(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
2024年10月17日(17.10.2024)

WIPO | PCT

(10) 国際公開番号
WO 2024/214260 A1

(51) 国際特許分類:
G06F 11/34 (2006.01) *G06F 21/56* (2013.01)

(21) 国際出願番号： PCT/JP2023/015088

(22) 国際出願日： 2023年4月13日(13.04.2023)

(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: 日本電信電話株式会社 (**NIPPON TELEGRAPH AND TELEPHONE CORPORATION**) [JP/JP]; 〒1008116 東京都千代田区大手町一丁目５番１号 Tokyo (JP).

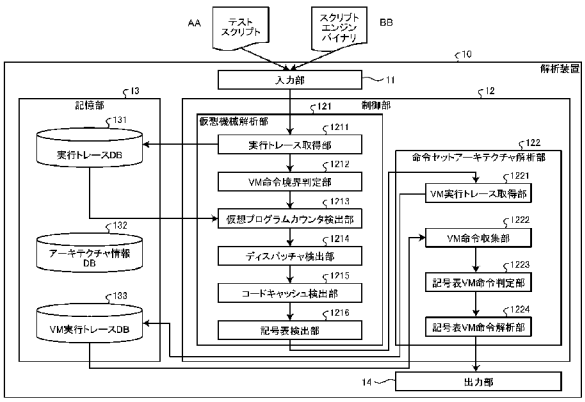
(72) 発明者: 碓井 利宣(**USUI, Toshinori**); 〒1808585 東京都武蔵野市緑町３丁目９－１１ ＮＴＴ 知的財産センタ内 Tokyo (JP). 川古谷 裕平(**KAWAKOYA, Yuhei**); 〒1808585 東京都武蔵野市緑町３丁目９－１１ ＮＴＴ 知的財産センタ内 Tokyo (JP). 岩村 誠(**IWAMURA, Makoto**); 〒1808585 東京都武蔵野市緑町３丁目９－１１ ＮＴＴ 知的財産センタ内 Tokyo (JP).

(74) 代理人: 弁理士法人酒井国際特許事務所 (**SAKAI INTERNATIONAL PATENT OFFICE**); 〒1000013 東京都千代田区霞が関３丁目８番１号 虎ノ門ダイビルイースト 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, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR,

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

(54) 発明の名称： 解析装置、解析方法及び解析プログラム



10 Analysis device
11 Input unit
12 Control unit
13 Storage unit
14 Output unit
121 Virtual machine analysis unit
122 Command set architecture analysis unit
131 Execution trace database
132 Architecture information database
133 VM execution trace database
1211 Execution trace acquisition unit
1212 VM command boundary determination unit
1213 Virtual program counter detection unit
1214 Dispatcher detection unit
1215 Code cache detection unit
1216 Symbol table detection unit
1221 VM execution trace acquisition unit
1222 VM command collection unit
1223 Symbol table VM command determination unit
1224 Symbol table VM command analysis unit
AA Test script
BB Script engine binary

(57) **Abstract:** An analysis device (10) includes: a virtual machine analysis unit (121) that analyzes a script engine VM on the basis of an execution trace acquired by executing a test script while monitoring script engine binary, and that detects, on the basis of the analysis results, a symbol table for holding information related to variables; and a command set architecture analysis unit (122) that collects VM commands on the basis of the results of analyzing the script engine VM, and that analyzes the collected VM commands on the basis of the results of analyzing the script engine VM.

WO 2024/214260 A1

［続葉有］

HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, 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, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, 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, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, SD, SL, ST, 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, ME, 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).

添付公開書類：

一　国際調査報告（条約第21条(3)）

(57) 要約：解析装置（１０）は、スクリプトエンジンのバイナリを監視しながらテストスクリプトを実行して取得した実行トレースを基に、スクリプトエンジンのＶＭを解析し、解析結果を基に、変数に関する情報を保持する記号表を検出する仮想機械解析部（１２１）と、スクリプトエンジンのＶＭの解析結果に基づいて、ＶＭ命令を収集し、スクリプトエンジンのＶＭの解析結果に基づいて、収集したＶＭ命令を解析する命令セットアーキテクチャ解析部（１２２）と、を有する。

明　細　書

発明の名称 ： 解析装置、解析方法及び解析プログラム

技術分野

[0001]　本発明は、解析装置、解析方法及び解析プログラムに関する。

背景技術

[0002] ［スクリプトの解析］

スクリプトを解析する技術は、多様な目的に用いられる。例えば、ＪＩＴ（Just-In-Time）コンパイルのためのコンパイラ最適化、ソフトウェアのテストやデバッグ、ファジング、マルウェア解析などが挙げられる。

[0003] ［変数情報の取得の重要性］

スクリプトの解析において、変数の情報を取得することは、重要な要素の一つである。スクリプトが実行時にどのような変数を保持するかは、そのスクリプトの持つ機能を理解するために、重要な情報となる。

[0004]　ここでの変数の情報として、どのようなスコープに何個の変数があるか、どの変数がどのような値を保持しているか、どのコード箇所がどの変数に読み書きしているか、などが挙げられる。このような変数の情報を取得するには、スクリプトを解析する必要がある。

[0005]　この解析を実現するために、例えば、デバッガやバイトコードの逆アセンブラなどの解析支援機能が提供されており、それらを用いることができる場合もある。しかしながら、そうした機能がない場合は、スクリプトを独自に解析して変数の情報を得る必要があり、容易ではない。

[0006] ［解析の手法］

スクリプトを含むプログラムを解析する技術に、静的解析と動的解析が存在する。動的解析は、解析対象のプログラムを実際に実行し、振る舞いを観測することで、その挙動を解析する技術である。静的解析は、解析対象のプログラムを実行することなく、プログラムの持つ意味を解釈していくことで、その機能を解析する技術である。分岐の情報を取得する際には、このよう

な解析技術を用いる。

[0007] ［難読化への対応の必要性］

ここで、解析を妨害する技術に、難読化がある。難読化は、主に静的解析を妨害するために、プログラムの解釈を困難にする変換を施すものである。スクリプトに対する難読化においては、例えば、スクリプトの一部にエンコードや暗号化が施されており、実行時に動的にデコードや復号をしてから実行する、というものがある。このような場合には、どのようなスクリプトが実行されるかは、実行時まで明らかにならない。このため、スクリプトの静的解析が困難になる。

[0008] 悪性なスクリプトや保護されたスクリプトは、一般に難読化されており、静的解析は困難となる。ソフトウェアの保護は、作成者の知的財産権を守る上で重要な技術である。ソフトウェアがどのような技術を用いて実装されているかを明らかにする技術に、リバースエンジニアリングがある。これは、プログラムを解析して、その構造や仕様を理解する技術である。ソフトウェア保護とは、こうしたリバースエンジニアリングのためのプログラム解析から、ソフトウェアを守るための技術である。

[0009] ［スクリプトの実行方式］

スクリプトはクリプトエンジン（インタプリタとも呼ばれる）によって実行される。スクリプトは一般に、実行時にバイトコードに変換され、そのバイトコードが仮想機械（Virtual Machine：ＶＭ）によって解釈実行される。このため、実行前にはスクリプトを解析し、実行時にはバイトコードを解析することになる。

[0010] ここで、前述の通り、スクリプトが難読化されている場合には、実行前の静的解析は困難となる。このため、バイトコードを動的解析して、実行時に得られる情報から、変数の情報を取得する必要がある。

[0011] ［記号表について］

スクリプトエンジンにおいて、変数は、一般に記号表によって管理される。記号表とは、変数やサブルーチンに関する名前などの情報を格納する表で

あり、意味解析のフェーズにおいて作成される。スクリプトエンジンの記号表は、一般に、名前、種別（変数か、サブルーチンか、など）、スコープ、属性（種別に応じる。例えば、定数か変数かや、型など）、そして値の情報を保持する。そして、このような記号表へのアクセスは、そのために用意されたＶＭ命令（以降、記号表ＶＭ命令と呼ぶ）を通して実現される。

[0012] したがって、スクリプトの用いる変数の情報を明らかにするためには、記号表に関する以下の４つの情報が必要となる。

[0013] １つ目は、記号表のメモリ上の位置と、記号表のメモリ領域を確保するコード箇所である。２つ目は、記号表の構造である。３つ目は、記号表ＶＭ命令のオペコードのリストである。そして、４つ目は、記号表ＶＭ命令のオペランド（命令の対象）、特に、どのオペランドがスコープや変数の識別を担うかである。

[0014] ここで、実行可能バイナリ形式のプログラムを解析し、データの構造を復元する手法が提案されている（非特許文献１）。非特許文献１記載の手法によれば、その記号表の一部を復元して、変数の情報を得ることができる。

先行技術文献

非特許文献

[0015] 非特許文献1：Asia Slowinska, Traian Stancescu, and Herbert Bos, “Howard: a Dynamic Excavator for Reverse Engineering Data Structures”, In Proceedings of the 18 the Network and Distributed Systems Security Symposium, 2011.

発明の概要

発明が解決しようとする課題

[0016] しかしながら、非特許文献１に記載の手法では、スクリプトエンジンが保持する記号表は対象としておらず、スクリプトに特有の記号表の構造や、前述の記号表ＶＭ命令の解析については実現されていないため、直接的に適用できないという課題があった。

[0017] 本発明は、上記に鑑みてなされたものであって、仕様が未知の多種多様なスクリプト言語のスクリプトエンジンに対しても、ＶＭの記号表を解析するとともに、記号表にアクセスするＶＭ命令を解析して、記号表によって管理される変数の情報を取得することができる解析装置、解析方法及び解析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

[0018] 上述した課題を解決し、目的を達成するために、本発明の解析装置は、スクリプトエンジンのバイナリを監視しながらテストスクリプトを実行して取得した実行トレースを基に、前記スクリプトエンジンの仮想機械を解析し、解析結果を基に、変数に関する情報を保持する記号表を検出する第１の解析部と、前記スクリプトエンジンの仮想機械を解析し、解析結果に基づいて、仮想機械命令を収集し、前記解析結果に基づいて、収集した前記仮想機械命令を解析する第２の解析部と、を有することを特徴とする。

発明の効果

[0019] 本発明によれば、仕様が未知の多種多様なスクリプト言語のスクリプトエンジンに対しても、ＶＭの記号表を解析するとともに、記号表にアクセスするＶＭ命令を解析して、記号表によって管理される変数の情報を取得することができる。

図面の簡単な説明

[0020] [図1]図１は、スクリプトエンジンの構成の一例を説明するための図である。

[図2]図２は、スクリプトエンジンが有するＶＭの擬似コードを示す図である。

[図3]図３は、実施の形態に係る解析装置の構成の一例を説明する図である。

[図4]図４は、仮想プログラムカウンタ（ＶＰＣ）の検出に用いる第１のテストスクリプトの一例を示す図である。

[図5]図５は、第２のテストスクリプトの一例を示す図である。

[図6]図６は、第３のテストスクリプトの一例を示す図である。

[図7]図７は、第４のテストスクリプトの一例を示す図である。

[図8]図８は、第４のテストスクリプトの一例を示す図である。

[図9]図９は、実行トレースの一例を示す図である。

[図10]図１０は、ＶＭ実行トレースの一例を示す図である。

[図11]図１１は、記号表の構造の一例を示す図である。

[図12]図１２は、記号表検出部の処理を説明する図である。

[図13]図１３は、記号表ＶＭ命令解析部の処理を説明する図である。

[図14]図１４は、記号表ＶＭ命令解析部の処理を説明する図である。

[図15]図１５は、ＶＭ命令境界検出部の処理を説明する図である。

[図16]図１６は、仮想プログラムカウンタ検出部の処理を説明する図である。

[図17]図１７は、ディスパッチャ検出部の処理を説明する図である。

[図18]図１８は、コードキャッシュ検出部の処理を説明する図である。

[図19]図１９は、実施の形態に係る解析処理の処理手順を示すフローチャートである。

[図20]図２０は、図１９に示す実行トレース取得処理の処理手順を示すフローチャートである。

[図21]図２１は、図１９に示すＶＭ命令境界検出処理の処理手順を示すフローチャートである。

[図22]図２２は、図１９に示す仮想プログラムカウンタ検出部の処理を説明する図である。

[図23]図２３は、図１９に示すディスパッチャ検出部の処理を説明する図である。

[図24]図２４は、図１９に示すコードキャッシュ検出処理の処理手順を示すフローチャートである。

[図25]図２５は、図１９に示す記号表検出処理の処理手順を示すフローチャートである。

[図26]図２６は、図１９に示すＶＭ実行トレース取得処理の処理手順を示すフローチャートである。

[図27]図２７は、図１９に示すＶＭ命令収集処理の処理手順の処理手順を示すフローチャートである。

[図28]図２８は、図１９に示す記号表ＶＭ命令判定処理の処理手順を示すフローチャートである。

[図29]図２９は、図１９に示す記号表ＶＭ命令解析処理の処理手順を示すフローチャートである。

[図30]図３０は、図１９に示す記号表ＶＭ命令解析処理の処理手順を示すフローチャートである。

[図31]図３１は、プログラムが実行されることにより、解析装置が実現されるコンピュータの一例を示す図である。

発明を実施するための形態

[0021] 以下に、本願に係る解析装置、解析方法及び解析プログラムの実施形態を図面に基づいて詳細に説明する。また、本発明は、以下に説明する実施形態により限定されるものではない。

[0022] ［実施の形態］

実施の形態に係る解析装置は、スクリプトエンジンのバイナリを監視しながらテストスクリプトを実行して、ブランチトレース及びメモリアクセストレースを実行トレースとして取得する。解析装置は、実行トレースに基づいて仮想機械（Virtual Machine：ＶＭ）を解析し、アーキテクチャ情報として、ＶＭ命令境界、仮想プログラムカウンタ（ＶＰＣ）、及び、ディスパッチャ、実行されるＶＭ命令が保存されるコードキャッシュを取得する。そして、解析装置は、スクリプトエンジンのＶＭの解析結果を基に、変数に関する情報を保持する記号表を検出する。

