相違の妥当性を評価し、ユーザに提示する。

[0164] ユーザは、図１７～１９で例示した各ＧＵＩを参照することで、配備プログラム１１２の冪等性を確保するために、見直すべき箇所を容易に把握することができる。例えば、ＧＵＩ５１０で提示された相違箇所は、変数に対する入力内容に応じた妥当な相違箇所である。このため、ユーザは、ＧＵＩ５１０で提示されたブロックの見直しの優先度を下げると判断できる。または

、ユーザは、ＧＵＩ５１０で提示されたブロックの見直しを行わないと判断してもよい。

[0165] また、例えば、ＧＵＩ５２０で提示された相違箇所は、変数に対する入力内容が要因となっていない非妥当な相違箇所である。このため、ユーザは、ＧＵＩ５２０で提示されたブロックの見直しの優先度を上げると判断できる。その際、ＧＵＩ５２０を参照することで、対象のブロックの配備プログラム１１２における記述箇所を容易に把握でき、迅速に作業に着手できる。

[0166] このように、旧結果データと相違する結果データに対して、その妥当性と要因とを出力することで、ユーザによる要因分析の作業や配備プログラム１１２の修正作業の省力化を図れる。これにより、実行結果が相違する要因の適切な把握を支援できる。また、プログラム開発における作業コストの軽減に寄与し得る。

[0167] なお、配備プログラムでは、ブロック自体の記述の変更が行われることもある。配備サーバ１００は当該ブロック自体の記述の変更もシステムコールログ１１５として取得できるように、変換後配備プログラム１１３を生成してもよい。

[0168] 図２０は、配備プログラムの他の例を示す図である。配備プログラム１１２ａは、旧配備プログラム１１１に対する、ユーザによる変更の他の例を示している。旧配備プログラム１１１の９行目の記述が、配備プログラム１１２ａでは、“ｅｘｅｃｕｔｅ “ｐｖｃｒｅａｔｅ #{ＤＩＳＫ}” ｄｏ ｃｏｍｍａｎｄ “ｐｖｃｒｅａｔｅ －－ｕｕｉｄ #{ｎｏｄｅ{‘ｕｕｉｄ’}} #{ＤＩＳＫ}”と変更されている。

[0169] プログラム変換部１２０は、当該ブロック（例えば、ブロックＩＤ“1”のブロック）に変更がある旨をシステムコールログ１１５として取得できるように、変換後配備プログラム１１３を生成してもよい。具体的には、プログラム変換部１２０は、旧配備プログラム１１１および配備プログラム１１２の同じ設定を行う当該ブロック同士（例えば、ディスク設定部分Ｂ１）を比較する。そして、ブロックのコードに相違があることを検出すると、“Ｉ

ｏｇ＿ｗｒｉｔｅ（“ｂｌｏｃｋ　１　ｅｎｔｅｒ”）の記述の直前に、“ｌｏｇ＿ｗｒｉｔｅ（“ｂｌｏｃｋ　ｃｈａｎｇｅｄ”）”といった記述を挿入する。例えば、プログラム変換部１２０によるこの処理を図１４のステップＳ１９に追加することができる。

[0170] すると、配備実行部１３０が、配備先サーバ２００から取得するシステムコールログ１１５には当該ブロックに変更がある旨も記述されることになる。また、検証部１４０により生成されたログテーブル１１６においても、当該ブロックに変更がある旨が登録されることになる。具体的には、上記の例によれば、“ｂｌｏｃｋ　１　ｅｎｔｅｒ”の直前に、“ｂｌｏｃｋ　ｃｈａｎｇｅｄ”が登録され、ブロックＩＤ“１”のブロック自体のコードに変更があった旨をログテーブル１１６から把握できるようになる。例えば、配備実行部１３０および検証部１４０による、これらの処理を図１３のステップＳ３，Ｓ４に追加することができる。

[0171] この場合、検証部１４０は、旧結果データと相違する結果データがある場合に、「当該結果データを出力したブロックで用いられる変数の入力内容に変更があったか否か」に加え、「当該ブロックのコードに変更があったか否か」に基づく妥当性の判断を行える。例えば、冪等性の検証において、旧結果データと相違する結果データが得られる場合、当該相違の妥当性について次のような判断基準を設けることができる。

[0172] （１）当該結果データを出力したブロックにコード変更があり、かつ、当該ブロックで用いられる変数の入力内容に変更がある場合、当該相違は妥当である。

（２）当該結果データを出力したブロックにコード変更があり、かつ、当該ブロックで用いられる変数の入力内容に変更がない場合、当該相違は妥当である。

[0173] （３）当該結果データを出力したブロックにコード変更がなく、かつ、当該ブロックで用いられる変数の入力内容に変更がある場合、当該相違は妥当である。

（４）当該結果データを出力したブロックにコード変更がなく、かつ、当該ブロックで用いられる変数の入力内容に変更がない場合、当該相違は非妥当である。

[0174] この場合、図１５のステップＳ２５の検証部１４０による判定を、ログテーブル１１６（ブロックのコード変更が登録されたもの）に基づいて、着目するブロック（ステップＳ２３で抽出した結果データを出力したブロック）が上記（１）～（４）の何れを満たしているかの判定に置き換える。そして、（１）～（３）に該当する場合、処理をステップＳ２６に進める。（４）に該当する場合、処理をステップＳ２７に進める。ただし、（２）に該当する場合、ステップＳ２６では、変数箇所を要因として出力しなくてもよい（入力内容に変更がない場合だからである）。このようにすれば、ユーザによるプログラムの検証作業を一層省力化することができる。

[0175] 次に、図１３のステップＳ１におけるプログラム変換部１２０の機能を記述したプログラムの例（一部拡張も含む）を説明する。一例として、バッカス・ナウア（ＢＮＦ：Backus - Naur）記法と呼ばれるメタ言語で当該プログラムのアブストラクトシンタクスを記す。

[0176] 図２１は、ＢＮＦ記法で記したプログラムの例を示す図である。以下、図２１に付した行番号により各記述を指し示す（以降の図も同様）。１行目の“Ｐ”はプログラムを表す。“Ｃ”は代入、ｗｈｉｌｅ文、ｉｆ文、ブロックの何れかを表す。２行目の“ｅ”は式を表す。“Ｄ”は配備のためのブロック（例えば、アカウントの設定といった機能毎のブロック）を表す。３行目の“ｏｐ”は等号“＝”や不等号“＜”、“＞”等の比較演算子やプラス記号“＋”やマイナス記号“－”等の基本演算子を表す。“ｖ”は数や文字列等の値を表す。

[0177] 図２２は、プログラム変換用の関数の記述例（その１）を示す図である。ここで、集合演算はプログラムで定義されているものとする。入力内容に変更のあった変数の集合を表す変数ｃｈａｎｇｅｄ＿ｖａｒを最初に宣言し、空集合とする。ｃｈａｎｇｅｄ＿ｖａｒへの要素の追加は、関数ｃｈａｎｇ

ｅｄ＿ｖａｒ．ａｄｄで実行するものとする。ｃｈａｎｇｅｄ＿ｖａｒの中にリストの要素があるかどうかを判定するのは、関数ｃｈａｎｇｅｄ＿ｖａｒ．ｉｎｃｌｕｄｅｄで行うものとする。

[0178] プログラム６１０は、配備プログラム１１２（または配備プログラム１１２ａ。以下同様）の構文から変数を取り出して、変数のリストを作成する関数ｖａｒｓの記述例である。プログラム６２０は、変数の依存関係を示す文字列を生成する関数Ｖの記述例である。ここで、“Ｖ（ｅ）”といった表記は、“Ｖ”という関数が“ｅ”というプログラムを引数にとることを示す。“ｅ”の具体的な値は、“ｘ＝１”だったり、“ｉｆ　ｘ＝０　ｔｈｅｎ　ｙ＝１　ｅｌｓｅ　ｙ＝２”というようなプログラムである。

[0179] 例えば、“Ｕ（ｅ）＝ｃａｓｅ　ｅ　ｗｉｔｈ　ｘ＝１　＝＞　ｘ＝１；ｌｏｇ＿ｗｒｉｔｅ（“ｘ　ｃｈａｎｇｅｄ”）；”という記述があるとすると、これは関数Ｕ（ｅ）の定義である。関数Ｕ（ｅ）は、“ｅ”が“ｘ＝１”という構文のとき、“ｘ＝１；ｌｏｇ＿ｗｒｉｔｅ（“ｘ　ｃｈａｎｇｅｄ”）；”に変換することを意味する。

[0180] 図２３は、プログラム変換用の関数の記述例（その２）を示す図である。プログラム６３０は、配備プログラム１１２を変換する関数Ｔの記述例である。プログラム６３０の関数Ｔは、旧配備プログラム１１１と配備プログラム１１２との間で、代入を行う部分が変更されているときと、各ブロック自体のコードが変更されているときに、変更を示すログを出力するように配備プログラム１１２を変換する。関数ｃｈａｎｇｅｃｈｅｃｋは、“Ｃ”が変更された部分であるか否かを確認する関数である。

[0181] 関数Ｓは、変数に対する入力が変更された可能性が高いとみなせる箇所を検出して、当該箇所について変更を示すログを出力するように配備プログラム１１２を変換するための関数である。変数に対する入力が変更された可能性が高いとみなせる箇所とは、具体的には、ｗｈｉｌｅ文やｉｆ文を用いて変数が定義されている箇所である。ｉｆ文やｗｈｉｌｅ文では、条件が変わることで変数への代入値が変わる可能性が高いからである。

[0182] また、２０行目には、コードの変更があったブロックに対して“ｌｏｇ＿ｗｒｉｔｅ（“ｂｌｏｃｋ　ｃｈａｎｇｅｄ；”）”を挿入する処理も記述されている。

図２４は、プログラム変換用の関数の記述例（その３）を示す図である。プログラム６４０は、上記関数Ｓの記述例である。関数Ｓは、ｉｆ文やｗｈｉｌｅ文の条件式が変わって、変数に対する代入箇所が変更される可能性があるときに、代入値の変更に関わりなく、代入値の変更の可能性が記録されるようにするものである。すなわち、変数に対する代入文は入力値や式それ自体に変更がなくても、条件によって代入値が変更される可能性がある。

[0183] 例えば、ｗｈｉｌｅ文では、繰り返す回数が異なることで、変数への代入値が変化することがある。また、ｉｆ文では、条件式に対する入力が変わることで、実行される節がｔｈｅｎ節からｅｌｓｅ節に変わり、実行される変数への代入文が変化することがある。このため、ｗｈｉｌｅ文やｉｆ文の中で定義されている変数については、変更された可能性が高いと考えて、入力内容に変更がある（変更された可能性がある）旨をログに出力するようにする。また、代入文が明らかに変わっている場合や入力が変わった場合には、その旨をログに出力することで、単なる可能性と区別できるようにする。