[0023] 解析装置は、仮想機械の命令の体系である命令セットアーキテクチャを解析する。解析装置は、ＶＰＣ及びディスパッチャを監視しながらテストスクリプトを実行して、ＶＭ実行トレースを取得する。解析装置は、ＶＭ実行トレースを解析することで、ＶＭ命令を収集する。解析装置は、収集されたＶＭ命令のうち、スクリプトエンジンのＶＭの解析結果に基づいて、収集した

ＶＭ命令のうち、記号表にアクセスする記号表ＶＭ命令を判定し、記号表ＶＭ命令のアクセス先を解析する。これによって、解析装置は、記号表によって管理される変数の情報を取得する。

[0024] 図１及び図２を参照して、一般的なスクリプトエンジンの構成とそれらの働きについて説明する。図１は、スクリプトエンジンの構成の一例を説明するための図である。図１に示すように、スクリプトエンジン１は、バイトコードコンパイラ２とＶＭ３を有する。また、バイトコードコンパイラ２は、構文解析部４、バイトコード生成部５を有する。また、ＶＭ３は、コードキャッシュ部６、フェッチ部７、デコード部８、実行部９を有する。これらのフェッチ部７、デコード部８、実行部９は、繰り返し実行され、インタプリタループと呼ばれる。そして、スクリプトエンジン１は、スクリプトの入力を受け付ける。

[0025] 構文解析部４は、スクリプトを入力として受け取り、字句解析及び構文解析を経て、抽象構文木（Abstract Syntax Tree：ＡＳＴ）を生成し、バイトコード生成部５に出力する。バイトコード生成部５は、ＡＳＴを入力として受け取り、バイトコードに変換してコードキャッシュ部６に格納する。

[0026] フェッチ部７は、コードキャッシュ部６からＶＭオペコードをフェッチし、デコード部８に出力する。ここで、ＶＭオペコードは、ＶＭ命令のオペコード部を指す。デコード部８は、ＶＭオペコードを入力として受け取り、デコーダ・ディスパッチャを用いてＶＭオペコードを解釈し、対応したプログラムにディスパッチする。実行部９は、ＶＭ命令に対応したプログラムを実行する。インタプリタループの繰り返しにより、ＶＭ命令を次々に実行していくことで、スクリプトに記述した内容が実行される。

[0027] 図２を参照して、スクリプトエンジンの構成要素の働きについて説明する。図２は、スクリプトエンジンが有するＶＭの擬似コードを示す図である。図２に示すように、まず、擬似コードは、ＶＰＣを初期化している（１行目）。擬似コードでは、while文のループがインタプリタループである（２～７行目）。擬似コードでは、フェッチ部７が、バイトコードを保持するコード

キャッシュ中でＶＰＣの指す位置にあるＶＭ命令のＶＭオペコードを取得する（３行目）。デコーダは、Switch文を用い、ＶＭ命令を解釈して（４行目）、そのＶＭオペコードに基づき、ディスパッチャが命令ハンドラを呼び出す（５行目～）。そして、擬似コードでは、命令ハンドラが命令に対応した操作を行う。入出力は、仮想スタックや仮想レジスタを用いて行い（６行目）、定数や変数の参照は記号表を介して行う（７行目）。

[0028] ［解析装置の構成］

続いて、図３を参照して、実施の形態に係る解析装置１０の構成について具体的に説明する。図３は、実施の形態に係る解析装置１０の構成の一例を説明する図である。

[0029] 図３に示すように、解析装置１０は、入力部１１、制御部１２、記憶部１３及び出力部１４を有する。そして、解析装置１０は、テストスクリプト及びスクリプトエンジンバイナリの入力を受け付ける。

[0030] 入力部１１は、キーボードやマウス等の入力デバイスで構成され、外部からの情報の入力を受け付け、制御部１２に入力する。また、入力部１１は、有線接続、或いは、ネットワーク等を介して接続された他の装置との間で、各種情報を送受信する通信インタフェースを有し、他の装置から送信された情報の入力を受け付ける。入力部１１は、テストスクリプト及びスクリプトエンジンバイナリの入力を受け付け、制御部１２に出力する。

[0031] テストスクリプトは、スクリプトエンジンを動的解析して実行トレース及びＶＭ実行トレースを取得する際に、入力されるスクリプトであり、ＶＭ解析用の第１のテストスクリプトと、記号表検出用の第２のテストスクリプトと、記号表ＶＭ命令解析用の第３のテストスクリプト及び第４のテストスクリプトとを含む。スクリプトエンジンバイナリは、スクリプトエンジンを構成する実行可能ファイルである。スクリプトエンジンバイナリは、複数の実行可能ファイルによって構成される場合がある。

[0032] ［テストスクリプトの構成］

テストスクリプトについて説明する。テストスクリプトは、スクリプトエ

ンジンを動的解析する際に入力されるスクリプトである。このテストスクリプトは、分岐命令の実行やメモリ読み書きの回数に着目し、異なる回数のテストスクリプトを実行したときに生じるスクリプトエンジンの挙動の差分を捉えるために用いられる。このテストスクリプトは、解析の事前に準備するものであり、手動で作成するものである。この作成には、対象のスクリプト言語の仕様に関する知識が必要となる。

[0033] 図４は、ＶＰＣの検出に用いる第１のテストスクリプトの一例を示す図である。第１のテストスクリプトでは、繰り返し処理を用いる（２行目）。第１のテストスクリプトでは、テストスクリプト内の繰り返し回数（２行目）や繰り返される文の数（３行目から５行目）を増減させることで、実行時の条件を変更し、差分を発生させる。なお、第１のテストスクリプトは、後述する実行トレース取得処理からコードキャッシュ検出処理までの処理対象となる。

[0034] 図５は、第２のテストスクリプトの一例を示す図である。第２のテストスクリプトでは、記号表を検出するために、値のマッチングを可能にするため、特徴的な値が用いられる。例えば、図５の例では、「3735928559」である。第２のテストスクリプトに特徴的な値を含めることにより、記号表検出時に、メモリアクセストレースから、特徴的な値のマッチングにより、この特徴的な値の格納箇所を検出することを可能にする。なお、第２のテストスクリプトは、後述する実行トレース取得処理及び記号表検出処理の対象となる。

[0035] 図６は、第３のテストスクリプトの一例を示す図である。第３のテストスクリプトは、異なる２つのスコープ（例えば、「func1」、「func2」）の変数（例えば、「ａ」、「ｂ」）にアクセスするテストスクリプトである。第３のテストスクリプトによるメモリアクセストレースから、第１の組み合わせの各記号表ＶＭ命令の実行中にアクセスされた各部分の差分となる箇所と、オペランドの位置との比較によって、差分となる箇所が、記号表を指定するオペランドであるか否かが判定される。

[0036] 図７及び図８は、第４のテストスクリプトの一例を示す図である。第４のテストスクリプトは、同じスコープ内の異なる２つの変数（例えば、「ａ」、「ｂ」）にアクセスするテストスクリプトである。第４のテストスクリプトによるメモリアクセストレースから、第２の組み合わせの各記号表ＶＭ命令の実行中にアクセスされた各部分の差分となる箇所と、オペランドの位置との比較によって、差分となる箇所が、変数を指定するオペランドであるか否かが判定される。なお、第３及び第４のテストスクリプトは、いずれも、後述する実行トレース取得処理、ＶＭ実行トレース取得処理及び記号表ＶＭ命令解析処理の対象となる。

[0037] 記憶部１３は、ＲＡＭ（Random Access Memory）、フラッシュメモリ（Flash Memory）等の半導体メモリ素子、または、ハードディスク、光ディスク等の記憶装置によって実現され、解析装置１０を動作させる処理プログラムや、処理プログラムの実行中に使用されるデータなどが記憶される。記憶部１３は、実行トレースデータベース（ＤＢ）１３１、ＶＭ実行トレースＤＢ１３３、及び、仮想機械解析部１２１及び命令セットアーキテクチャ解析部１２２（後述）によって取得されたアーキテクチャ情報を記憶するアーキテクチャ情報ＤＢ１３２を有する。

[0038] 実行トレースＤＢ１３１及びＶＭ実行トレースＤＢ１３３は、それぞれ実行トレース取得部１２１１及びＶＭ実行トレース取得部１２２１によって取得された実行トレース及びＶＭ実行トレースを格納する。実行トレースＤＢ１３１及びＶＭ実行トレースＤＢ１３３は、解析装置１０によって管理される。もちろん、実行トレースＤＢ１３１及びＶＭ実行トレースＤＢ１３３は、他の装置（サーバ等）によって管理されていてもよく、この場合には、実行トレース取得部１２１１（後述）及びＶＭ実行トレース取得部１２２１（後述）は、出力部１４の通信インタフェースを介して、取得した実行トレース及びＶＭ実行トレースを、実行トレースＤＢ１３１及びＶＭ実行トレースＤＢ１３３の管理サーバ等に出力して、実行トレースＤＢ１３１及びＶＭ実行トレースＤＢ１３３に記憶させる。

[0039] ［実行トレースの構成］

次に、実行トレースについて説明する。図９は、実行トレースの一例を示す図である。実行トレースは、ブランチトレースとメモリアクセストレースによって構成されている。図９は、実行トレースの一部を切り出したものである。以降、図６を用いて実行トレースの構成を示す。

[0040] 実行トレースは、traceという要素を有する。traceには、そのログ行がブランチトレースか、メモリアクセストレースかが示される。

[0041] ブランチトレースのログ行は、例えば、図９の１行目から１０行目に記載の書式になっており、type、src、dstの三つの要素からなる。typeは、実行された分岐命令がcall命令によるものか、jmp命令によるものか、ret命令によるものかを示す。また、srcは、分岐元のアドレスを示し、dstは、分岐先のアドレスを示す。

[0042] メモリアクセストレースのログ行は、例えば、図９の１１行目から１３行目に記載の書式になっており、type、target、valueの三つの要素からなる。typeは、メモリアクセスが読み込みか書き込みかを示す。targetは、メモリアクセスの対象となるメモリアドレスを示す。また、valueには、メモリアクセスの結果の値が格納される。

[0043] ［ＶＭ実行トレースの構成］

次に、ＶＭ実行トレースについて説明する。図１０は、ＶＭ実行トレースの一例を示す図である。ＶＭ実行トレースは、前述の通り、ＶＭオペコードとＶＰＣとを記録したものである。図１０は、ＶＭ実行トレースの一部を切り出したものである。以降、図１０を用いてＶＭ実行トレースの構成を示す。

[0044] ＶＭ実行トレースのログ行は、例えば、図１０に記載の書式になっており、vpc及びvmop（vm opcode）の二つの要素からなる。vpcは、ＶＰＣの値を示す。また、vmopは、ポインタキャッシュから取得された、実行されるＶＭ命令ハンドラの先頭を指すポインタごとに仮想的に割り振られたＶＭオペコードの値を示す。

[0045] 記号表は、スコープごとに存在する。図１１は、記号表の構造の一例を示す図である。図１１に示す記号表２１０に示すように、記号表は、変数及び定数に関する情報と、関数に関する情報とを有する。例えば、記号表２１０は、１列目に序数が示され、２列目に変数、定数、または、関数を示す情報が示され、３行目にオブジェクトの実体のアドレスが示される。

[0046] 図３に戻り、制御部１２について、説明する。制御部１２は、各種の処理手順などを規定したプログラム及び所要データを格納するための内部メモリを有し、これらによって種々の処理を実行する。例えば、制御部１２は、ＣＰＵ（Central Processing Unit）やＭＰＵ（Micro Processing Unit）などの電子回路である。制御部１２は、仮想機械解析部１２１（第１の解析部）及び命令セットアーキテクチャ解析部１２２（第２の解析部）を有する。

[0047] 仮想機械解析部１２１は、スクリプトエンジンのＶＭを解析する。仮想機械解析部１２１は、実行時の条件を変えて複数の実行トレースを取得し、差分実行解析を用いて複数の実行トレースを解析し、ＶＰＣを取得する。また、仮想機械解析部１２１は、スクリプトエンジンバイナリを解析して、ＶＭ命令の境界及びディスパッチャを取得する。仮想機械解析部１２１は、実行トレースから、コードキャッシュを検出する。コードキャッシュには、実行されるＶＭ命令が保存される。仮想機械解析部１２１は、記号表を検出する。

[0048] 仮想機械解析部１２１は、実行トレース取得部１２１１（第１の取得部）、ＶＭ命令境界検出部１２１２（第１の検出部）、仮想プログラムカウンタ検出部１２１３（第２の検出部）、ディスパッチャ検出部１２１４（第３の検出部）、コードキャッシュ検出部１２１５（第４の検出部）及び記号表検出部１２１６（第５の検出部）を有する。

[0049] 実行トレース取得部１２１１は、第１～第４のテストスクリプト及びスクリプトエンジンバイナリを入力として受け付ける。実行トレース取得部１２１１は、スクリプトエンジンバイナリの実行を監視しながら、第１～第４の

テストスクリプトを実行することで、実行トレースを取得する。

[0050] 実行トレースは、ブランチトレースとメモリアクセストレースとによって構成される。ブランチトレースは、実行の際の分岐命令の種類と、分岐元アドレスと分岐先アドレスを記録する。メモリアクセストレースは、実行時のメモリ操作の種類（読み書き）と、操作対象のメモリアドレスと値とを記録する。ブランチトレース及びメモリアクセストレースは、メモリ操作命令をフックしてログ出力用のコードを挿入し、それを実行させることによって取得可能であることが知られている。実行トレース取得部１２１１が取得した実行トレースは、実行トレースＤＢ１３１に格納される。