[0184] 図２５は、配備プログラムの他の変換例を示す図である。配備プログラム１１２ｂでは、変数ＤＩＳＫに対する代入式がｉｆ文の中に記述されている場合を例示している。この場合、ｉｆ文の条件式に対する入力に応じて、変数ＤＩＳＫへの代入値が変更されることになる。具体的には、ｉｆ文の条件式に応じて、ｔｈｅｎ節の“ＤＩＳＫ＝“／ｄｅｖ／ｈｄａ０””が実行されたり、ｅｌｓｅ節の“ＤＩＳＫ＝“／ｄｅｖ／ｈｄａ１””が実行されたりする。

[0185] 変換後配備プログラム１１３ａは、関数Ｔを用いて配備プログラム１１２ｂが変換された結果を例示している。例えば、関数Ｔ，Ｓによれば、（旧配備プログラム１１１と比べて）変数ＤＩＳＫに対する代入値が変更されているか否かに関わらず、ｔｈｅｎ節およびｅｌｓｅ節における変数ＤＩＳＫの

代入文の直後に“ｌｏｇ＿ｗｒｉｔｅ（“ＤＩＳＫ　ｃｈａｎｇｅｄ；”）”をそれぞれ挿入する。挿入された記述は、変換後配備プログラム１１３ａの５，９行目に相当する。プログラム変換部１２０は、図１３のステップＳ１において、このような変換を行ってもよい。これにより、入力内容が変更される可能性の高い変数ＤＩＳＫについて、その旨をシステムコールログ１１５に記録できる。

[0186] 図２６は、ログの出力フォーマットの例を示す図である。ログフォーマット６５０は、ログの文法をＢＮＦ記法で示したものである。ログは、起こった順に並ぶリストである。３行目の“Ｘ”はプログラムの変数を表す。また、“ＩＤ”はブロックＩＤを表す。

[0187] ２行目の記述“ｆｏｌｌｏｗｉｎｇ　ｂｌｏｃｋｓ　ｃａｎ　ｂｅ　ｃｈａｎｇｅｄ；　ＬＳ　ｐｒｅｖ　ｂｌｏｃｋｓ　ｃａｎ　ｂｅ　ｃｈａｎｇｅｄ；”は、“ＬＳ”の部分の定義が変更され得る可能性があることを示している。

[0188] ３行目の記述“Ｘ　ｃｈａｎｇｅｄ　ＡＤＤ＿ＬＩＳＴ”は、変数Ｘに対する入力内容が変更されたことを示している。その変更に際し、“ＡＤＤ＿ＬＩＳＴ”が変更された入力（依存先の変数）であったことを示す。例えば、“ＡＤＤ＿ＬＩＳＴ”が“ｕｓｅ　Ｙ；ｕｓｅ　Ｚ；”ならば、変数Ｘに代入する値に“Ｙ＋Ｚ”のように変数Ｙ，Ｚが用いられている（変数Ｘは変数Ｙ，Ｚに依存している）ことを示す。

[0189] ３行目の記述“ｂｌｏｃｋ　ＩＤ　ｅｎｔｅｒ”は、“ＩＤ”で示されるブロックの処理の開始を示す。“ｂｌｏｃｋ　ｃｈａｎｇｅｄ；ｂｌｏｃｋ　ＩＤ　ｅｎｔｅｒ”は、“ＩＤ”で示されるブロックのコードが変更されていること、および、当該ブロックの処理の開始を示す。“ｂｌｏｃｋ　ＩＤ　ｅｘｉｔ”は、“ＩＤ”で示されるブロックの処理の終了を示す。

[0190] 配備サーバ１００は、図２２～２４で例示した各関数を用いて、変換後配備プログラムを生成できる。配備サーバ１００は、当該変換後配備プログラムを用いて配備を行うことで、配備先サーバ２００によりログフォーマット

６５０で示されるログを含むシステムコールログ１１５を出力させ、配備プログラムの検証に利用できる。システムコールログ１１５（およびログテーブル１１６）による検証方法は、前述の通りである。

[0191] なお、第１の実施の形態の情報処理は、演算部１ｂにプログラムを実行させることで実現できる。また、第２の実施の形態の情報処理は、プロセッサ１０１にプログラムを実行させることで実現できる。プログラムは、コンピュータ読み取り可能な記録媒体（例えば、光ディスク１３、メモリ装置１４およびメモリカード１６等）に記録できる。

[0192] 例えば、プログラムを記録した記録媒体を配布することで、プログラムを流通させることができる。また、プログラムを他のコンピュータに格納しておき、ネットワーク経由でプログラムを配布してもよい。コンピュータは、例えば、記録媒体に記録されたプログラムまたは他のコンピュータから受信したプログラムを、ＲＡＭ１０２やＨＤＤ１０３等の記憶装置に格納し（インストールし）、当該記憶装置からプログラムを読み込んで実行してもよい。

[0193] 上記については単に本発明の原理を示すものである。更に、多数の変形や変更が当業者にとって可能であり、本発明は上記に示し、説明した正確な構成および応用例に限定されるものではなく、対応する全ての変形例および均等物は、添付の請求項およびその均等物による本発明の範囲とみなされる。

符号の説明

[0194] １　検証装置

１ａ　記憶部

１ｂ　演算部

２　プログラム

２ａ，２ｂ，２ｃ　プログラム要素

３　ログ

請求の範囲

[請求項1] 変数に応じた処理を示す複数のプログラム要素を含むプログラムの検証方法であって、コンピュータが、

プログラムの実行時に、過去に当該プログラムを実行したときに対して入力内容が変更された変数を識別する情報、および、プログラム要素と当該プログラム要素が実行されることで出力される実行結果との対応を示す情報を含むログを取得し、

プログラムの何れかの実行結果が過去の実行結果と相違するとき、前記ログを参照して、当該実行結果に対応するプログラム要素および当該プログラム要素に用いられる変数を検索し、検索した変数に対する入力内容の変更状況に基づいて、実行結果が相違する要因と当該相違の妥当性とを評価し、

実行結果の相違する箇所と相違の要因と相違の妥当性とを示す情報を出力する、

検証方法。

[請求項2] 前記評価では、検索した変数に対する入力内容の変更がある場合、当該変数に対する入力内容の変更と当該変数を用いるプログラム要素とが相違の要因であり、当該相違が妥当であると評価し、また、検索した変数に対する入力内容に変更がない場合、当該変数を用いるプログラム要素が相違の要因であり、当該相違が妥当でないと評価する、請求項１記載の検証方法。

[請求項3] 前記取得では、前記ログとして、過去にプログラムを実行したときとプログラム要素の記述が変更されたプログラム要素を識別する情報を更に取得し、

前記評価では、前記ログを参照して、過去の実行結果と相違する実行結果に対応するプログラム要素の記述の変更状況に基づいて、実行結果が相違する要因と当該相違の妥当性とを評価する、請求項１または２記載の検証方法。

[請求項4] 前記評価では、検索した変数に対する入力内容の変更がなく、かつプログラム要素の記述に変更がある場合、当該プログラム要素が相違の要因であり、当該相違が妥当であると評価し、また、検索した変数に対する入力内容の変更がなく、かつプログラム要素の記述に変更がない場合、当該プログラム要素が相違の要因であり、当該相違が妥当でないと評価する、請求項３記載の検証方法。

[請求項5] プログラムの実行中に前記ログに含まれる情報を出力するコードを、当該プログラムの実行前に、当該プログラムに挿入する、請求項１乃至４の何れか１項に記載の検証方法。

[請求項6] 前記挿入では、過去にプログラムを実行したときの各変数の入力内容と、今回の各変数の入力内容とを比較することで、入力内容が変更された変数を特定し、当該変数を識別する情報を出力するコードを挿入する、請求項５記載の検証方法。

[請求項7] 前記挿入では、第１の変数により依存される第２の変数を特定し、前記第２の変数の入力内容が過去の入力内容に対して変更されているか否かに応じて、前記第１の変数の入力内容が変更されているか否かを判定する、請求項６記載の検証方法。

[請求項8] 前記挿入では、各変数の入力内容を定めた情報を参照して、条件文の中の複数の節それぞれで入力内容が定められた変数を特定し、特定した変数を入力内容が変更された変数として識別する情報を出力するコードを挿入する、請求項５乃至７の何れか１項に記載の検証方法。

[請求項9] 変数に応じた処理を示す複数のプログラム要素を含むプログラムの検証に用いられる検証装置であって、

プログラムの実行状況を示すログを記憶する記憶部と、

プログラムの実行時に、過去に当該プログラムを実行したときに対して入力内容が変更された変数を識別する情報、および、プログラム要素と当該プログラム要素が実行されることで出力される実行結果との対応を示す情報を含む前記ログを取得し、

プログラムの何れかの実行結果が過去の実行結果と相違するとき、前記ログを参照して、当該実行結果に対応するプログラム要素および当該プログラム要素に用いられる変数を検索し、当該変数に対する入力内容の変更状況に基づいて、実行結果が相違する要因と当該相違の妥当性とを評価し、

実行結果の相違する箇所と相違の要因と相違の妥当性とを示す情報を出力する、演算部と、

を有する検証装置。

[請求項10] 変数に応じた処理を示す複数のプログラム要素を含むプログラムの検証に用いられる検証プログラムであって、コンピュータに、

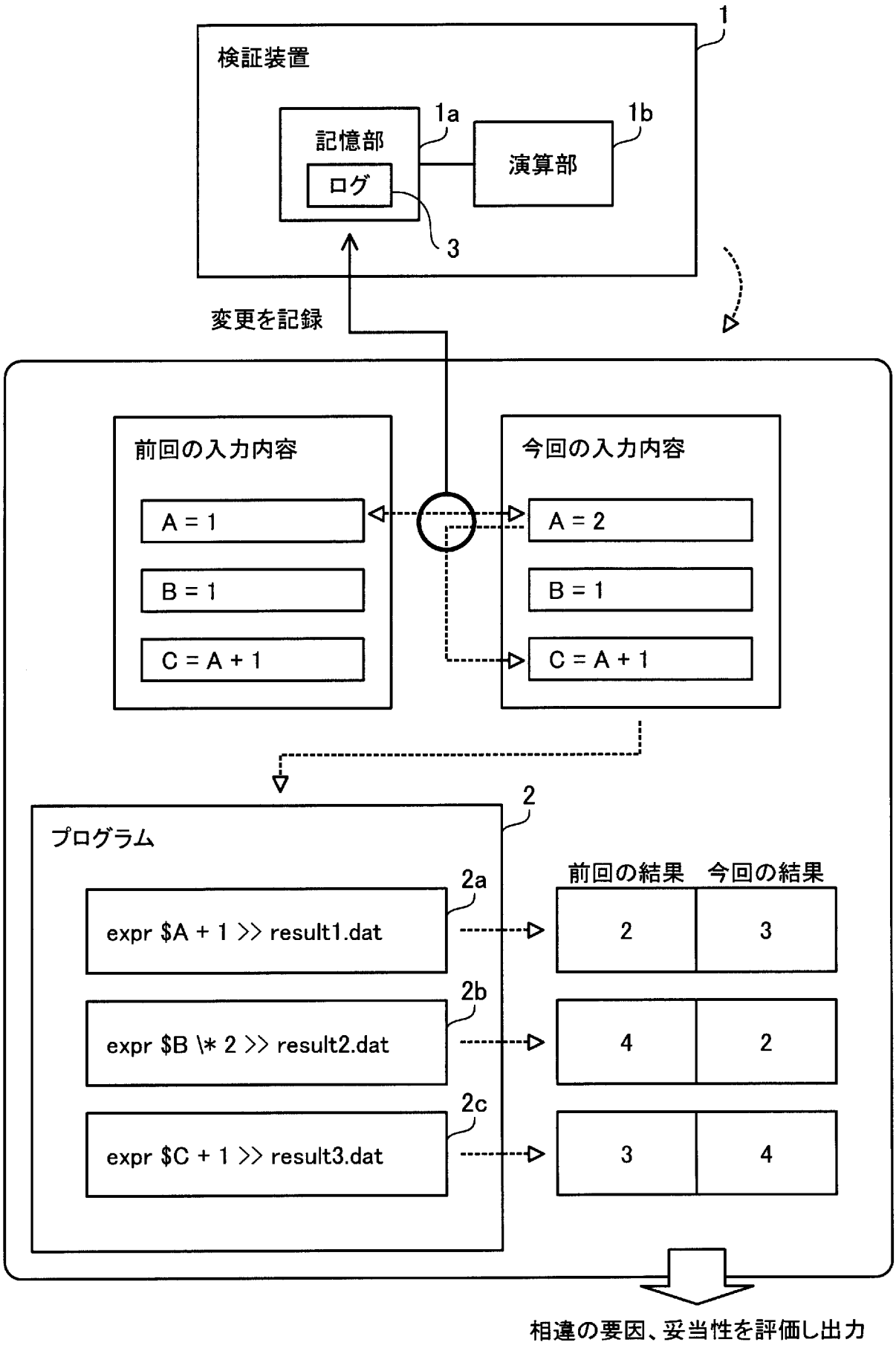
プログラムの実行時に、過去に当該プログラムを実行したときに対して入力内容が変更された変数を識別する情報、および、プログラム要素と当該プログラム要素が実行されることで出力される実行結果との対応を示す情報を含むログを取得し、

プログラムの何れかの実行結果が過去の実行結果と相違するとき、前記ログを参照して、当該実行結果に対応するプログラム要素および当該プログラム要素に用いられる変数を検索し、当該変数に対する入力内容の変更状況に基づいて、実行結果が相違する要因と当該相違の妥当性とを評価し、

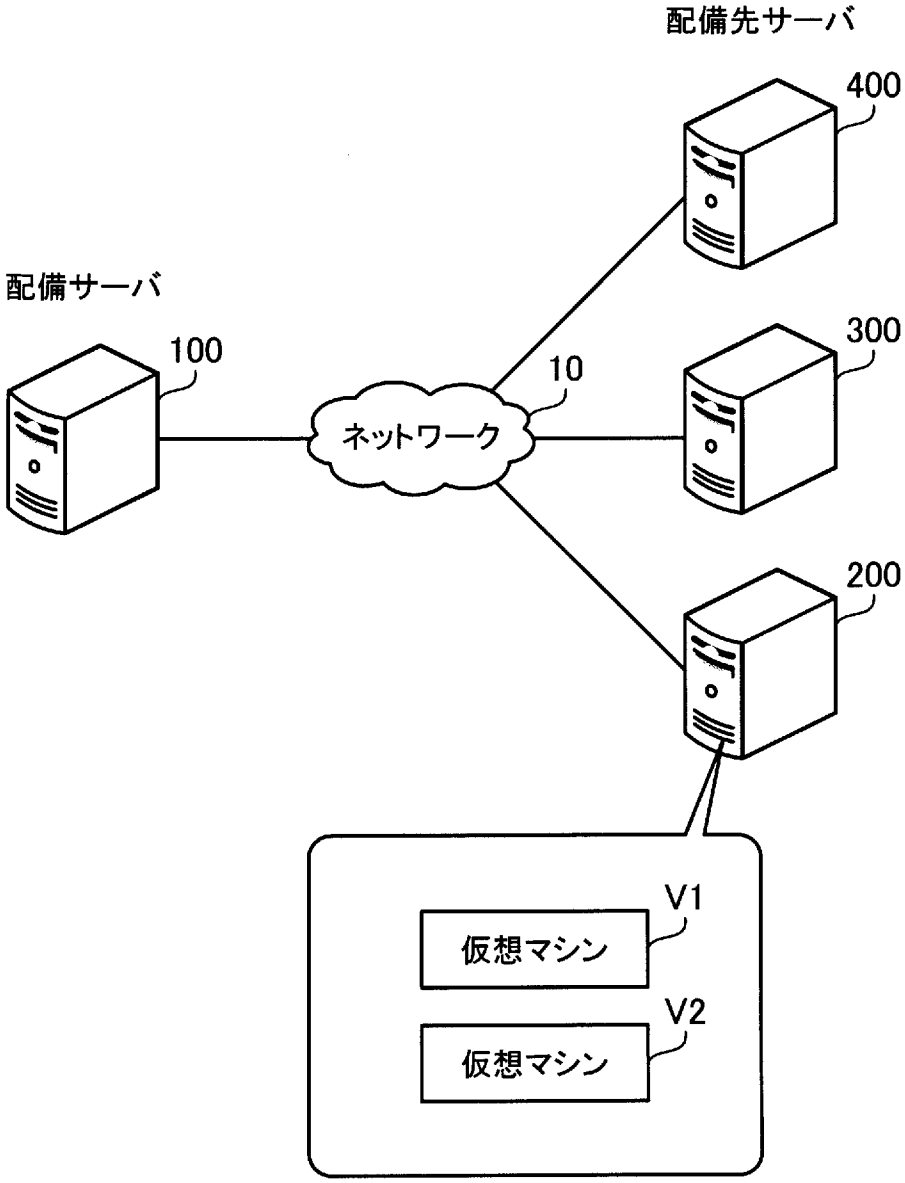
実行結果の相違する箇所と相違の要因と相違の妥当性とを示す情報を出力する、

処理を実行させる検証プログラム。

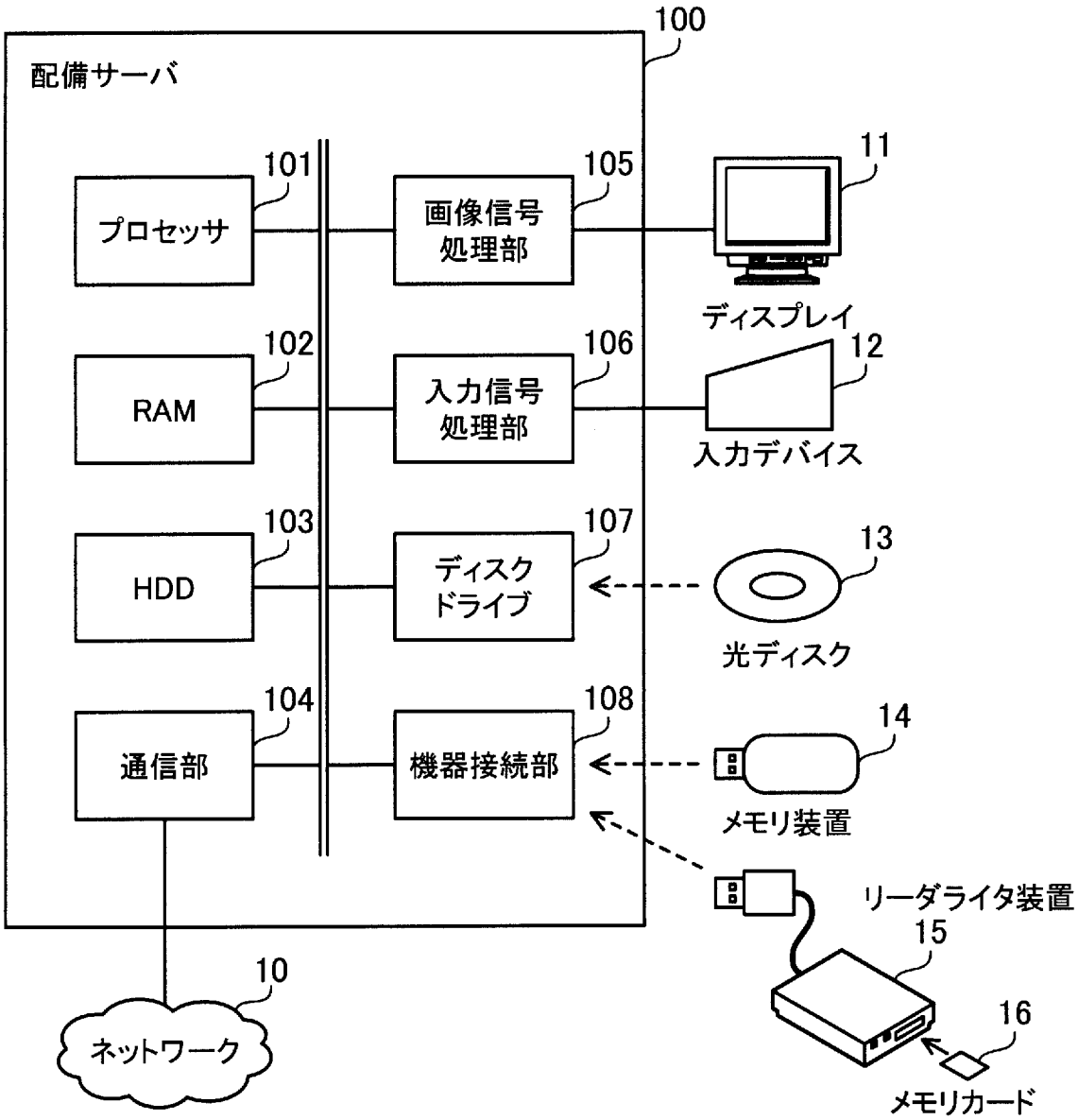
[図1]