[0051] また、実行トレース取得部１２１１は、実行トレース取得時に、ＡＰＩ（Application Programming Interface）トレースを取得し、実行トレースＤＢ１３１に格納する。ＡＰＩトレースは、実行の際に、呼び出されたシステムＡＰＩとその引数を記録したものである。

[0052] ＶＭ命令境界検出部１２１２は、第１のテストスクリプトに対する実行トレースをクラスタリングして、各ＶＭ命令の境界を検出する。ＶＭ命令境界検出部１２１２は、実行トレースをクラスタリングして、実行回数が閾値以上のクラスタをＶＭ命令として検出する。クラスタリングでは、複数回実行される連続したコード領域を検出する。これには、例えば、実行された命令間のコード上の距離が近いものをまとめてもよいし、実行されたコードブロックの共通部分列を探してもよいし、他の方法によってもよい。解析装置１０は、検出したＶＭ命令を構成する連続した命令列の開始点と終了点とを境界として検出する。ここで検出されたＶＭ命令の境界は、ＶＰＣ検出、ディスパッチャ検出において用いられる。

[0053] 仮想プログラムカウンタ検出部１２１３は、実行トレースＤＢ１３１に格納された第１のテストスクリプトに対する実行トレースを取り出して解析し、ＶＰＣを検出する。仮想プログラムカウンタ検出部１２１３は、メモリの読み込み回数に着目した差分実行解析とＶＭ命令境界検出部１２１２によって検出された各ＶＭ命令の境界とを用いて複数の実行トレースを解析し、Ｖ

ＰＣを検出する。仮想プログラムカウンタ検出部１２１３は、各ＶＭ命令の実行後には、必ずＶＰＣを保持するメモリへの読み込みが発生することを利用し、この読み込み先を発見することで、ＶＰＣを検出する。

[0054] このため、仮想プログラムカウンタ検出部１２１３は、ＶＰＣの検出として、メモリの読み込み回数に着目した差分実行解析を用いる。仮想プログラムカウンタ検出部１２１３は、第１のテストスクリプトを用いて取得された複数のテストスクリプトの実行トレースを比較し、メモリ読み込み回数が、繰り返される回数及び繰り返される文の数との双方の増減に比例して変化するメモリを発見する。そして、仮想プログラムカウンタ検出部１２１３は、ＶＭ命令境界検出部１２１２によって検出された各ＶＭ命令の境界を参照して、読み込んだメモリの値が常にＶＭ命令の開始点を指しているものに絞り込む。仮想プログラムカウンタ検出部１２１３は、このメモリをＶＰＣとして検出する。

[0055] ディスパッチャ検出部１２１４は、ＶＭ命令境界検出部１２１２が検出したＶＭ命令の境界を基に、スクリプトエンジンバイナリから各ＶＭ命令部分を切り出し、各ＶＭ命令間で類似度が高い部分をディスパッチャとして検出する。前提として、ディスパッチャは、ポインタキャッシュの参照と次のＶＭ命令ハンドラのポインタへのジャンプで実現される。ディスパッチャは、各々のＶＭ命令ハンドラの後部に分散的に配置されており、一般にそれらのコードの同一性は高い。こうしたＶＭ命令ハンドラの後部に存在し、同一性の高いコードを探すことで、解析装置は、所定の方法でディスパッチャを検出する。類似度の高い部分の検出には、例えば、系列アライメントアルゴリズムを用いてもよく、その他の方法によってもよい。

[0056] コードキャッシュ検出部１２１５は、実行トレース、ＶＰＣ及びＶＭ実行トレースを基に、ＶＭ実行トレースから、実行される仮想機械命令が保存されるキャッシュであるコードキャッシュを検出する。

[0057] コードキャッシュ検出部１２１５は、ＶＰＣが指すメモリ領域をコードキャッシュとしてＶＭ実行トレースから検出する。コードキャッシュ検出部１

２１５は、このコードキャッシュを確保したメモリ割り当て関数の呼び出し元のコード箇所を、実行トレースから検出する。コードキャッシュ検出部１２１５は、ＶＭ実行トレースのうち、このコード箇所で確保された全てのメモリ領域をコードキャッシュとして検出する。

[0058] コードキャッシュ検出部１２１５は、コードキャッシュに書き込みをしているコード箇所を実行トレースから検出する。コードキャッシュ検出部１２１５は、実行トレースのうち、このコード箇所による書き込みをコードキャッシュの更新として検出する。なお、本実施の形態では、記号表の検出及び記号表ＶＭ命令の解析に関し、コードキャッシュを直接使用していないため、コードキャッシュ検出部１２１５及び後述のコードキャッシュ検出処理を省略することもできる。

[0059] 記号表検出部１２１６は、スクリプトエンジンのＶＭの解析結果を基に、変数に関する情報を保持する記号表のアーキテクチャ情報を検出する。

[0060] 図１２は、記号表検出部１２１６の処理を説明する図である。記号表検出部１２１６は、特徴的な値が用いられる第２のテストスクリプト及び第２のテストスクリプトによるメモリアクセストレースを用いて、記号表のメモリ領域上の位置と、記号表の構造とを検出する。記号表検出部１２１６は、第２のテストスクリプトのメモリアクセストレースから、特徴的な値のメモリ領域上の格納箇所と該特徴的な値への参照元を基に、記号表のメモリ領域上の位置と、特徴的な値へ参照する構造体とを検出する。記号表検出部１２１６は、記号表の作成時にメモリアクセストレース中に値が現れることから（図１２の（１））、第２のテストスクリプトの特徴的な値のマッチングで記号表内の変数を検出する（図１２の（２））。記号表検出部１２１６は、特徴的な値へ参照している構造体で、参照がまとまった領域を記号表として検出する（図１２の（３））。具体的には、記号表検出部１２１６は、ＡＰＩトレースから、記号表のメモリ領域を確保するコード箇所を検出する。記号表検出部１２１６は、記号表のメモリ領域上の位置と、その領域を確保するコード箇所とを出力する。

[0061] 命令セットアーキテクチャ解析部１２２は、スクリプトエンジンのＶＭの命令の体系である命令セットアーキテクチャを解析する。命令セットアーキテクチャ解析部１２２は、スクリプトエンジンのＶＭを解析し、スクリプトエンジンのＶＭの解析結果に基づいて、ＶＭ命令を収集する。命令セットアーキテクチャ解析部１２２は、スクリプトエンジンのＶＭの解析結果に基づいて、収集したＶＭ命令のうち、記号表にアクセスする記号表ＶＭ命令を判定し、記号表ＶＭ命令のアクセス先を解析する。

[0062] 命令セットアーキテクチャ解析部１２２は、ＶＭ実行トレース取得部１２２１（第２の取得部）、ＶＭ命令収集部１２２２（第１の収集部）、記号表ＶＭ命令判定部１２２３（第１の判定部）、記号表ＶＭ命令解析部１２２４（第２の解析部）を有する。

[0063] ＶＭ実行トレース取得部１２２１は、テストスクリプト及びスクリプトエンジンバイナリを入力として受け付ける。ＶＭ実行トレース取得部１２２１は、ＶＰＣの監視と、ディスパッチャがディスパッチするＶＭ命令ハンドラのポインタの監視により、ＶＭ実行トレースを取得する。ＶＭ実行トレース取得部１２２１は、スクリプトエンジンバイナリの実行を監視しながら、第３及び第４のテストスクリプトを実行することで、ＶＭ上で実行された実行トレースであるＶＭ実行トレースを取得する。ＶＭ実行トレース取得部１２２１は、ＶＭ命令へのポインタとＶＭ命令とを紐づけ、各々に識別子としてＶＭオペコードを仮想的に割り振る。

[0064] ＶＭ実行トレースは、ＶＭにおいて実行された実行トレースであって、識別子としてＶＭオペコードが仮想的に割り振られ、実行されたＶＭハンドラのポインタとＶＰＣとを記録したものである。ＶＭ実行トレースは、実行されたＶＭ命令ハンドラのポインタと、ＶＰＣを記録したものである。具体的には、ＶＭ実行トレースは、実行されたＶＭ命令ごとのＶＰＣとＶＭオペコードで構成される。ＶＰＣの記録は、仮想プログラムカウンタ検出部１２１３で検出されたＶＰＣのメモリを監視することで実現できる。ＶＭオペコードは、ＶＭ命令へのポインタとＶＭ命令とを紐づけた各々に仮想的に割り振

られた識別子である。ＶＭ実行トレース取得部１２２１が取得したＶＭ実行トレースは、ＶＭ実行トレースＤＢ１３３に格納される。

[0065] ＶＭ命令収集部１２２２は、ＶＰＣ及びディスパッチャを入力として受け付け、ＶＰＣ及びディスパッチャを監視しながらテストスクリプトを実行しＶＭ実行トレースを取得する。ＶＭ命令収集部１２２２は、ＶＭ実行トレースからＶＭ命令を収集する。

[0066] 記号表ＶＭ命令判定部１２２３は、ＶＭ命令収集部１２２２によって収集されたＶＭ命令のうち、ＶＭの実行中に、記号表のメモリ領域にアクセスしたＶＭ命令を記号表ＶＭ命令と判定する。記号表ＶＭ命令判定部１２２３は、記号表のメモリ領域にアクセスしたＶＭ命令のうち、読み込みを命令するＶＭ命令を、読み込みの記号表ＶＭ命令と判定する。記号表ＶＭ命令判定部１２２３は、記号表のメモリ領域にアクセスしたＶＭ命令のうち、書き込みを命令するＶＭ命令を、書き込みの記号表ＶＭ命令と判定する。記号表ＶＭ命令判定部１２２３は、各ＶＭ命令に対する判定結果に基づいて、記号表ＶＭ命令と判定されたＶＭ命令及びその種別を示すリストを出力する。

[0067] 記号表ＶＭ命令解析部１２２４は、ＶＭ実行トレースＤＢ１３３に格納されたＶＭ実行トレースを取り出して解析する。記号表ＶＭ命令解析部１２２４は、ＶＭ実行トレースから取り出した２つの記号表ＶＭ命令の組み合わせについて、メモリアクセストレースにおける、組み合わせの各記号表ＶＭ命令の実行中にアクセスされた部分の差分となる箇所が、オペランドの位置に存在する場合、差分となる箇所を、記号表ＶＭ命令のオペランドであると判定する。なお、ＶＭ命令のオペランドは、ＶＭ命令のＶＰＣとオペコードから求めることができる。オペランドは、一般に、「[opcode A][operand 1][operand 2][opcode B]…」のように、メモリ上でオペコードの隣、すなわち、現在のオペコードと次のオペコードとの間の部分に存在するため、ＶＰＣとオペコードとが分かれば、オペランドの位置を求めることができる。

[0068] 図１３は、記号表ＶＭ命令解析部１２２４の処理を説明する図である。記号表ＶＭ命令解析部１２２４は、異なる２つのスコープ（図１３の（１））

の変数にアクセスする第３のテストスクリプトによるメモリアクセストレースと、ＶＭ実行トレースとを用いて、記号表を指定するオペランドを判定する。

[0069] 記号表ＶＭ命令解析部１２２４は、第３のテストスクリプトによるＶＭ実行トレースから取り出した２つの記号表ＶＭ命令の第１の組み合わせについて、第３のテストスクリプトによるメモリアクセストレースから、第１の組み合わせの各記号表ＶＭ命令の実行中にアクセスされた部分をそれぞれ取り出す。記号表ＶＭ命令解析部１２２４は、取り出した２つの部分の差分となる箇所がオペランドの位置に存在する場合、差分となる箇所を、記号表を指定するオペランドと判定する（図１３の（２））。

[0070] 図１４は、記号表ＶＭ命令解析部１２２４の処理を説明する図である。記号表ＶＭ命令解析部１２２４は、同じスコープ内の異なる２つの変数（図１４の（１））にアクセスする第４のテストスクリプトによるメモリアクセストレースと、ＶＭ実行トレースとを用いて、変数を指定するオペランドを判定する。

[0071] 記号表ＶＭ命令解析部１２２４は、第４のテストスクリプトによるＶＭ実行トレースから取り出した２つの記号表ＶＭ命令の第２の組み合わせについて、第４のテストスクリプトによるメモリアクセストレースから、第２の組み合わせの各記号表ＶＭ命令の実行中にアクセスされた部分をそれぞれ取り出す。記号表ＶＭ命令解析部１２２４は、取り出した２つの部分の差分となる箇所が、オペランドの位置に存在する場合、差分となる箇所を、変数を指定するオペランドと判定する（図１４の（２））。

[0072] 記号表ＶＭ命令解析部１２２４は、記号表を指定するオペランドの情報、及び、変数を指定するオペランドの情報を出力する。

[0073] 出力部１４は、例えば、液晶ディスプレイやプリンタ等であって、解析装置１０に関する情報を含む各種情報を出力する。また、出力部１４は、外部装置との間で、各種データの入出力を司るインタフェースであってもよく、外部装置に各種情報を出力してもよい。

[0074] ［ＶＭ命令境界検出部の処理］

次に、ＶＭ命令境界検出部１２１２の処理について説明する。図１５は、ＶＭ命令境界検出部１２１２の処理を説明する図である。

[0075] ＶＭ命令境界検出部１２１２は、各ＶＭ命令の境界を検出する。この時、ＶＭ命令境界検出部１２１２は、インタプリタループを持たないためにＶＭ命令の境界の把握が難しいスレッデッドコード型ＶＭのために、ＶＭ命令とその境界の検出を行う。具体的には、ＶＭ命令境界検出部１２１２は、実行トレースＤＢ１３１から実行トレースを取り出す。そして、図１５に示すように、ＶＭ命令境界検出部１２１２は、実行トレースを、所定の方法でクラスタリングして、実行回数が閾値以上のクラスタをＶＭ命令（例えば、ＶＭ命令ハンドラ１～３）として検出する。ＶＭ命令境界検出部１２１２は、ＶＭ命令を構成する連続した命令列の開始点と終了点とを境界として検出する。