1
検証装置
記憶部
1a
ログ
3
演算部
1b
変更を記録
前回の入力内容
A = 1
B = 1
C = A + 1
今回の入力内容
A = 2
B = 1
C = A + 1
2
プログラム
2a
expr $A + 1 >> result1.dat
2b
expr $B * 2 >> result2.dat
2c
expr $C + 1 >> result3.dat
前回の結果
今回の結果
2
3
4
2
3
4
相違の要因、妥当性を評価し出力

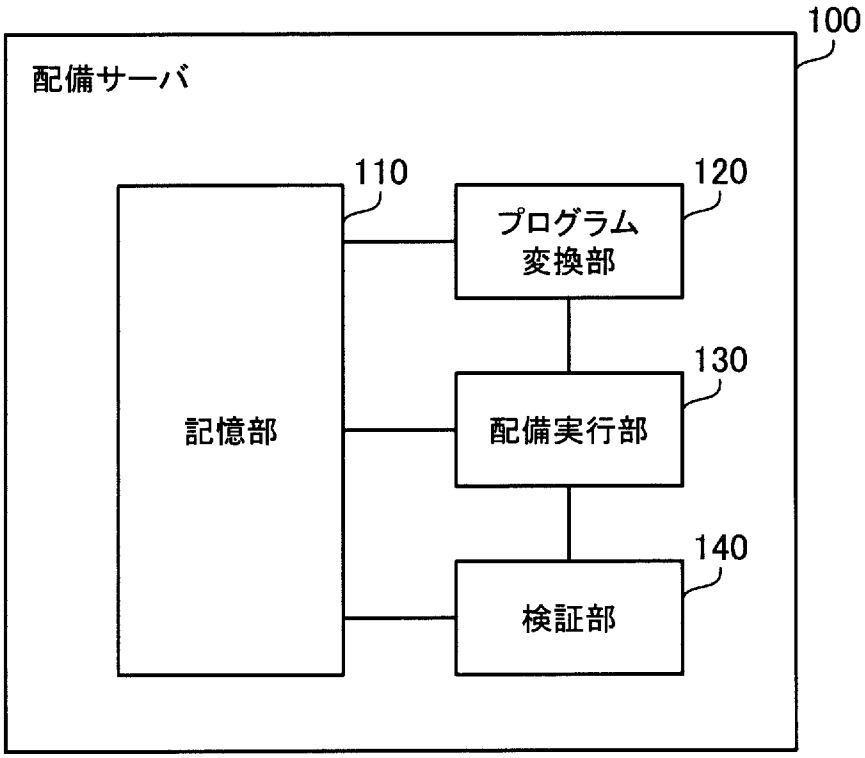
[図2]



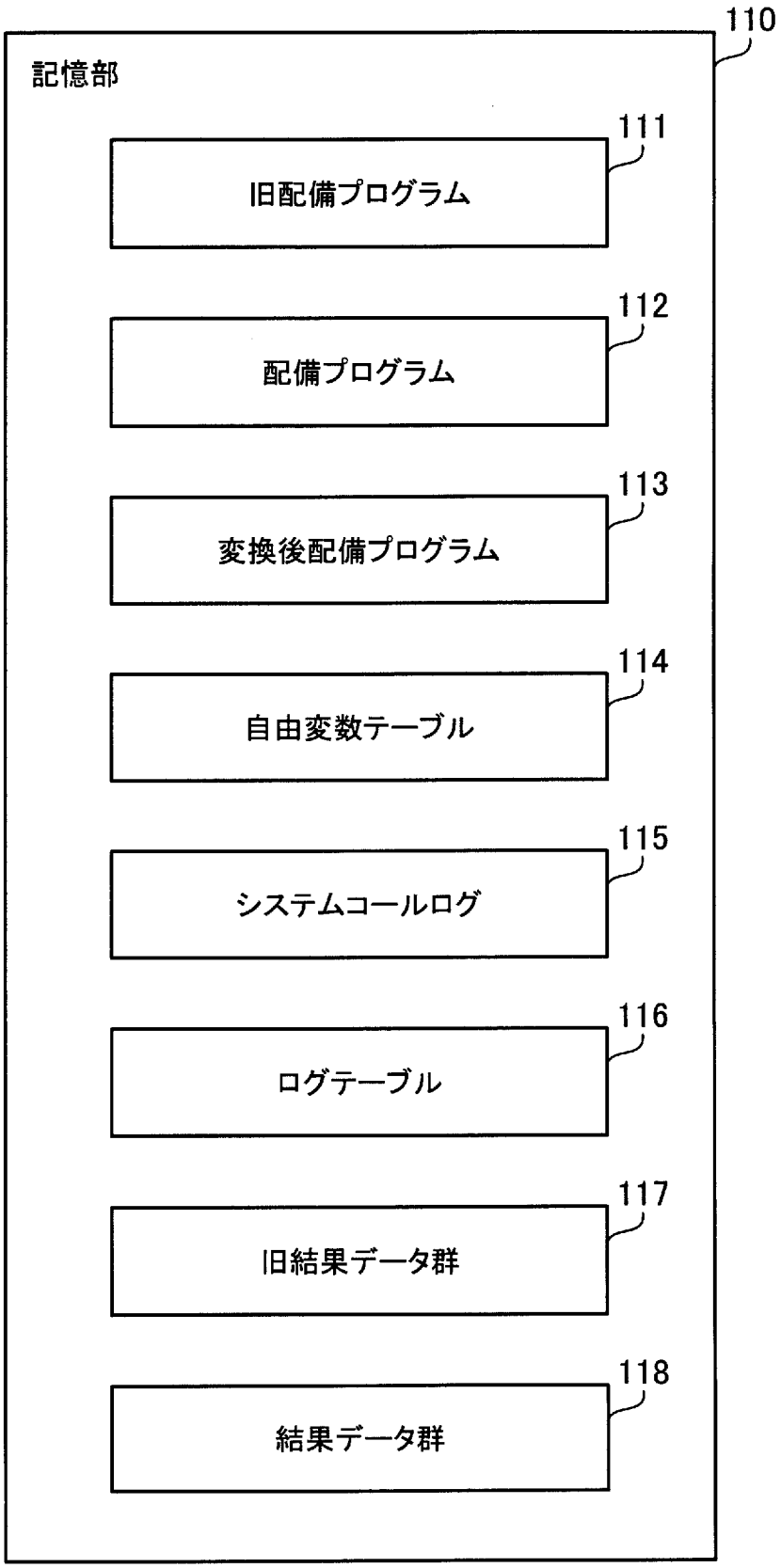
[図3]


100
配備サーバ
101
プロセッサ
102
RAM
103
HDD
104
通信部
105
画像信号処理部
106
入力信号処理部
107
ディスクドライブ
108
機器接続部
11
ディスプレイ
12
入力デバイス
13
光ディスク
14
メモリ装置
リーダライタ装置
15
16
メモリカード
10
ネットワーク

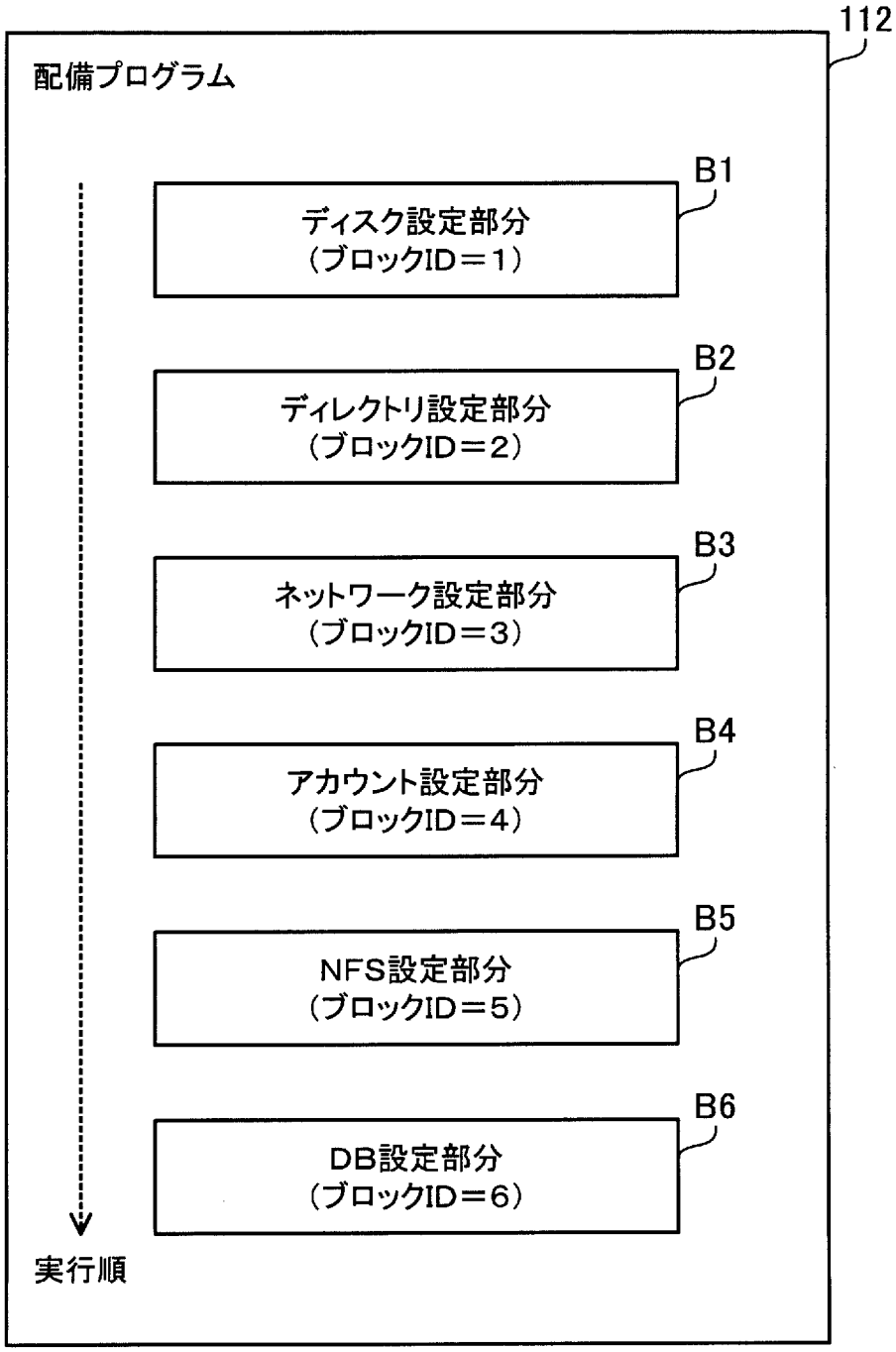
[図4]