[0076] ［仮想プログラムカウンタ検出部の処理］

次に、仮想プログラムカウンタ検出部１２１３の処理について説明する。仮想プログラムカウンタ検出部１２１３は、ＶＰＣ、ポインタキャッシュの検出を行う。仮想プログラムカウンタの検出は、取得した実行トレースのメモリアクセストレースのログを解析することで実現される。仮想プログラムカウンタ検出部１２１３は、メモリの読み込み回数に着目した差分実行解析を用いる。図１６は、仮想プログラムカウンタ検出部１２１３の処理を説明する図である。

[0077] 仮想プログラムカウンタ検出部１２１３は、実行トレースＤＢ１３１から第１のテストスクリプトによる実行トレースを一つ取り出す。ＶＰＣの読み込みの回数は、テストスクリプト内の繰り返し回数及び、繰り返し処理の中の文の数に比例する。繰り返しの回数をＮ、繰り返される文の数をＭとしたとき、概ねＭＮ程度のＶＰＣの読み込みが発生する。このため、仮想プログラムカウンタ検出部１２１３は、Ｎ及びＭをそれぞれ２Ｎと２Ｍ、３Ｎと３Ｍと増やした第１のテストスクリプトに対する実行トレースにおいて、４Ｍ

Ｎ、９ＭＮという増え方をしたメモリを抽出する。具体的には、図１６に示すように、仮想プログラムカウンタ検出部１２１３は、１ＶＭ命令実行毎にRead／Writeがあり、単調増加するメモリ領域を抽出する（図１６の（１））。

[0078] そして、仮想プログラムカウンタ検出部１２１３は、読み込んだメモリの値が常にＶＭ命令の開始点を指しているものを、ＶＰＣとして検出する。具体的には、仮想プログラムカウンタ検出部１２１３は、ＶＰＣの指し先とＶＭ命令ハンドラのアドレスとを照合して、一致するメモリ領域に絞り込む（図１６の（２））。

[0079] ［ディスパッチャ検出部の処理］

次に、ディスパッチャ検出部１２１４の処理について説明する。ディスパッチャ検出部１２１４は、スクリプトエンジンのバイナリを所定の手法で解析することで、ディスパッチャを検出する。図１７は、ディスパッチャ検出部１２１４の処理を説明する図である。

[0080] ディスパッチャ検出部１２１４は、ディスパッチャの検出を行う。ディスパッチャ検出部１２１４は、ＶＭ命令境界検出部１２１２が検出したＶＭ命令の境界を基に、スクリプトエンジンバイナリから各ＶＭ命令部分を切り出す。そして、ディスパッチャ検出部１２１４は、ディスパッチャのコードの類似性は高いとした仮定の基（図１７の（１））、各ＶＭ命令間でコード間の類似度を算出し、全ＶＭ命令間で類似度が高い部分を、ディスパッチャとして検出する。ディスパッチャ検出部１２１４は、ＶＭ命令の後半部で共通的に実行されるコードを、ディスパッチャとして検出できる（図１７の（１））。

[0081] ［コードキャッシュ検出部］

次に、コードキャッシュ検出部１２１５の処理について説明する。図１８は、コードキャッシュ検出部１２１５の処理を説明する図である。

[0082] コードキャッシュ検出部１２１５は、ＶＰＣが指すメモリ領域をコードキャッシュとしてＶＭ実行トレースから検出する（図１８の（１））。

[0083] コードキャッシュ検出部１２１５は、このコードキャッシュを確保したメモリ割り当て関数の呼び出し元のコード箇所を、実行トレースから検出する（図１８の（２））。コードキャッシュ検出部１２１５は、ＶＭ実行トレースのうち、このコード箇所で確保された全てのメモリ領域をコードキャッシュとして検出する（図１８の（３））。

[0084] コードキャッシュ検出部１２１５は、コードキャッシュに書き込みをしているコード箇所を実行トレースから検出する（図１８の（４））。コードキャッシュ検出部１２１５は、ＶＭ実行トレースのうち、このコード箇所による書き込みをコードキャッシュの更新として検出する（図１８（５））。

[0085] ［解析装置の処理手順］

次に、解析装置１０による解析処理の処理手順について説明する。図１９は、実施の形態に係る解析処理の処理手順を示すフローチャートである。

[0086] まず、入力部１１は、テストスクリプト及びスクリプトエンジンバイナリを入力として受け取る（ステップＳ１）。テストスクリプトは、テストスクリプトを含む。

[0087] そして、実行トレース取得部１２１１は、スクリプトエンジンのバイナリを監視しながらテストスクリプトを実行してブランチトレースとメモリアクセストレースを取得する実行トレース取得処理を行う（ステップＳ２）。

[0088] ＶＭ命令境界検出部１２１２は、ＶＭ命令を検出し、ＶＭ命令の境界を検出するＶＭ命令境界検出処理を行う（ステップＳ３）。仮想プログラムカウンタ検出部１２１３は、実行トレースＤＢ１３１に格納された第１のテストスクリプトに対する実行トレースを取り出して解析し、ＶＰＣを発見する仮想プログラムカウンタ検出処理を行う（ステップＳ４）。

[0089] ディスパッチャ検出部１２１４は、スクリプトエンジンバイナリから各ＶＭ命令部分を切り出し、各ＶＭ命令間で類似度が高い部分をディスパッチャとして検出するディスパッチャ検出処理を行う（ステップＳ５）。

[0090] コードキャッシュ検出部１２１５は、実行トレース及びＶＰＣを基に、メモリ割り当て関数の呼び出し元のコード箇所の領域をコードキャッシュとし

て検出し、コード箇所の領域に書き込みをしている領域をコードキャッシュの更新として検出するコードキャッシュ検出処理を行う（ステップＳ６）。

[0091] 記号表検出部１２１６は、第２のテストスクリプト及び第２のテストスクリプトによるメモリアクセストレースを用いて、記号表のアーキテクチャ情報を検出する記号表検出処理を行う（ステップＳ７）。

[0092] ＶＭ実行トレース取得部１２２１は、テストスクリプト及びスクリプトエンジンバイナリを入力として受け付け、スクリプトエンジンバイナリの実行を監視しながら、テストスクリプトを実行することで、ＶＭ実行トレースを取得するＶＭ実行トレース取得処理を行う（ステップＳ８）。

[0093] ＶＭ命令収集部１２２２は、ＶＭ実行トレースからＶＭ命令を収集するＶＭ命令収集処理を行う（ステップＳ９）。

[0094] 記号表ＶＭ命令判定部１２２３は、ＶＭ命令収集部１２２２が収集したＶＭ命令のうち、ＶＭの実行中に、記号表のメモリ領域にアクセスしたＶＭ命令を記号表ＶＭ命令と判定する記号表ＶＭ命令判定処理を行う（ステップＳ１０）。

[0095] 記号表ＶＭ命令解析部１２２４は、解析対象である記号表ＶＭ命令のオペランドを解析し、記号表を指定するオペランド、及び、変数を指定するオペランドを判定する記号表ＶＭ命令解析処理を行う（ステップＳ１１）。

[0096] 出力部１４は、ステップＳ７において検出された記号表、及び、ステップＳ１０，Ｓ１１において解析された記号表ＶＭ命令の情報を出力する（ステップＳ１２）。

[0097] ［実行トレース取得処理の処理手順］

次に、図１９に示す実行トレース取得処理の流れについて説明する。図２０は、図１９に示す実行トレース取得処理の処理手順を示すフローチャートである。

[0098] まず、実行トレース取得部１２１１は、テストスクリプト及びスクリプトエンジンバイナリを入力として受け取る（ステップＳ２１）。そして、実行トレース取得部１２１１は、受け取ったスクリプトエンジンに対して、ブラ

ンチトレースを取得するためのフックを施す（ステップＳ２２）。また、実行トレース取得部１２１１は、受け取ったスクリプトエンジンに対して、メモリアクセストレースを取得するためのフックも施す（ステップＳ２３）。

[0099] そして、実行トレース取得部１２１１は、その状態で受け取ったテストスクリプトをスクリプトエンジンに入力して実行させ（ステップＳ２４）、それによって取得される実行トレースを実行トレースＤＢ１３１に格納する（ステップＳ２５）。

[0100] 実行トレース取得部１２１１は、入力されたテストスクリプトを全て実行し終えているか否かを判定する（ステップＳ２６）。実行トレース取得部１２１１は、入力されたテストスクリプトを全て実行し終えている場合（ステップＳ２６：Ｙｅｓ）、処理を終了する。これに対し、実行トレース取得部１２１１は、入力されたテストスクリプトを全て実行していない場合（ステップＳ２６：Ｎｏ）、ステップＳ２４の第１のテストスクリプトの実行に戻って処理を続ける。

[0101] ［ＶＭ命令境界検出処理の処理手順］

次に、図１９に示すＶＭ命令境界検出処理の流れについて説明する。図２１は、図１９に示すＶＭ命令境界検出処理の処理手順を示すフローチャートである。

[0102] まず、ＶＭ命令境界検出部１２１２は、実行トレースＤＢ１３１から実行トレースを取り出す（ステップＳ３１）。ＶＭ命令境界検出部１２１２は、実行トレースを所定の方法でクラスタリングする（ステップＳ３２）。クラスタリングは、いずれの手法を用いてもよい。

[0103] ＶＭ命令境界検出部１２１２は、実行回数が閾値以上のクラスタをＶＭ命令として検出する（ステップＳ３３）。そして、ＶＭ命令境界検出部１２１２は、ＶＭ命令を構成する連続した命令列の開始点と終了点とを境界とする（ステップＳ３４）。ＶＭ命令境界検出部１２１２は、ＶＭ命令の境界を返り値として出力して（ステップＳ３５）、ＶＭ命令境界検出処理を終了する。

[0104] ［仮想プログラムカウンタ検出処理の処理手順］

次に、図１９に示す仮想プログラムカウンタ検出処理の流れについて説明する。図２２は、図１９に示す仮想プログラムカウンタ検出処理の処理手順を示すフローチャートである。

[0105] まず、仮想プログラムカウンタ検出部１２１３は、実行トレースＤＢ１３１からテストスクリプトによる実行トレースを一つ取り出す（ステップＳ４１）。続いて、仮想プログラムカウンタ検出部１２１３は、実行トレースのうちのメモリアクセストレースに着目し、メモリ読み込み先ごとに読み込み回数を数え上げる（ステップＳ４２）。

[0106] 仮想プログラムカウンタ検出部１２１３は、実行トレースの取得に用いたテストスクリプトを入力として受け取り（ステップＳ４３）、そのテストスクリプトを解析して繰り返しの回数と繰り返される文の数とを取得する（ステップＳ４４）。

[0107] 続いて、仮想プログラムカウンタ検出部１２１３は、実行トレースＤＢ１３１から、繰り返し回数や繰り返される文の数の異なるテストスクリプトによる実行トレースを、さらに一つ取り出す（ステップＳ４５）。そして、仮想プログラムカウンタ検出部１２１３は、メモリアクセストレースに着目し、メモリ読み込み先ごとに読み込み回数を数え上げる（ステップＳ４６）。また、仮想プログラムカウンタ検出部１２１３は、実行トレースの取得に用いたテストスクリプトを入力として受け取り（ステップＳ４７）、テストスクリプトを解析して、繰り返しの回数と繰り返される文の数とを取得する（ステップＳ４８）。

[0108] ここで、仮想プログラムカウンタ検出部１２１３は、繰り返し回数や繰り返される文の増減に比例して読み込み回数が変化するメモリ読み込み先のみに絞り込む（ステップＳ４９）。さらに、仮想プログラムカウンタ検出部１２１３は、ステップＳ４９において絞り込んだメモリ読み込み先を、読み込んだメモリの値が常にＶＭ命令の開始点を指しているものに絞り込む（ステップＳ５０）。

[0109] そして、仮想プログラムカウンタ検出部１２１３は、メモリ読み込み先を一つのみに絞り込めたか否かを判定する（ステップＳ５１）。仮想プログラムカウンタ検出部１２１３は、メモリ読み込み先を一つのみに絞り込めていない場合（ステップＳ５１：Ｎｏ）、ステップＳ４５に戻り、次の実行トレースを一つ取り出して処理を継続する。一方、仮想プログラムカウンタ検出部１２１３は、メモリ読み込み先を一つのみに絞り込めた場合（ステップＳ５１：Ｙｅｓ）、絞り込まれたメモリ読み込み先を仮想プログラムカウンタとしてアーキテクチャ情報ＤＢ１３２に格納して（ステップＳ５２）、処理を終了する。

[0110] ［ディスパッチャ検出処理の処理手順］

次に、図１９に示すディスパッチャ検出処理の流れについて説明する。図２３は、図１９に示すディスパッチャ検出処理の処理手順を示すフローチャートである。

[0111] まず、ディスパッチャ検出部１２１４は、スクリプトエンジンバイナリを入力として受け取る（ステップＳ６１）。ディスパッチャ検出部１２１４は、ＶＭ命令境界検出部１２１２から、ＶＭ命令の境界を受け取る（ステップＳ６２）。

[0112] ディスパッチャ検出部１２１４は、ＶＭ命令境界検出部１２１２から受け取ったＶＭ命令の境界を基に、スクリプトエンジンバイナリから各ＶＭ命令部分を切り出す（ステップＳ６３）。ディスパッチャ検出部１２１４は、各ＶＭ命令間でコード間の類似度を所定の方法で算出する（ステップＳ６４）。類似度の算出手法は、コード間の類似度を算出できる手法であれば、どの手法でもよい。

[0113] ディスパッチャ検出部１２１４は、ステップＳ６４において算出した類似度を基に、全ＶＭ命令間で類似度が高い部分を取り出す（ステップＳ６５）。そして、ディスパッチャ検出部１２１４は、ＶＭ命令の終端部分であるかを判定する（ステップＳ６６）。

[0114] ＶＭ命令の終端部分でない場合（ステップＳ６６：Ｎｏ）、ディスパッチ

ャ検出部１２１４は、ステップＳ６５に戻り処理を続ける。また、ＶＭ命令の終端部分である場合（ステップＳ６６：Ｙｅｓ）、ディスパッチャ検出部１２１４は、取り出した部分をディスパッチャとして出力して（ステップＳ６７）、処理を終了する。

[0115] ［コードキャッシュ検出処理の処理手順］

次に、図１９に示すコードキャッシュ検出処理の流れについて説明する。図２４は、図１９に示すコードキャッシュ検出処理の処理手順を示すフローチャートである。

[0116] コードキャッシュ検出部１２１５は、実行トレース及びＶＭ実行トレースを入力として受け付けると（ステップＳ７１）、ＶＰＣが指すメモリ領域をＶＭ実行トレースから取得する（ステップＳ７２）。ＶＭ実行トレースは、ＶＭ実行トレース取得部１２２１によって取得される。

[0117] コードキャッシュ検出部１２１５は、ステップＳ７２において取得した該メモリ領域を確保したメモリ割り当て関数の呼び出し元のコード箇所を実行トレースから取得する（ステップＳ７３）。コードキャッシュ検出部１２１５は、ＶＭ実行トレースのうち、ステップＳ７３において取得した該コード箇所で確保された全ての領域をコードキャッシュとして検出する（ステップＳ７４）。

[0118] コードキャッシュ検出部１２１５は、コードキャッシュに書き込みをしているコード箇所を実行トレースから取得する（ステップＳ７５）。コードキャッシュ検出部１２１５は、ＶＭ実行トレースのうち、ステップＳ７５において取得した該コード箇所で書き込みされた全ての領域をコードキャッシュの更新として検出する（ステップＳ７６）。コードキャッシュ検出部１２１５は、検出したコードキャッシュ及びその更新箇所を返し（ステップＳ７７）、コードキャッシュ検出処理を終了する。

[0119] ［記号表検出処理の処理手順］

次に、図１９に示す記号表検出処理の流れについて説明する。図２５は、図１９に示す記号表検出処理の処理手順を示すフローチャートである。

[0120] 記号表検出部１２１６は、第２のテストスクリプト及び該第２のテストスクリプトによるメモリアクセストレースを入力として受け付ける（ステップＳ８１）。記号表検出部１２１６は、第２のテストスクリプト内で用いている特徴的な値を抽出する（ステップＳ８２）。

[0121] 記号表検出部１２１６は、ステップＳ８１において入力を受け付けたメモリアクセストレースから、ステップＳ８２において抽出した特徴的な値のマッチングにより、この特徴的な値のメモリ上の格納箇所を検出する（ステップＳ８３）。

[0122] 記号表検出部１２１６は、メモリアクセストレースから、ステップＳ８３において検出した、特徴的な値へ参照している構造体を検出する（ステップＳ８４）。メモリアクセストレースから、どのポインタがどの領域を参照しているかが分かるため、記号表検出部１２１６は、メモリアクセストレースの値間の参照関係を繋げていくことによって、どういった構造を有するかを検出することができる。また、記号表検出部１２１６は、構造体の構造を解析する手法として、既存の手法を採用して、構造体の検出を行ってもよい。

[0123] 記号表検出部１２１６は、複数の値への参照がまとまったメモリ領域を、記号表のメモリ領域上の位置として検出する（ステップＳ８５）。

[0124] 記号表検出部１２１６は、ＡＰＩトレースからメモリ割り当ての関数呼び出しを抽出する（ステップＳ８６）。なお、「メモリ割り当ての関数呼び出し」をしたコード箇所を、「記号表のメモリを割り当てるコード」として検出することで、このコード箇所を監視して、割り当てられたメモリがどのアドレスかを記録すれば、新しく記号表が作成される度に、その記号表のメモリ上の位置が分かるようになる。

[0125] 記号表検出部１２１６は、ＡＰＩトレースから、記号表のあるメモリ領域を確保しているコード、すなわち、記号表のメモリ領域を確保するコード箇所を検出する（ステップＳ８７）。記号表検出部１２１６は、記号表のメモリ領域上の位置とその領域を確保するコード箇所を特定可能にして、記号表として出力する（ステップＳ８８）。

[0126] ［ＶＭ実行トレース取得処理の処理手順］

次に、図１９に示すＶＭ実行トレース取得処理の流れについて説明する。図２６は、図１９に示すＶＭ実行トレース取得処理の処理手順を示すフローチャートである。

[0127] まず、ＶＭ実行トレース取得部１２２１は、テストスクリプト及びスクリプトエンジンバイナリを入力として受け取る（ステップＳ９１）。そして、ＶＭ実行トレース取得部１２２１は、受け取ったスクリプトエンジンに対して、ＶＰＣ及びＶＭオペコードを記録するためのフックを施す（ステップＳ９２）。

[0128] ＶＭ実行トレース取得部１２２１は、その状態で受け取ったテストスクリプトをスクリプトエンジンに入力して実行させ（ステップＳ９３）、それによって取得されるＶＭ実行トレースをＶＭ実行トレースＤＢ１３３に格納する（ステップＳ９４）。

[0129] ＶＭ実行トレース取得部１２２１は、入力されたテストスクリプトを全て実行したか否かを判定する（ステップＳ９５）。ＶＭ実行トレース取得部１２２１は、入力されたテストスクリプトを全て実行し終えている場合（ステップＳ９５：Ｙｅｓ）、処理を終了する。ＶＭ実行トレース取得部１２２１は、入力されたテストスクリプトを全て実行し終えていない場合（ステップＳ９５：Ｎｏ）、ステップＳ９３のテストスクリプトの実行に戻って処理を続ける。

[0130] ［ＶＭ命令収集処理の処理手順］

次に、図１に示すＶＭ命令収集処理の流れについて説明する。図２７は、図１９に示すＶＭ命令収集処理の処理手順を示すフローチャートである。

[0131] ＶＭ命令収集部１２２２は、ＶＰＣ及びディスパッチャを入力として受け付ける（ステップＳ１０１）、インターネット上から多様なスクリプトを取得する（ステップＳ１０２）。ＶＭ命令収集部１２２２は、ＶＰＣ及びディスパッチャを監視しながらスクリプトを実行しＶＭ実行トレースを取得する（ステップＳ１０３）。

[0132] ＶＭ命令収集部１２２２は、ＶＭ実行トレースからＶＭ命令を取得し（ステップＳ１０４）、ＶＭ命令のリストに追加する（ステップＳ１０５）。ＶＭ命令収集部１２２２は、リストにないＶＭ命令が見られた場合（ステップＳ１０６：Ｎｏ）、ステップＳ１０２に戻る。ＶＭ命令収集部１２２２は、リストにないＶＭ命令が見られなくなった場合（ステップＳ１０６：Ｙｅｓ）、ＶＭ命令のリストを返し（ステップＳ１０７）、ＶＭ命令収集処理を終了する。

[0133] ［記号表ＶＭ命令判定処理の処理手順］

次に、図１９に示す記号表ＶＭ命令判定処理の流れについて説明する。図２８は、図１９に示す記号表ＶＭ命令判定処理の処理手順を示すフローチャートである。

[0134] 記号表ＶＭ命令判定部１２２３は、ＶＭ実行トレースＤＢ１３３から、ＶＭ実行トレース及びメモリアクセストレースを入力として受け付ける（ステップＳ１１１）。記号表ＶＭ命令判定部１２２３は、記号表検出処理において検出された、記号表のメモリ領域上の位置を入力として受け付ける（ステップＳ１１２）。

[0135] 記号表ＶＭ命令判定部１２２３は、ＶＭ実行トレースから、ＶＭ命令を一つ取り出す（ステップＳ１１３）。ＶＭ命令は、ＶＰＣの値及びＶＭオペコードの値からなる。記号表ＶＭ命令判定部１２２３は、取り出した、該ＶＭ命令の実行中にアクセスしたメモリ領域を確認し（ステップＳ１１４）、記号表のメモリ領域にアクセスしたか否かを判定する（ステップＳ１１５）。

[0136] 記号表のメモリ領域にアクセスしていない場合（ステップＳ１１５：Ｎｏ）、記号表ＶＭ命令判定部１２２３は、ステップＳ１１３において取り出したＶＭ命令は記号表ＶＭ命令ではないと判定する（ステップＳ１１６）。

[0137] 記号表のメモリ領域にアクセスした場合（ステップＳ１１５：Ｙｅｓ）、記号表ＶＭ命令判定部１２２３は、ステップＳ１１３において取り出したＶＭ命令が、読み込み・書き込みのうち、読み込みであったか否かを判定する（ステップＳ１１７）。

[0138] 読み込みであった場合（ステップＳ１１７：Ｙｅｓ）、記号表ＶＭ命令判定部１２２３は、ステップＳ１１３において取り出したＶＭ命令が、読み込みの記号表ＶＭ命令であると判定する（ステップＳ１１８）。

[0139] 一方、読み込みではなかった場合（ステップＳ１１７：Ｎｏ）、記号表ＶＭ命令判定部１２２３は、ステップＳ１１３において取り出したＶＭ命令が、書き込みの記号表ＶＭ命令であると判定する（ステップＳ１１９）。

[0140] ステップＳ１１６、ステップＳ１１８、または、ステップＳ１１９処理後、記号表ＶＭ命令判定部１２２３は、ＶＭ実行トレースの全てのＶＭ命令について記号表ＶＭ命令であるか否かを確認したかを判定する（ステップＳ１２０）。

[0141] 全てのＶＭ命令について記号表ＶＭ命令であるか否かを確認していない場合（ステップＳ１２０：Ｎｏ）、記号表ＶＭ命令判定部１２２３は、ＶＭ実行トレースから次のＶＭ命令を取り出し（ステップＳ１２１）、ステップＳ１１４に進み、処理を続ける。

[0142] 全てのＶＭ命令について記号表ＶＭ命令であるか否かを確認した場合（ステップＳ１２０：Ｙｅｓ）、記号表ＶＭ命令判定部１２２３は、記号表ＶＭ命令と判定されたＶＭ命令のオペコードのリストを出力して（ステップＳ１２２）、記号表ＶＭ命令判定処理を終了する。

[0143] ［記号表ＶＭ命令解析処理の処理手順］

次に、図１９に示す記号表ＶＭ命令解析処理の流れについて説明する。図２９及び図３０は、図１９に示す記号表ＶＭ命令解析処理の処理手順を示すフローチャートである。

[0144] 記号表ＶＭ命令解析部１２２４は、第３のテストスクリプトによるメモリアクセストレース及びＶＭ実行トレースを入力として受け付ける（ステップＳ１３１）。

[0145] 記号表ＶＭ命令解析部１２２４は、ステップＳ１３１において入力を受け付けたＶＭ実行トレースから２つの記号表ＶＭ命令の組み合わせを取り出す（ステップＳ１３２）。記号表ＶＭ命令解析部１２２４は、記号表ＶＭ命令

判定処理において出力された記号表ＶＭ命令と判定されたＶＭ命令のリストを参照して、記号表ＶＭ命令の組み合わせを取り出す。

[0146] 記号表ＶＭ命令解析部１２２４は、ステップＳ１３１において入力を受け付けたメモリアクセストレースから、それぞれの記号表ＶＭ命令の実行中にアクセスされた部分を取り出す（ステップＳ１３３）。

[0147] 記号表ＶＭ命令解析部１２２４は、ステップＳ１３３において取り出した部分を比較して差分となる箇所を抽出する（ステップＳ１３４）。

[0148] 記号表ＶＭ命令解析部１２２４は、差分となる箇所がオペランドの位置に存在するか否かを判定する（ステップＳ１３５）。

[0149] 差分となる箇所がオペランドの位置に存在しない場合（ステップＳ１３５：Ｎｏ）、記号表ＶＭ命令解析部１２２４は、ステップＳ１３１において入力を受け付けたＶＭ実行トレースの全ての記号表ＶＭ命令を処理したか否かを判定する（ステップＳ１３６）。

[0150] 記号表ＶＭ命令解析部１２２４は、ＶＭ実行トレースの全ての記号表ＶＭ命令を処理していない場合（ステップＳ１３６：Ｎｏ）、ステップＳ１３１において入力を受け付けたＶＭ実行トレースから、次の記号表ＶＭ命令の組み合わせを取り出す（ステップＳ１３７）。そして、記号表ＶＭ命令解析部１２２４は、ステップＳ１３３の処理に進む。

[0151] 差分となる箇所がオペランドの位置に存在する場合（ステップＳ１３５：Ｙｅｓ）、記号表ＶＭ命令解析部１２２４は、差分となる箇所を、記号表を指定するオペランドと判定する（ステップＳ１３８）。

[0152] 記号表ＶＭ命令解析部１２２４は、ステップＳ１３１において入力を受け付けたＶＭ実行トレースの全ての記号表ＶＭ命令を処理していない場合（ステップＳ１３６：Ｙｅｓ）、または、ステップＳ１３８処理後、第４のテストスクリプトによるメモリアクセストレース及びＶＭ実行トレースを入力として受け付ける（ステップＳ１３９）。