[図5]


110
記憶部
111
旧配備プログラム
112
配備プログラム
113
変換後配備プログラム
114
自由変数テーブル
115
システムコールログ
116
ログテーブル
117
旧結果データ群
118
結果データ群

[図6]


112
配備プログラム
B1
ディスク設定部分
（ブロックID＝1）
B2
ディレクトリ設定部分
（ブロックID＝2）
B3
ネットワーク設定部分
（ブロックID＝3）
B4
アカウント設定部分
（ブロックID＝4）
B5
NFS設定部分
（ブロックID＝5）
B6
DB設定部分
（ブロックID＝6）
実行順

[図7]

旧配備プログラムの例

111

```
1  …
2  DISK = “/dev/hda1”
3  …
4  W = “pochi”
5  X = “192.168.0.1”
6  Y = “taro”
7  Z = f(X)
8  …
9  execute “pvcreate #{DISK}” do command “pvcreate #{DISK}”
10 end
11 …
```

配備プログラムの例

112

```
1  …
2  DISK=“/dev/hda0”
3  …
4  W = “pochi”
5  X = “192.168.10.1”
6  Y = “jiro”
7  Z = f(X)
8  …
9  execute “pvcreate #{DISK}” do command “pvcreate #{DISK}”
10 end
11 …
```

[図8]

変換後配備プログラムの例

113

```
1  …
2  DISK = “/dev/hda0”
3  log_write ( “DISK changed” )
4  …
5  W = “pochi”
6  X = “192.168.10.1”
7  log_write ( “X changed” )
8  Y = “jiro”
9  log_write ( “Y changed” )
10 Z = f(X)
11 log_write ( “Z changed; use X” )
12 …
13 log_write ( “block 1 enter” )
14 execute “pvcreate #{DISK}” do command “pvcreate #{DISK}”
15 end
16 log_write ( “block 1 exit” )
17 …
```

[図9]

114

自由変数テーブル	
ブロックID	自由変数集合
1	DISK
2	—
3	W,X
4	Y
5	Z
6	Y

[図10]

システムコールログの例

115

```
1  …
2  open f
3  write f “DISK changed”
4  close f
5  …
6  open f
7  write f “X changed”
8  close f
9
10 open f
11 write f “Y changed”
12 close f
13
14 open f
15 write f “Z changed; use X”
16 close f
17 …
18 open f
19 write f “block 1 enter”
20 close f
21
22 open /var/log/messages
23 write /var/log/messages “/dev/hda0 created”
24 close /var/log/messages
25
26 open f
27 write f “block 1 exit”
28 close f
29 …
```

[図11]

システムコールログの例（続き）

115

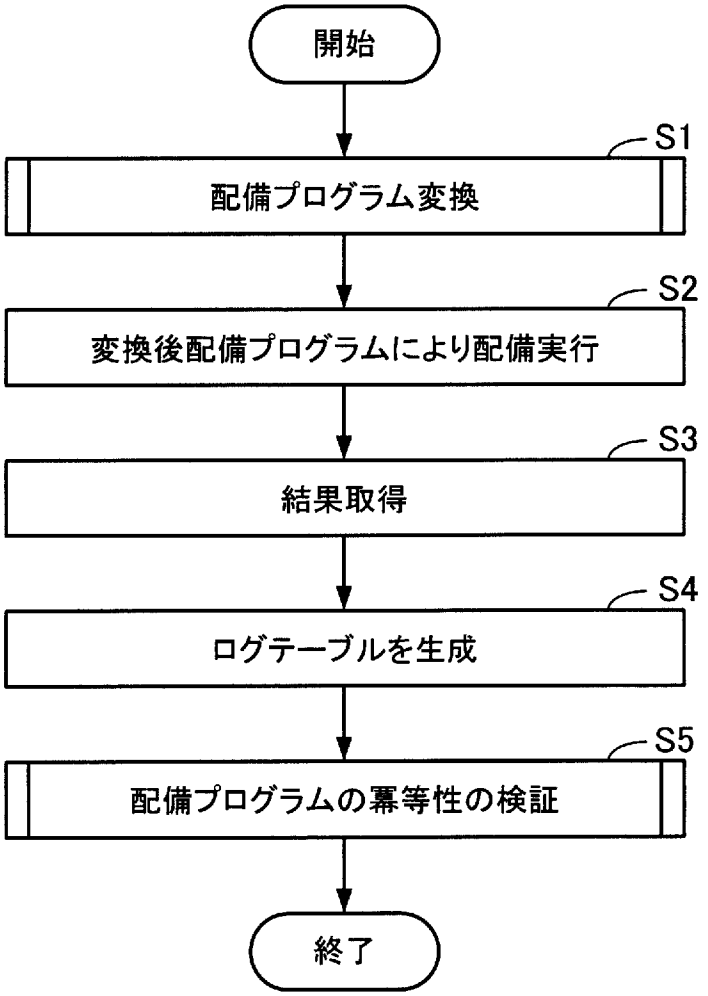
```
30  …
31  open f
32  write f “block 3 enter”
33  close f
34
35  open /etc/hosts
36  write /etc/hosts “127.0.0.1 localhost”
37  write /etc/hosts “192.168.10.1 pochi”
38  close /etc/hosts
39
40  open f
41  write f “block 3 exit”
42  close f
43
44  open f
45  write f “block 4 enter”
46  close f
47
48  open /etc/passwd
49  write /etc/passwd “jiro …”
50  close /etc/passwd
51
52  open f
53  write f “block 4 exit”
54  close f
55
56  open f
57  write f “block 5 enter”
58  close f
59
60  open /etc/exports
61  write /etc/exports “/home/nfs 192.168.10.2/24 (rw)”
62  close /etc/exports
63
64  open f
65  write f “block 5 exit”
66  close f
67  …
```

[図12]

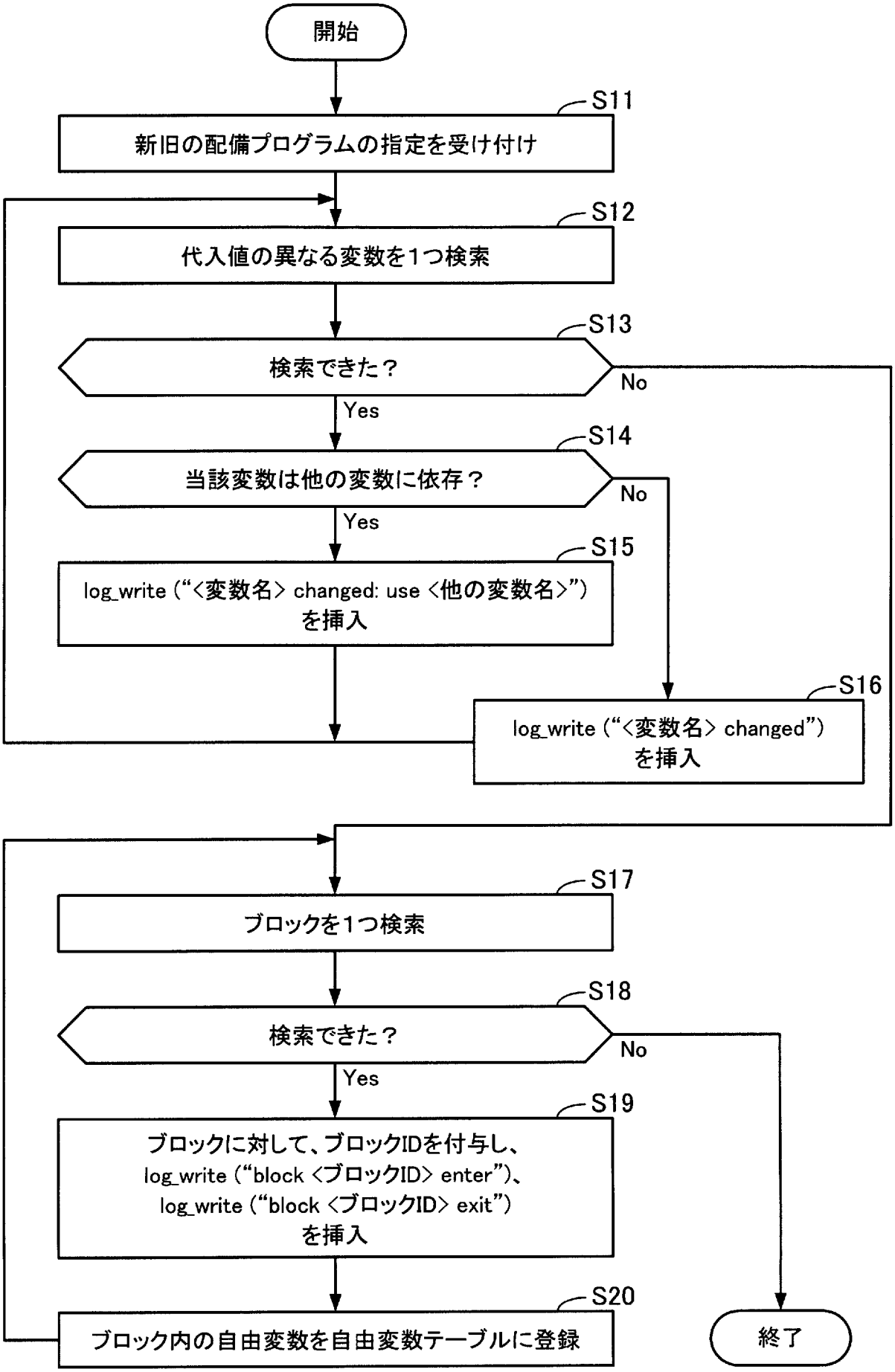
116

ログテーブル	
ブロックID	ログ
—	DISK changed
—	X changed
—	Y changed
—	Z changed; use X
1	block 1 enter
1	write /var/log/messages “/dev/hda0 created”
1	block 1 exit
2	block 2 enter
・・・	・・・
3	block 3 enter
3	open /etc/hosts
3	write /etc/hosts “127.0.0.1 localhost”
3	write /etc/hosts “192.168.10.1 pochi”
3	close/etc/hosts
3	block 3 exit
4	block 4 enter
4	open /etc/passwd
4	write /etc/passwd “jiro ...”
4	close /etc/passwd
4	block 4 exit
5	block 5 enter
5	open /etc/exports
5	write /etc/exports “/home/nfs 192.168.10.2/24 (rw)”
5	close /etc/exports
5	block 5 exit
6	block 6 enter
・・・	・・・