[0153] 記号表ＶＭ命令解析部１２２４は、ステップＳ１３９において入力を受け付けたＶＭ実行トレースから２つの記号表ＶＭ命令の組み合わせを取り出す

（ステップＳ１４０）。記号表ＶＭ命令解析部１２２４は、ステップＳ１３９において入力を受け付けたメモリアクセストレースから、それぞれの記号表ＶＭ命令の実行中にアクセスされた部分を取り出す（ステップＳ１４１）。

[0154] 記号表ＶＭ命令解析部１２２４は、ステップＳ１４１において取り出した部分を比較して差分となる箇所を抽出する（ステップＳ１４２）。

[0155] 記号表ＶＭ命令解析部１２２４は、差分となる箇所がオペランドの位置に存在するか否かを判定する（ステップＳ１４３）。

[0156] 差分となる箇所がオペランドの位置に存在しない場合（ステップＳ１４３：Ｎｏ）、記号表ＶＭ命令解析部１２２４は、ステップＳ１３９において入力を受け付けたＶＭ実行トレースの全ての記号表ＶＭ命令を処理したか否かを判定する（ステップＳ１４４）。

[0157] 記号表ＶＭ命令解析部１２２４は、ＶＭ実行トレースの全ての記号表ＶＭ命令を処理していない場合（ステップＳ１４４：Ｎｏ）、ステップＳ１３９において入力を受け付けたＶＭ実行トレースから次の記号表ＶＭ命令の組み合わせを取り出す（ステップＳ１４５）。そして、記号表ＶＭ命令解析部１２２４は、ステップＳ１４１の処理に進む。

[0158] 差分となる箇所がオペランドの位置に存在する場合（ステップＳ１４３：Ｙｅｓ）、記号表ＶＭ命令解析部１２２４は、差分の箇所を、変数を指定するオペランドと判定する（ステップＳ１４６）。

[0159] 全ての記号表ＶＭ命令を処理した場合（ステップＳ１４４：Ｙｅｓ）、または、ステップＳ１４６の処理後、記号表ＶＭ命令解析部１２２４は、ステップＳ１３８，Ｓ１４６の判定結果を基に、記号表を指定するオペランドの情報、及び、変数を指定するオペランドの情報を出力する（ステップＳ１４７）。

[0160] ［実施の形態の効果］

以上のように、本実施の形態に係る解析装置１０は、スクリプトエンジンのバイナリを監視しながらテストスクリプトを実行して、ブランチトレース

とメモリアクセストレースとを実行トレースとして取得する。解析装置１０は、実行トレースを基に、スクリプトエンジンのＶＭを解析し、ＶＰＣ、ディスパッチャ、コードキャッシュ、記号表のアーキテクチャ情報を取得する。

[0161] 具体的には、解析装置１０は、特徴的な値が用いられる第２のテストスクリプトのメモリアクセストレースから、特徴的な値のメモリ領域上の格納箇所と該特徴的な値への参照元を基に、記号表のメモリ領域上の位置と、特徴的な値へ参照する構造体とを検出する。そして、解析装置１０は、ＡＰＩトレースから、記号表のメモリ領域を確保するコード箇所を検出する。

[0162] このように、解析装置１０は、記号表のアーキテクチャ情報として、記号表のメモリ領域上の位置と、記号表のメモリ領域を確保するコード箇所と、記号表の構造とを取得する。

[0163] そして、解析装置１０は、仮想機械の命令の体系である命令セットアーキテクチャを解析する。たとえば、解析装置１０は、スクリプトエンジンのＶＭの解析結果に基づいて、ＶＭ命令を収集し、収集したＶＭ命令のうち記号表ＶＭ命令を判定し、記号表ＶＭ命令のアクセス先を解析する。

[0164] 具体的には、解析装置１０は、ＶＰＣ及びディスパッチャを監視しながらテストスクリプトを実行して、ＶＭ実行トレースを取得する。解析装置１０は、このＶＭ実行トレースを解析することで、ＶＭ命令を収集するとともに、記号表にアクセスするＶＭ命令の判定とそのアクセス先の解析を行う。

[0165] 具体的には、解析装置１０は、ＶＭ実行トレースから取り出した２つの記号表ＶＭ命令の組み合わせについて、メモリアクセストレースにおける、組み合わせの各記号表ＶＭ命令の実行中にアクセスされた部分の差分となる箇所がオペランドの位置に存在する場合、差分となる箇所を、記号表ＶＭ命令のオペランドであると判定する。解析装置１０は、この判定により、記号表ＶＭ命令のオペコードのリスト、記号表を指定するオペランド、及び、記号表を指定するオペランドの情報、及び、変数を指定するオペランドの情報を取得する。

[0166] このように、解析装置１０は、ＶＭの内部仕様が未知のスクリプトエンジンに対しても、実行トレースおよびＶＭ実行トレースの取得に基づく解析により各種アーキテクチャ情報を検出し、人手でのリバースエンジニアリングを要することなく、記号表の情報を取得できる。

[0167] また、解析装置１０は、多様なスクリプトエンジンに対して、テストスクリプトさえ用意すれば自動で記号表を解析できるため、個別の設計や実行を要することなく、変数情報の取得を実現できる。

[0168] これにより、解析装置１０は、様々なスクリプト言語で作成されたスクリプトに対しても、変数の解析が可能となり、より詳細な挙動の把握を実現できるようになる。このように、本実施の形態に係る解析装置１０によれば、スクリプトエンジンを解析し、記号表の情報を取得することにより、多種多様なスクリプト言語のスクリプトエンジンに対して、変数情報の解明を実現できる。

[0169] 上述したように、本実施の形態に係る解析装置１０は、多種多様なスクリプトエンジンにおける記号表の情報の取得に有用であり、デバッガなどの解析支援機能の不在やVMの内部仕様が未知であるために変数情報の解析が難しいスクリプトに対しても、解析を実現することに適している。

[0170] このため、本実施の形態に係る解析装置、解析方法及び解析プログラムを用いて、様々なスクリプトエンジンの記号表の情報を取得することで、スクリプトの変数を解析するツールの実現に生かすことが可能である。

[0171] ［実施形態のシステム構成について］

図３に示す解析装置１０の各構成要素は機能概念的なものであり、必ずしも物理的に図示のように構成されていることを要しない。すなわち、解析装置１０の機能の分散及び統合の具体的形態は図示のものに限られず、その全部または一部を、各種の負荷や使用状況などに応じて、任意の単位で機能的または物理的に分散または統合して構成することができる。

[0172] また、解析装置１０においておこなわれる各処理は、全部または任意の一部が、ＣＰＵ及びＣＰＵにより解析実行されるプログラムにて実現されても

よい。また、解析装置１０においておこなわれる各処理は、ワイヤードロジックによるハードウェアとして実現されてもよい。

[0173] また、実施の形態において説明した各処理のうち、自動的におこなわれるものとして説明した処理の全部または一部を手動的に行うこともできる。もしくは、手動的におこなわれるものとして説明した処理の全部または一部を公知の方法で自動的に行うこともできる。この他、上述及び図示の処理手順、制御手順、具体的名称、各種のデータやパラメータを含む情報については、特記する場合を除いて適宜変更することができる。

[0174] ［プログラム］

図３１は、プログラムが実行されることにより、解析装置１０が実現されるコンピュータの一例を示す図である。コンピュータ１０００は、例えば、メモリ１０１０、ＣＰＵ１０２０を有する。また、コンピュータ１０００は、ハードディスクドライブインタフェース１０３０、ディスクドライブインタフェース１０４０、シリアルポートインタフェース１０５０、ビデオアダプタ１０６０、ネットワークインタフェース１０７０を有する。これらの各部は、バス１０８０によって接続される。

[0175] メモリ１０１０は、ＲＯＭ１０１１及びＲＡＭ１０１２を含む。ＲＯＭ１０１１は、例えば、ＢＩＯＳ（Basic Input Output System）等のブートプログラムを記憶する。ハードディスクドライブインタフェース１０３０は、ハードディスクドライブ１０９０に接続される。ディスクドライブインタフェース１０４０は、ディスクドライブ１１００に接続される。例えば磁気ディスクや光ディスク等の着脱可能な記憶媒体が、ディスクドライブ１１００に挿入される。シリアルポートインタフェース１０５０は、例えばマウス１１１０、キーボード１１２０に接続される。ビデオアダプタ１０６０は、例えばディスプレイ１１３０に接続される。

[0176] ハードディスクドライブ１０９０は、例えば、ＯＳ１０９１、アプリケーションプログラム１０９２、プログラムモジュール１０９３、プログラムデータ１０９４を記憶する。すなわち、解析装置１０の各処理を規定するプロ

グラムは、コンピュータ１０００により実行可能なコードが記述されたプログラムモジュール１０９３として実装される。プログラムモジュール１０９３は、例えばハードディスクドライブ１０９０に記憶される。例えば、解析装置１０における機能構成と同様の処理を実行するためのプログラムモジュール１０９３が、ハードディスクドライブ１０９０に記憶される。なお、ハードディスクドライブ１０９０は、ＳＳＤ（Solid State Drive）により代替されてもよい。

[0177] また、上述した実施の形態の処理で用いられる設定データは、プログラムデータ１０９４として、例えばメモリ１０１０やハードディスクドライブ１０９０に記憶される。そして、ＣＰＵ１０２０が、メモリ１０１０やハードディスクドライブ１０９０に記憶されたプログラムモジュール１０９３やプログラムデータ１０９４を必要に応じてＲＡＭ１０１２に読み出して実行する。

[0178] なお、プログラムモジュール１０９３やプログラムデータ１０９４は、ハードディスクドライブ１０９０に記憶される場合に限らず、例えば着脱可能な記憶媒体に記憶され、ディスクドライブ１１００等を介してＣＰＵ１０２０によって読み出されてもよい。あるいは、プログラムモジュール１０９３及びプログラムデータ１０９４は、ネットワーク（ＬＡＮ（Local Area Network）、ＷＡＮ（Wide Area Network）等）を介して接続された他のコンピュータに記憶されてもよい。そして、プログラムモジュール１０９３及びプログラムデータ１０９４は、他のコンピュータから、ネットワークインタフェース１０７０を介してＣＰＵ１０２０によって読み出されてもよい。

[0179] 以上、本発明者によってなされた発明を適用した実施の形態について説明したが、本実施の形態による本発明の開示の一部をなす記述及び図面により本発明は限定されることはない。すなわち、本実施の形態に基づいて当業者等によりなされる他の実施の形態、実施例及び運用技術等はすべて本発明の範疇に含まれる。

符号の説明

[0180] １０　解析装置

１１　入力部

１２　制御部

１３　記憶部

１４　出力部

１２１　仮想機械解析部

１２２　命令セットアーキテクチャ解析部

１３１　実行トレースデータベース（ＤＢ）

１３２　アーキテクチャ情報ＤＢ

１３３　ＶＭ実行トレースＤＢ

１２１１　実行トレース取得部

１２１２　ＶＭ命令境界検出部

１２１３　仮想プログラムカウンタ検出部

１２１４　ディスパッチャ検出部

１２１５　コードキャッシュ検出部

１２１６　記号表検出部

１２２１　ＶＭ実行トレース取得部

１２２２　ＶＭ命令収集部

１２２３　記号表ＶＭ命令判定部

１２２４　記号表ＶＭ命令解析部

請求の範囲

[請求項1] スクリプトエンジンのバイナリを監視しながらテストスクリプトを実行して取得した実行トレースを基に、前記スクリプトエンジンの仮想機械を解析し、解析結果を基に、変数に関する情報を保持する記号表を検出する第１の解析部と、

前記スクリプトエンジンの仮想機械を解析し、解析結果に基づいて、仮想機械命令を収集し、前記解析結果に基づいて、収集した前記仮想機械命令を解析する第２の解析部と、

を有することを特徴とする解析装置。

[請求項2] 前記第２の解析部は、前記スクリプトエンジンの仮想機械の解析結果に基づいて、収集した前記仮想機械命令のうち、前記記号表にアクセスする記号表仮想機械命令を判定し、前記記号表仮想機械命令のアクセス先を解析することを特徴とする請求項１に記載の解析装置。

[請求項3] 前記第１の解析部は、

テストスクリプトの実行時の条件を変えて、実行時のメモリの読み込みまたは書き込みの操作の種類と、操作対象のメモリアドレスと値とを記録するメモリアクセストレースを、前記実行トレースとして取得する第１の取得部と、

前記実行トレースをクラスタリングして、各仮想機械命令の境界を検出する第１の検出部と、

メモリの読み込み回数に着目した差分実行解析と前記第１の検出部によって検出された各仮想機械命令の境界とを用いて複数の前記実行トレースを解析し、次に実行される前記仮想機械の命令を指し示す変数である仮想プログラムカウンタを検出する第２の検出部と、

前記第１の検出部によって検出された各仮想機械命令の境界を基に、前記スクリプトエンジンのバイナリを解析し、ディスパッチャを検出する第３の検出部と、

前記実行トレース、前記仮想プログラムカウンタ、及び、前記仮想

機械において実行された実行トレースである仮想機械実行トレースを基に、前記仮想機械実行トレースから、実行される仮想機械命令が保存されるキャッシュであるコードキャッシュを検出する第４の検出部と、

特徴的な値が用いられる第２のテストスクリプトの前記メモリアクセストレースから、前記特徴的な値のメモリ領域上の格納箇所と該特徴的な値への参照元を基に、前記記号表のメモリ領域上の位置と、前記特徴的な値へ参照する構造体とを検出し、ＡＰＩトレースから、前記記号表のメモリ領域を確保するコード箇所を検出する第５の検出部と、

を有することを特徴とする請求項２に記載の解析装置。

[請求項4] 前記第２の解析部は、

前記仮想機械実行トレースを取得する第２の取得部と、

前記仮想プログラムカウンタ及び前記ディスパッチャを監視しながらテストスクリプトを実行して前記仮想機械実行トレースを取得し、前記仮想機械実行トレースから仮想機械命令を収集する第１の収集部と、