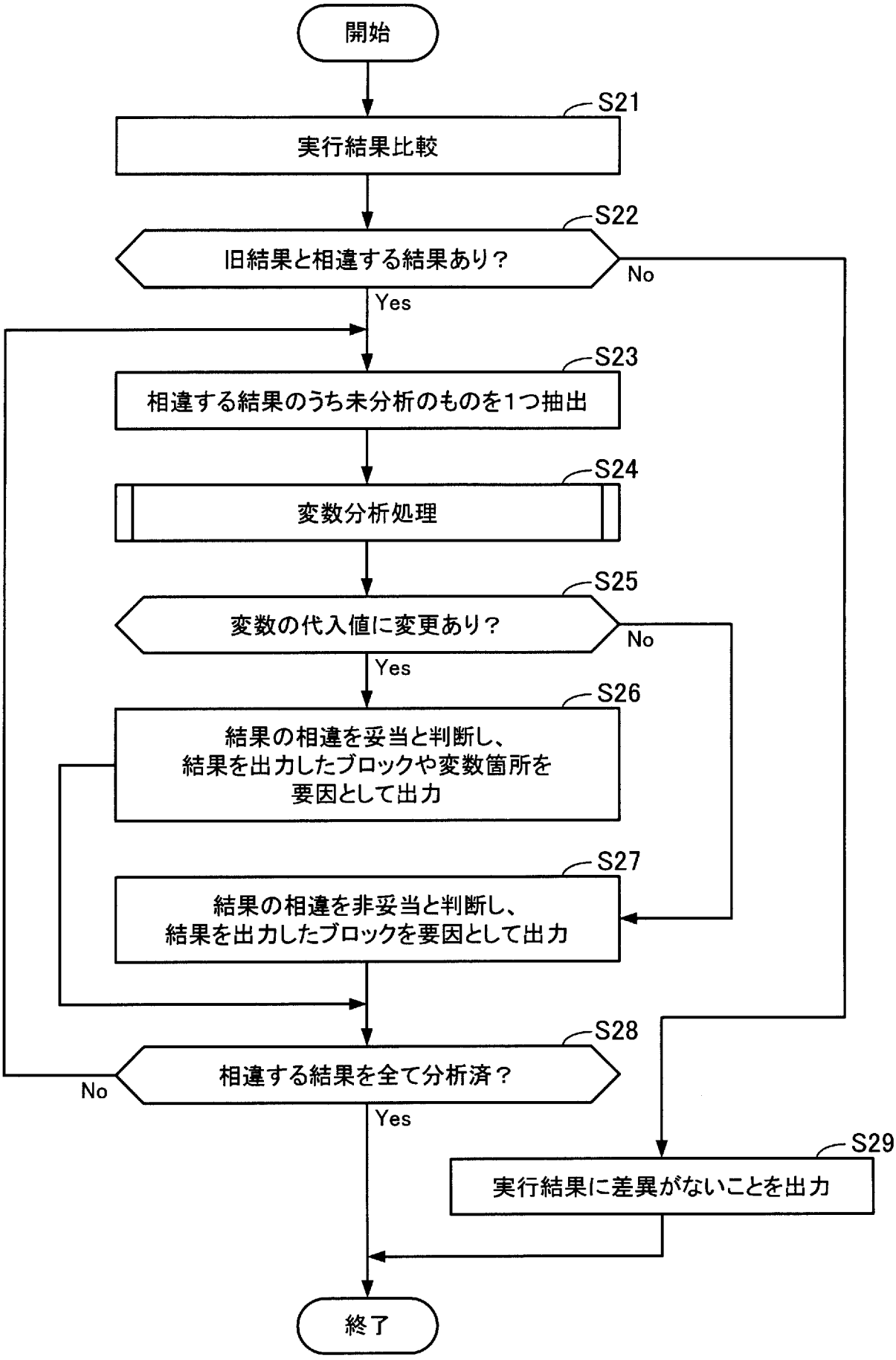
[図13]


開始
S1
配備プログラム変換
S2
変換後配備プログラムにより配備実行
S3
結果取得
S4
ログテーブルを生成
S5
配備プログラムの冪等性の検証
終了

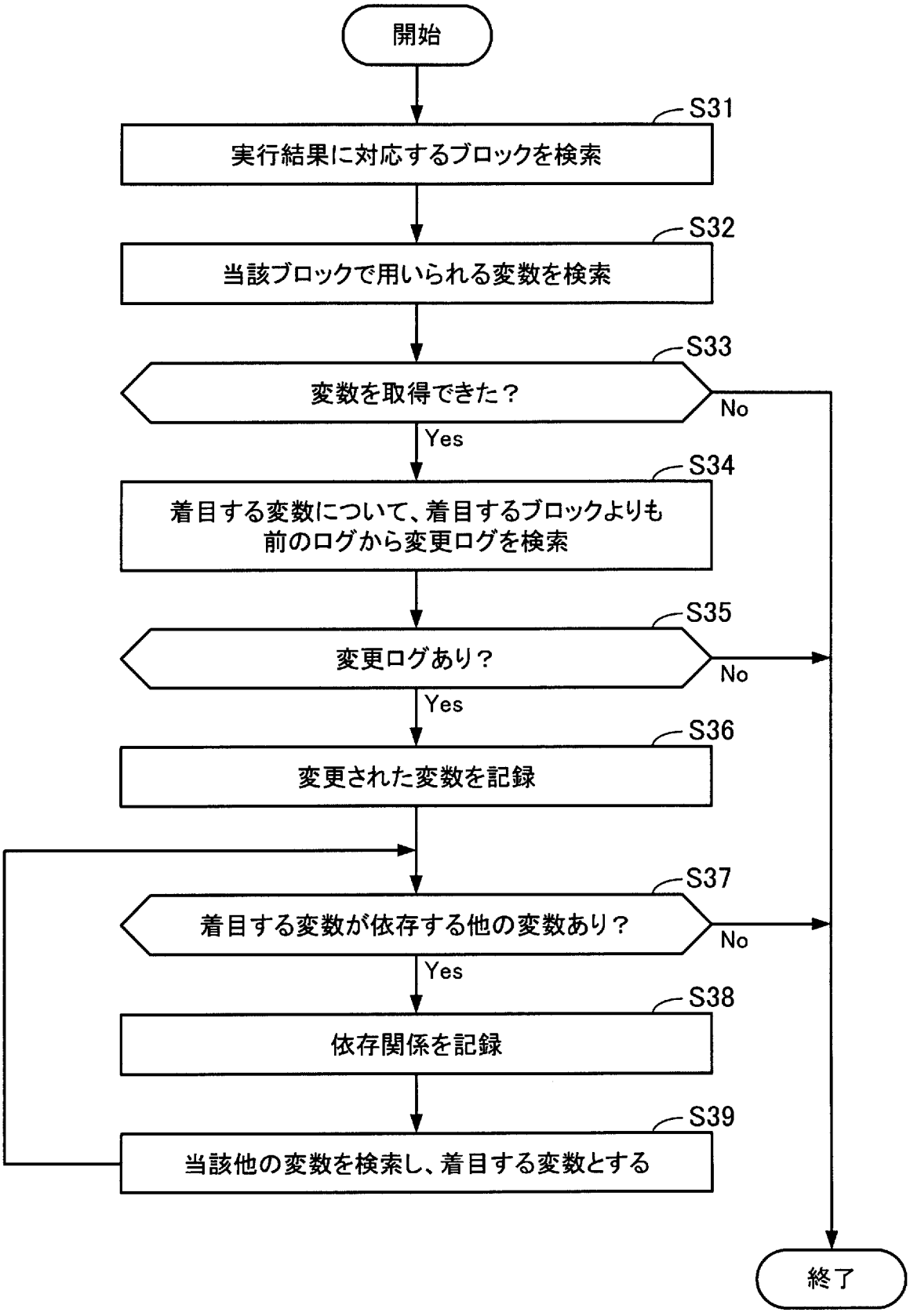
[図14]


開始
S11
新旧の配備プログラムの指定を受け付け
S12
代入値の異なる変数を1つ検索
S13
検索できた？
No
Yes
S14
当該変数は他の変数に依存？
No
Yes
S15
log_write (“<変数名> changed: use <他の変数名>”)
を挿入
S16
log_write (“<変数名> changed”)
を挿入
S17
ブロックを1つ検索
S18
検索できた？
No
Yes
S19
ブロックに対して、ブロックIDを付与し、
log_write (“block <ブロックID> enter”)、
log_write (“block <ブロックID> exit”)
を挿入
S20
ブロック内の自由変数を自由変数テーブルに登録
終了