前記第１の収集部が収集した仮想機械命令のうち、前記仮想機械命令の実行中に前記記号表のメモリ領域にアクセスした仮想機械命令を記号表仮想機械命令と判定する第１の判定部と、

前記仮想機械実行トレースから取り出した２つの前記記号表仮想機械命令の組み合わせについて、前記メモリアクセストレースにおける、前記組み合わせの各記号表仮想機械命令の実行中にアクセスされた部分の差分となる箇所がオペランドの位置に存在する場合、前記差分となる箇所を、前記記号表仮想機械命令のオペランドであると判定する第２の解析部と、

を有することを特徴とする請求項３に記載の解析装置。

[請求項5] 前記第２の解析部は、

異なる２つのスコープの変数にアクセスする第３の前記テストスクリプトによる前記仮想機械実行トレースから取り出した２つの前記記号表仮想機械命令の第１の組み合わせについて、前記第３のテストスクリプトによるメモリアクセストレースから、前記第１の組み合わせの各記号表仮想機械命令の実行中にアクセスされた部分をそれぞれ取り出し、取り出した２つの部分の差分となる箇所がオペランドの位置に存在する場合、前記差分となる箇所を、前記記号表を指定するオペランドと判定することを特徴とする請求項４に記載の解析装置。

[請求項6] 前記第２の解析部は、

同じスコープ内の異なる２つの変数にアクセスする第４の前記テストスクリプトによる前記仮想機械実行トレースから取り出した２つの前記記号表仮想機械命令の第２の組み合わせについて、前記第４のテストスクリプトによるメモリアクセストレースから、前記第２の組み合わせの各記号表仮想機械命令の実行中にアクセスされた部分をそれぞれ取り出し、取り出した２つの部分の差分となる箇所が、オペランドの位置に存在する場合、前記差分となる箇所を、前記変数を指定するオペランドと判定することを特徴とする請求項４に記載の解析装置。

[請求項7] 解析装置が実行する解析方法であって、

スクリプトエンジンのバイナリを監視しながらテストスクリプトを実行して取得した実行トレースを基に、前記スクリプトエンジンの仮想機械を解析し、解析結果を基に、変数に関する情報を保持する記号表を検出する第１の解析工程と、

前記スクリプトエンジンの仮想機械を解析し、解析結果に基づいて、仮想機械命令を収集し、前記解析結果に基づいて、収集した前記仮想機械命令を解析する第２の解析工程と、

を含んだことを特徴とする解析方法。

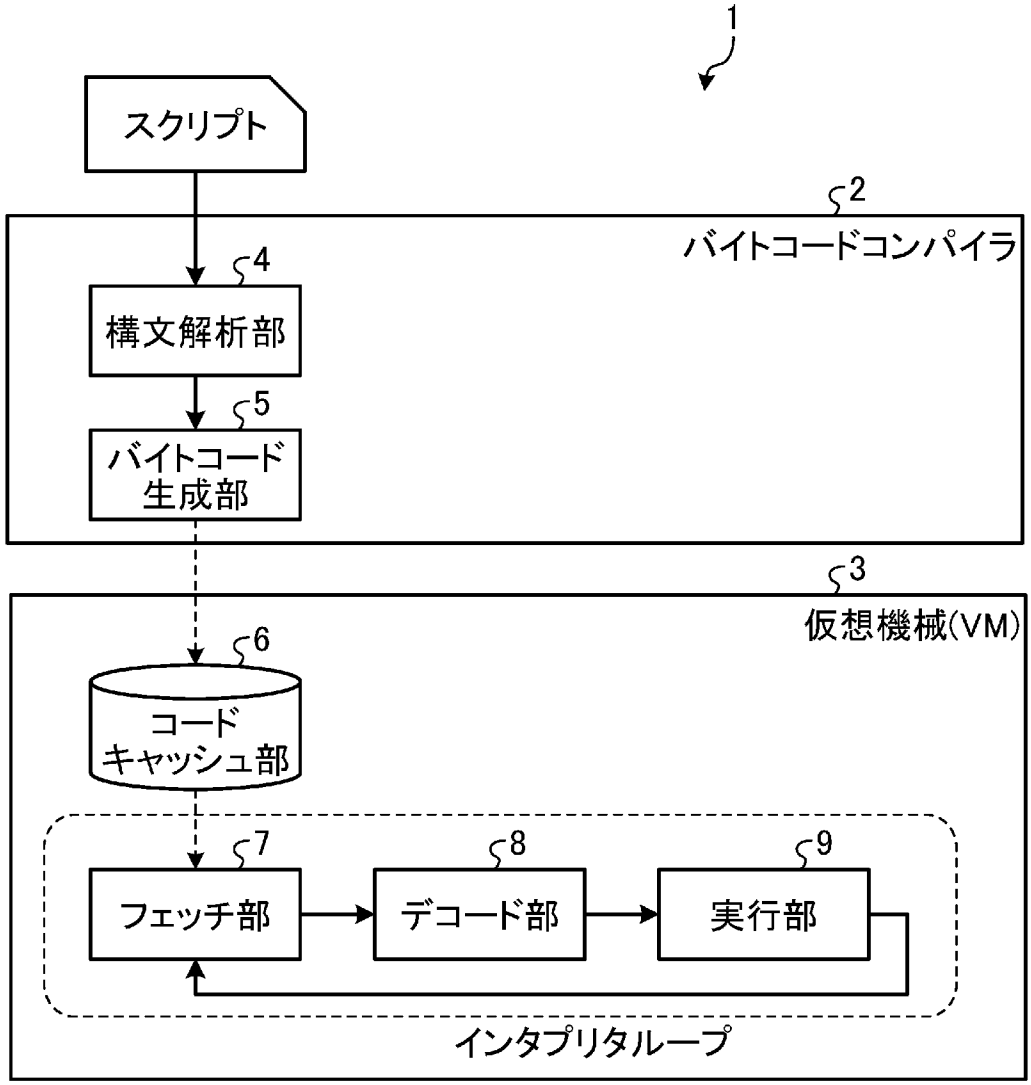
[請求項8] スクリプトエンジンのバイナリを監視しながらテストスクリプトを

実行して取得した実行トレースを基に、前記スクリプトエンジンの仮想機械を解析し、解析結果を基に、変数に関する情報を保持する記号表を検出する第１の解析ステップと、

前記スクリプトエンジンの仮想機械を解析し、解析結果に基づいて、仮想機械命令を収集し、前記解析結果に基づいて、収集した前記仮想機械命令を解析する第２の解析ステップと、

をコンピュータに実行させるための解析プログラム。

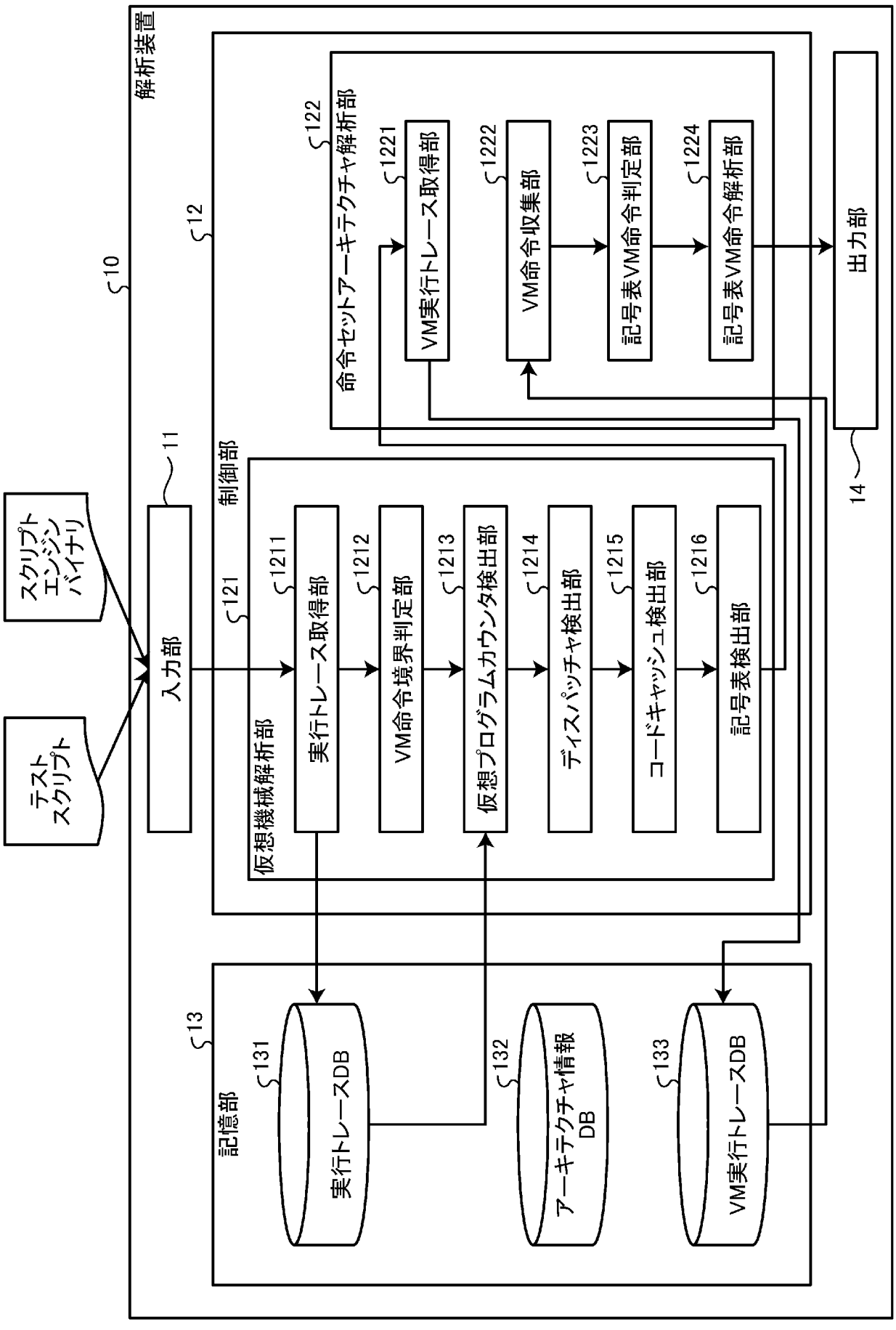
[図1]



[図2]

```
1 while (True)
2       opcode = code_cache[vpc++]
3       switch (opcode)
4           case VMOP_1:
5               arg1 = pop_stack()
6               var1 = lookup_symbol_table(arg1)
7               …
```

[図3]


解析装置
10
12
制御部
11
入力部
121
仮想機械解析部
1211
実行トレース取得部
1212
VM命令境界判定部
1213
仮想プログラムカウンタ検出部
1214
ディスパッチャ検出部
1215
コードキャッシュ検出部
1216
記号表検出部
122
命令セットアーキテクチャ解析部
1221
VM実行トレース取得部
1222
VM命令収集部
1223
記号表VM命令判定部
1224
記号表VM命令解析部
14
出力部
スクリプトエンジンバイナリ
テストスクリプト
13
記憶部
131
実行トレースDB
132
アーキテクチャ情報DB
133
VM実行トレースDB

[図4]

[図5]

[図6]

```
def func1():
    a = 1

def func2():
    b = 2

func1()
func2()
```

[図7]

```
a = 1
b = 2
```

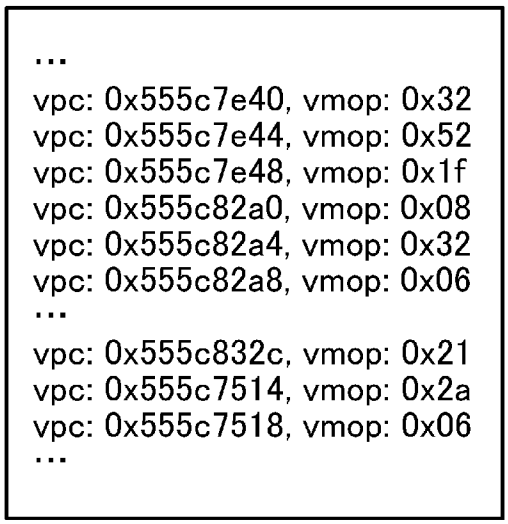
[図8]

```
def func():
    a = 1
    b = 2
```