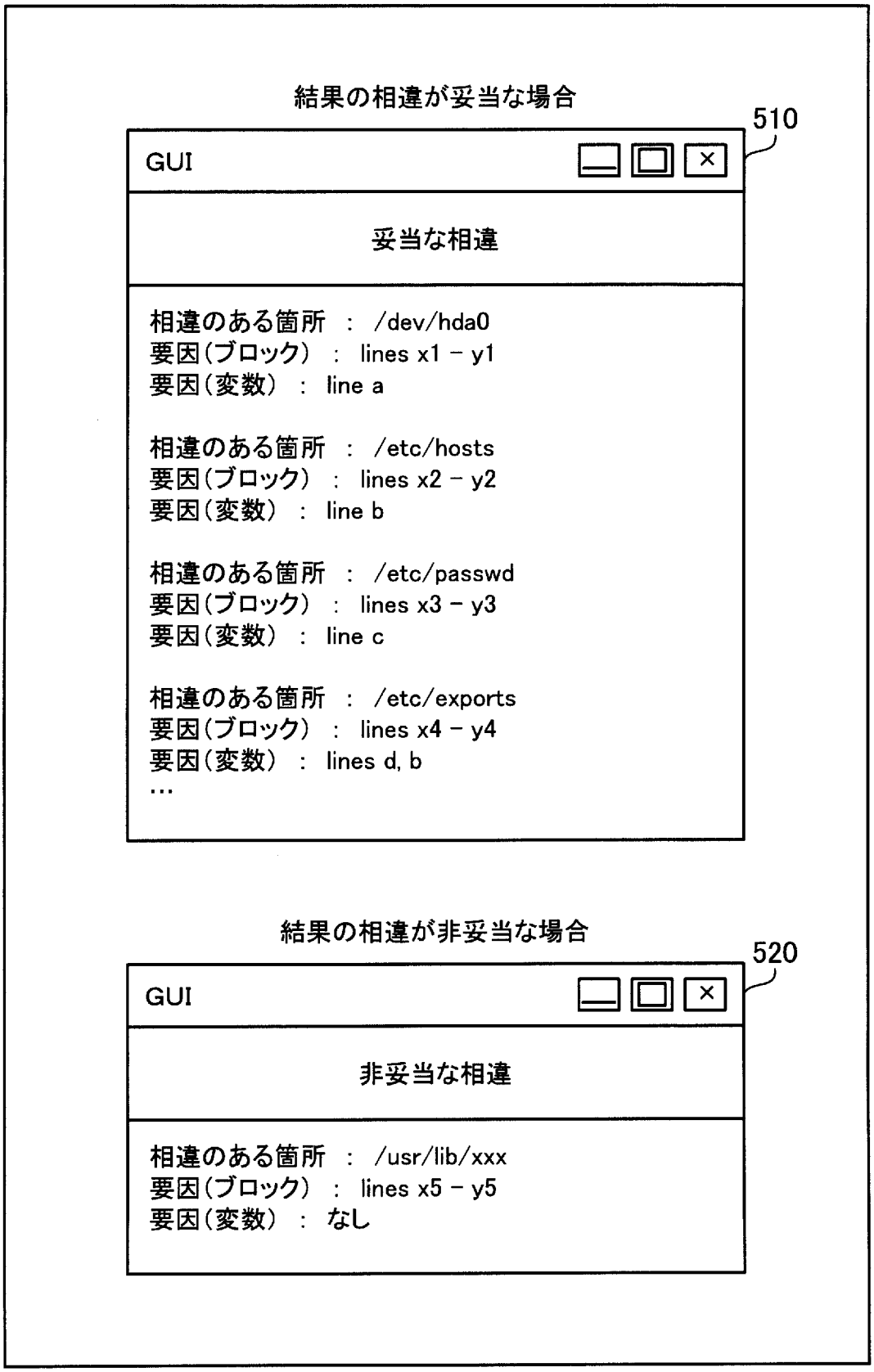
[図15]


開始
S21
実行結果比較
S22
旧結果と相違する結果あり？
No
Yes
S23
相違する結果のうち未分析のものを1つ抽出
S24
変数分析処理
S25
変数の代入値に変更あり？
No
Yes
S26
結果の相違を妥当と判断し、
結果を出力したブロックや変数箇所を
要因として出力
S27
結果の相違を非妥当と判断し、
結果を出力したブロックを要因として出力
S28
相違する結果を全て分析済？
No
Yes
S29
実行結果に差異がないことを出力
終了

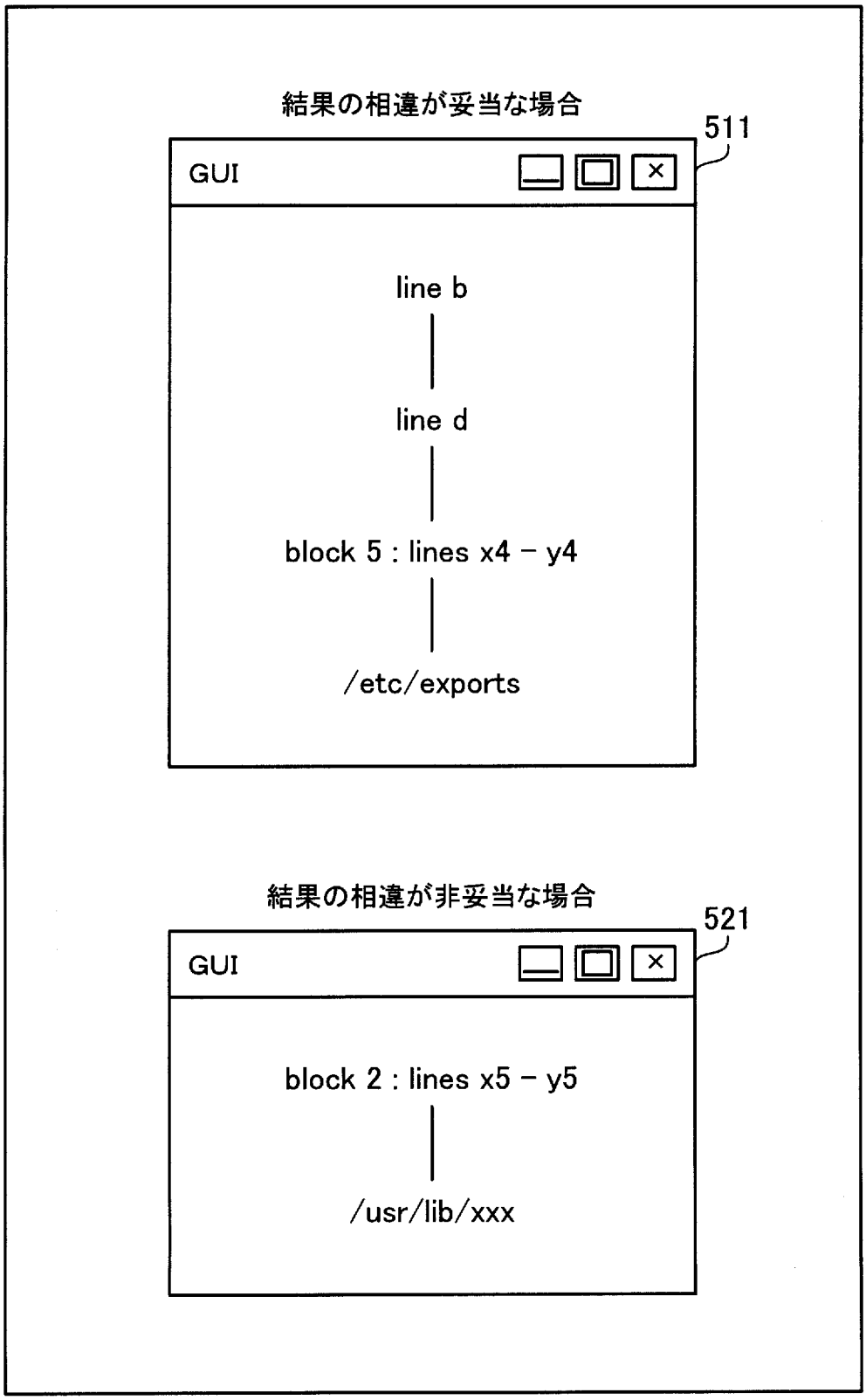
[図16]


開始
S31
実行結果に対応するブロックを検索
S32
当該ブロックで用いられる変数を検索
S33
変数を取得できた？
No
Yes
S34
着目する変数について、着目するブロックよりも前のログから変更ログを検索
S35
変更ログあり？
No
Yes
S36
変更された変数を記録
S37
着目する変数が依存する他の変数あり？
No
Yes
S38
依存関係を記録
S39
当該他の変数を検索し、着目する変数とする
終了

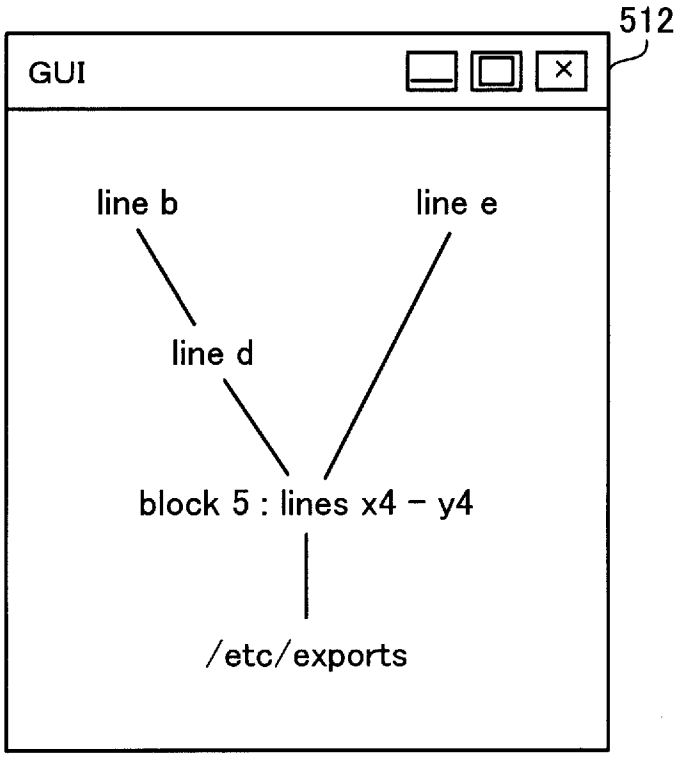
[図17]


結果の相違が妥当な場合
510
GUI
妥当な相違
相違のある箇所 : /dev/hda0
要因(ブロック) : lines x1 - y1
要因(変数) : line a
相違のある箇所 : /etc/hosts
要因(ブロック) : lines x2 - y2
要因(変数) : line b
相違のある箇所 : /etc/passwd
要因(ブロック) : lines x3 - y3
要因(変数) : line c
相違のある箇所 : /etc/exports
要因(ブロック) : lines x4 - y4
要因(変数) : lines d, b
…
結果の相違が非妥当な場合
520
GUI
非妥当な相違
相違のある箇所 : /usr/lib/xxx
要因(ブロック) : lines x5 - y5
要因(変数) : なし

[図18]


結果の相違が妥当な場合
511
GUI
line b
line d
block 5 : lines x4 - y4
/etc/exports
結果の相違が非妥当な場合
521
GUI
block 2 : lines x5 - y5
/usr/lib/xxx

[図19]



[図20]

配備プログラムの他の例

112a

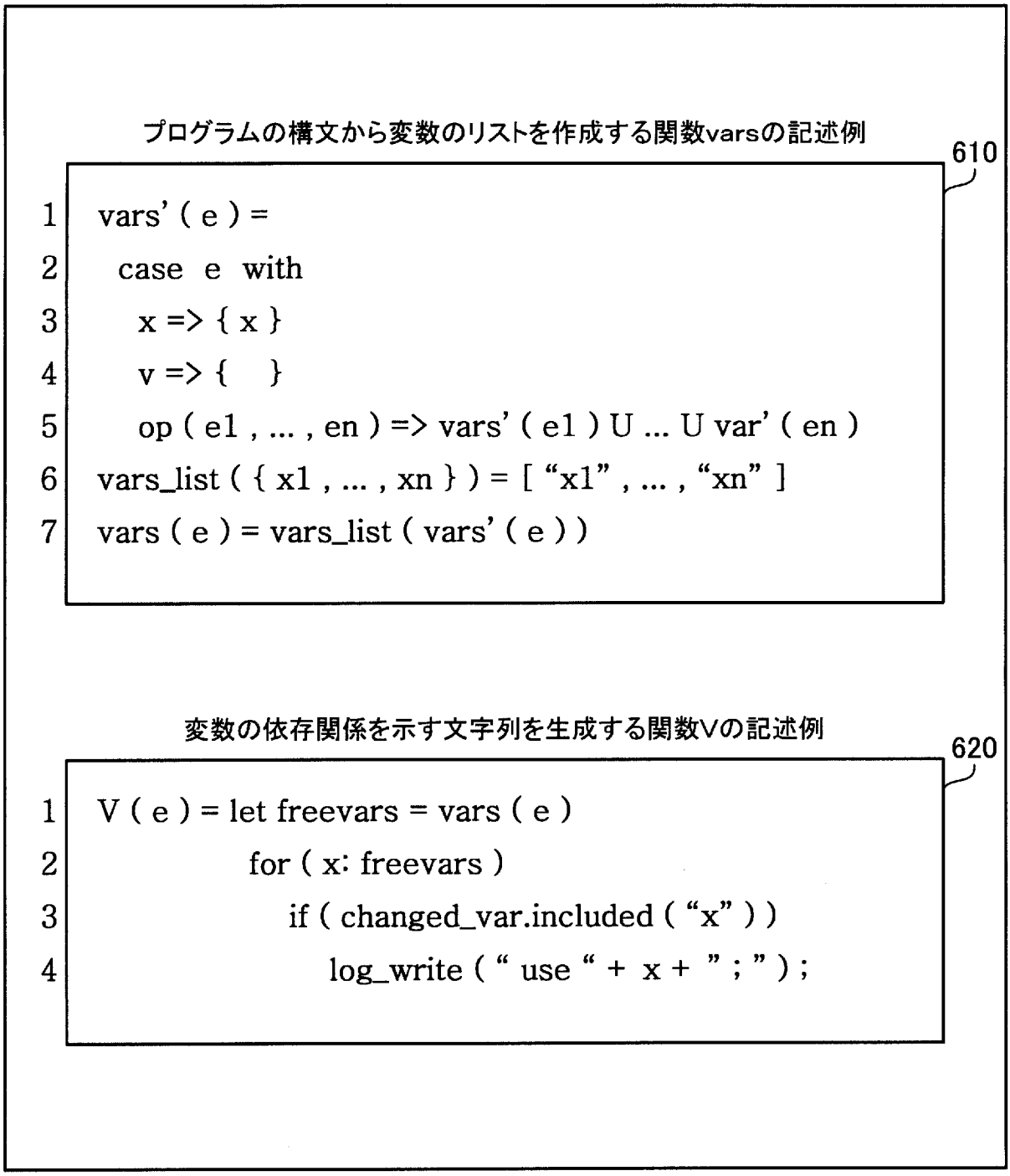
```
1  …
2  DISK="/dev/hda1"
3  …
4  execute "pvcreate #{DISK}"
5      do command "pvcreate --uuid #{node{ 'uuid' }} #{DISK}"
6  end
7  …
```

[図21]

BNF記法で記したプログラムのアブストラクトシンタクス

```
1  P ::= C | C ; P
2  C ::= x := e | while ( e ) C | if e then C else C | D
3  e ::= x | v | op ( e , ... , e )
```

[図22]


プログラムの構文から変数のリストを作成する関数varsの記述例
610
1 vars' (e) =
2 case e with
3 x => { x }
4 v => { }
5 op (e1 , ... , en) => vars' (e1) U ... U var' (en)
6 vars_list ({ x1 , ... , xn }) = ["x1" , ... , "xn"]
7 vars (e) = vars_list (vars' (e))
変数の依存関係を示す文字列を生成する関数Vの記述例
620
1 V (e) = let freevars = vars (e)
2 for (x: freevars)
3 if (changed_var.included ("x"))
4 log_write (" use " + x + " ; ") ;

[図23]

配備プログラムを変換するプログラムの例

630

```
T(C;P) = T(C);T(P)
T(C) =
    case C with
        while (e) C => if(changed_var.included(vars(e))) then
                        log_write ("following blocks can be changed;");
                            while (e) S(C)
                       if(changed_var.included(vars(e))) then
                        log_write ("prev blocks can be changed;");
         if e then C1 else C2 => if(changed_var.included(vars(e))) then
                        log_write ("following blocks can be changed;");
                            if e then S(C1) else S(C2)
                       if(changed_var.included(vars(e))) then
                        log_write ("prev blocks can be changed;");
    _ =>
    if (changecheck(C))
     then
      case C with
      x:=e => x:=e;log_write("x changed;");changed_var.add("x");
      D => let id = new() in
        log_write ("block changed;");log_write ("block " + id + " enter;");D ;log_write ("block " + id + " exit;")
     else
      case C with
      x:=e => x:=e; if(changed_var.included(vars(e))){log_write("x changed;");V(e);}
      D => let id = new() in
        log_write ("block " + id + "enter;");D ;log_write ("block " + id + " exit;")
```

[図24]

代入値の変更有無に関わらず変更の可能性を示すログを作成する関数Sの例

640

```
1  S(C) =
2    case C with
3      while (e) C => while (e) S(C)
4    if e then C1 else C2 => if e then S(C1) else S(C2)
5      _ =>
6    if (changecheck(C))
7    then
8      case C with
9      x:=e => x:=e;log_write("x changed;");changed_var.add("x");
10     D => let id = new() in
11       log_write ("block changed;");log_write ("block " + id + "enter;");D ;log_write ("block " + id + " exit;")
12   else
13     case C with
14     x:=e => x:=e; log_write("x changed;");V(e)
15     D => let id = new() in
16       log_write ("block " + id + "enter;");D ;log_write ("block " + id + " exit;")
```

[図25]

配備プログラムの他の例

112b

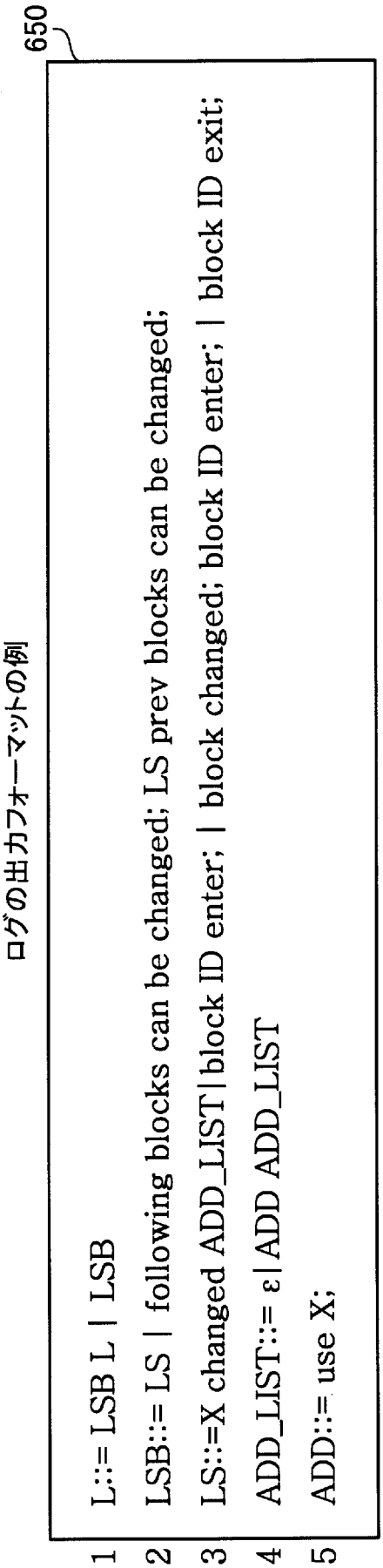
```
1  …
2  if ( atoi (argv[1]) == 1 ) then {
3          DISK = “/dev/hda0”
4  } else {
5          DISK = “/dev/hda1”
6  }
7  …
```

変換後配備プログラムの他の例

113a

```
1  …
2  log_write ( “following blocks can be changed;” )
3  if ( atoi (argv[1]) == 1 ) then {
4          DISK = “/dev/hda0”
5          log_write ( “DISK changed;” )
6          changed_var.add ( “DISK” )
7  } else {
8          DISK = “/dev/hda1”
9          log_write ( “DISK changed;” )
10         changed_var.add ( “DISK” )
11 }
12 log_write ( “prev blocks can be changed;” )
13 …
```

[図26]


650
ログの出力フォーマットの例
1 L::= LSB L | LSB
2 LSB::= LS | following blocks can be changed; LS prev blocks can be changed;
3 LS::=X changed ADD_LIST|block ID enter; | block changed; block ID enter; | block ID exit;
4 ADD_LIST::= ε|ADD ADD_LIST
5 ADD::= use X;

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP2013/071724

A. CLASSIFICATION OF SUBJECT MATTER
G06F11/28(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)
G06F11/28

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 1996-2013
Kokai Jitsuyo Shinan Koho 1971-2013 Toroku Jitsuyo Shinan Koho 1994-2013

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 10-320234 A (Hitachi, Ltd.), 04 December 1998 (04.12.1998), entire text; all drawings (Family: none)	1-10

☐ 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	Date of mailing of the international search report
11 September, 2013 (11.09.13)	22 October, 2013 (22.10.13)
Name and mailing address of the ISA/ Japanese Patent Office Facsimile No.	Authorized officer Telephone No.

国際調査報告

国際出願番号　ＰＣＴ／ＪＰ２０１３／０７１７２４

Ａ．　発明の属する分野の分類（国際特許分類（ＩＰＣ））

Int.Cl.　G06F11/28(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.　G06F11/28

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	１９７１－２０１３年
日本国実用新案登録公報	１９９６－２０１３年
日本国登録実用新案公報	１９９４－２０１３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 10-320234 A (株式会社日立製作所) 1998.12.04, 全文, 全図（ファミリーなし）	１－１０

☐　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
１１．０９．２０１３	２２．１０．２０１３

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　３５６２
多胡　滋

電話番号　０３－３５８１－１１０１　内線　３５４５