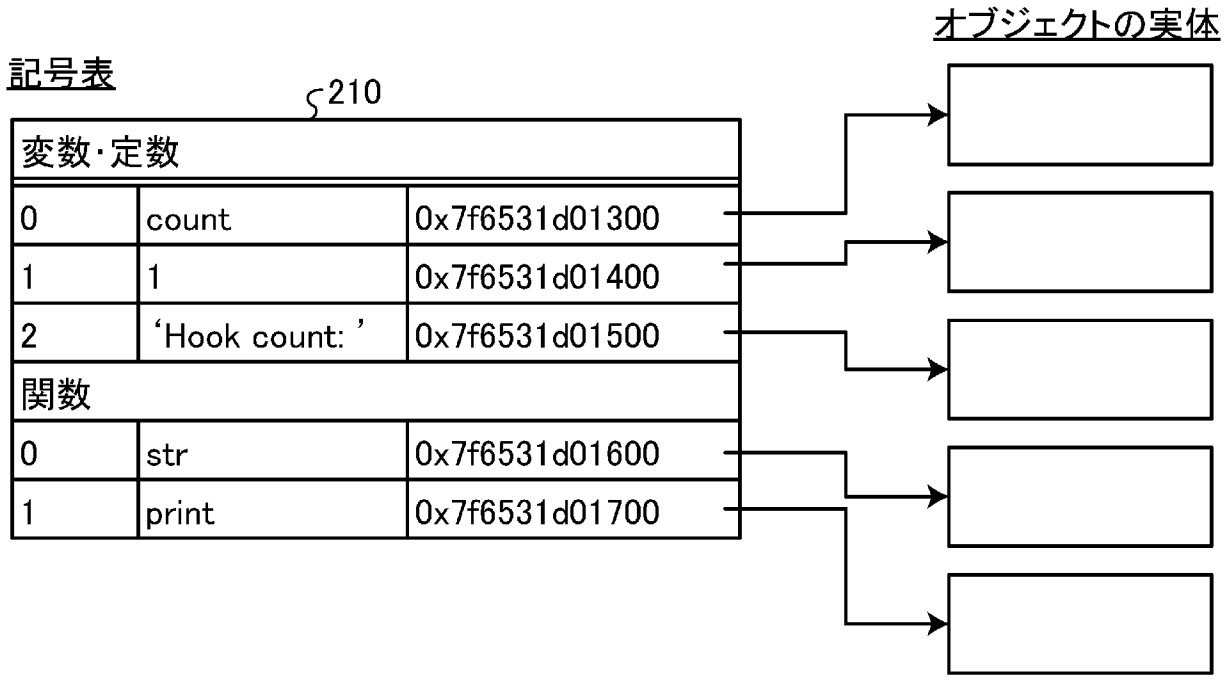
[図9]

```
…
trace: branch, type: jmp, src: 0x6d375f2e, dst: 0x6d39022f
trace: branch, type: jmp, src: 0x6d375f36, dst: 0x6d39023b
trace: branch, type: jmp, src: 0x6d361c86, dst: 0x6d39aee0
trace: branch, type: ret, src: 0x6d361c8c, dst: 0x6d375f4c
trace: branch, type: ret, src: 0x6d375f4f, dst: 0x6d375ee1
trace: branch, type: jmp, src: 0x6d375ee3, dst: 0x6d38fb18
trace: branch, type: ret, src: 0x6d38fb1c, dst: 0x6d375cc9
trace: branch, type: jmp, src: 0x6d375ccb, dst: 0x6d39500a
trace: branch, type: jmp, src: 0x6d375cd6, dst: 0x6d39508a
trace: branch, type: call, src: 0x6d375ceb, dst: 0x75e45435
trace: memaccess, type: read, target: 0x7fff0268, value: 0x0
trace: memaccess, type: write, target: 0x7fffc520, value: 0x55553268
trace: memaccess, type: read, target: 0x55554048, value: 0x31
…
```

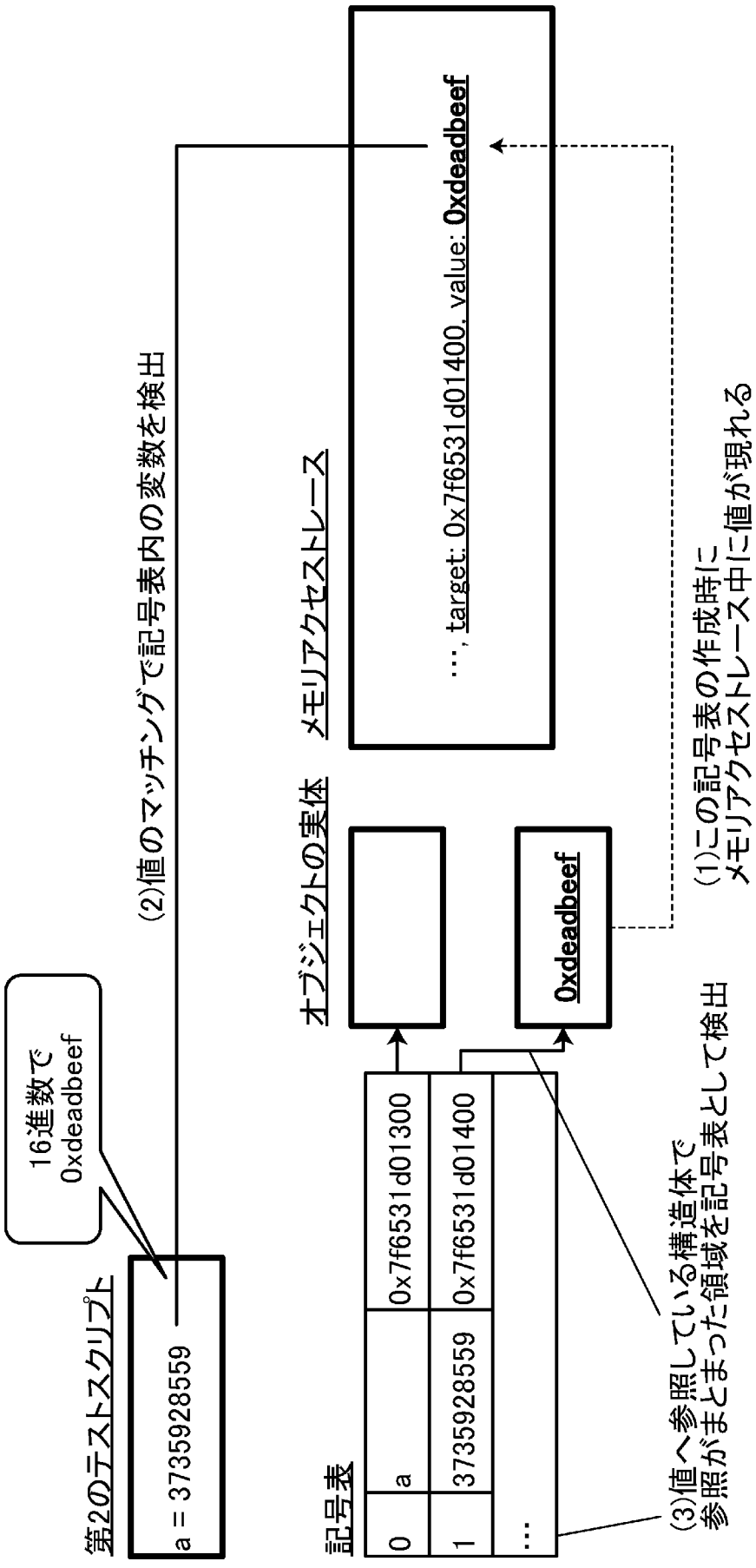
[図10]


…
vpc: 0x555c7e40, vmop: 0x32
vpc: 0x555c7e44, vmop: 0x52
vpc: 0x555c7e48, vmop: 0x1f
vpc: 0x555c82a0, vmop: 0x08
vpc: 0x555c82a4, vmop: 0x32
vpc: 0x555c82a8, vmop: 0x06
…
vpc: 0x555c832c, vmop: 0x21
vpc: 0x555c7514, vmop: 0x2a
vpc: 0x555c7518, vmop: 0x06
…

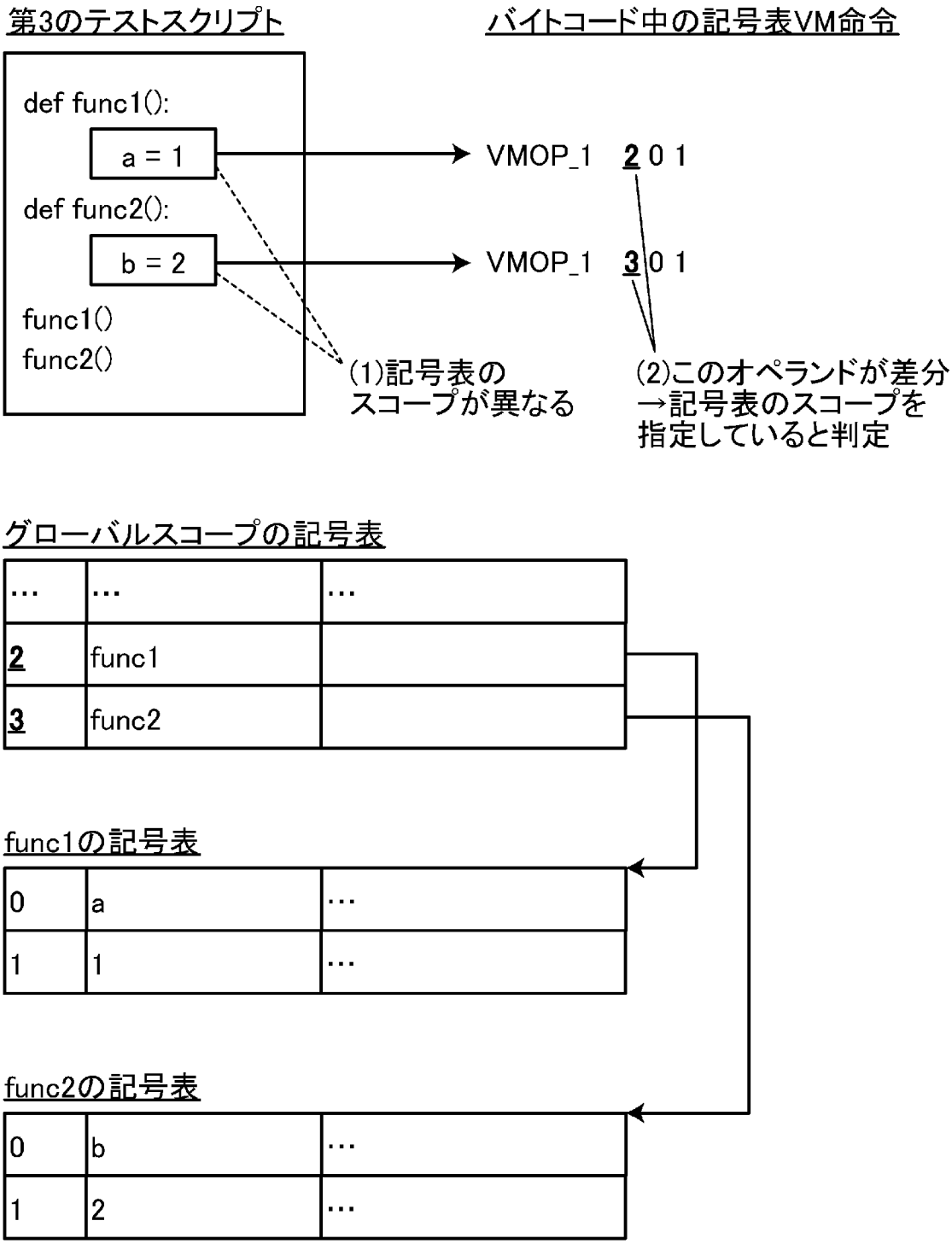
[図11]


オブジェクトの実体
記号表
210
変数・定数
0
count
0x7f6531d01300
1
1
0x7f6531d01400
2
‘Hook count: ’
0x7f6531d01500
関数
0
str
0x7f6531d01600
1
print
0x7f6531d01700

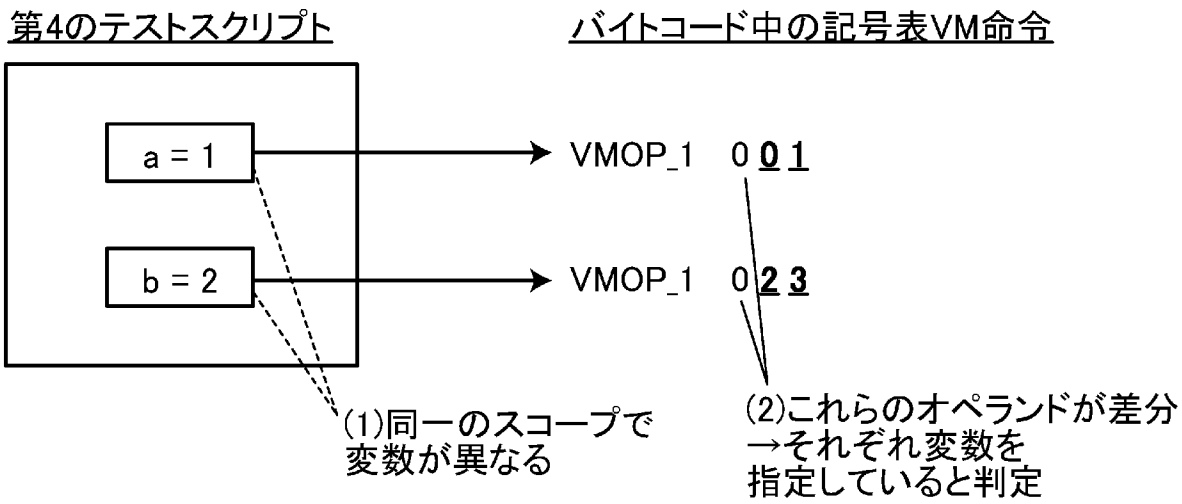
[図12]


第2のテストスクリプト
a = 3735928559
16進数で
0xdeadbeef
(2)値のマッチングで記号表内の変数を検出
記号表
0 a 0x7f6531d01300
1 3735928559 0x7f6531d01400
...
オブジェクトの実体
メモリアクセストレース
..., target: 0x7f6531d01400, value: 0xdeadbeef
0xdeadbeef
(1)この記号表の作成時に
メモリアクセストレース中に値が現れる
(3)値へ参照している構造体で
参照がまとまった領域を記号表として検出

[図13]


第3のテストスクリプト
バイトコード中の記号表VM命令
def func1():
a = 1
def func2():
b = 2
func1()
func2()
VMOP_1 2 0 1
VMOP_1 3 0 1
(1)記号表の
スコープが異なる
(2)このオペランドが差分
→記号表のスコープを
指定していると判定
グローバルスコープの記号表
…
…
…
2
func1
3
func2
func1の記号表
0
a
…
1
1
…
func2の記号表
0
b
…
1
2
…

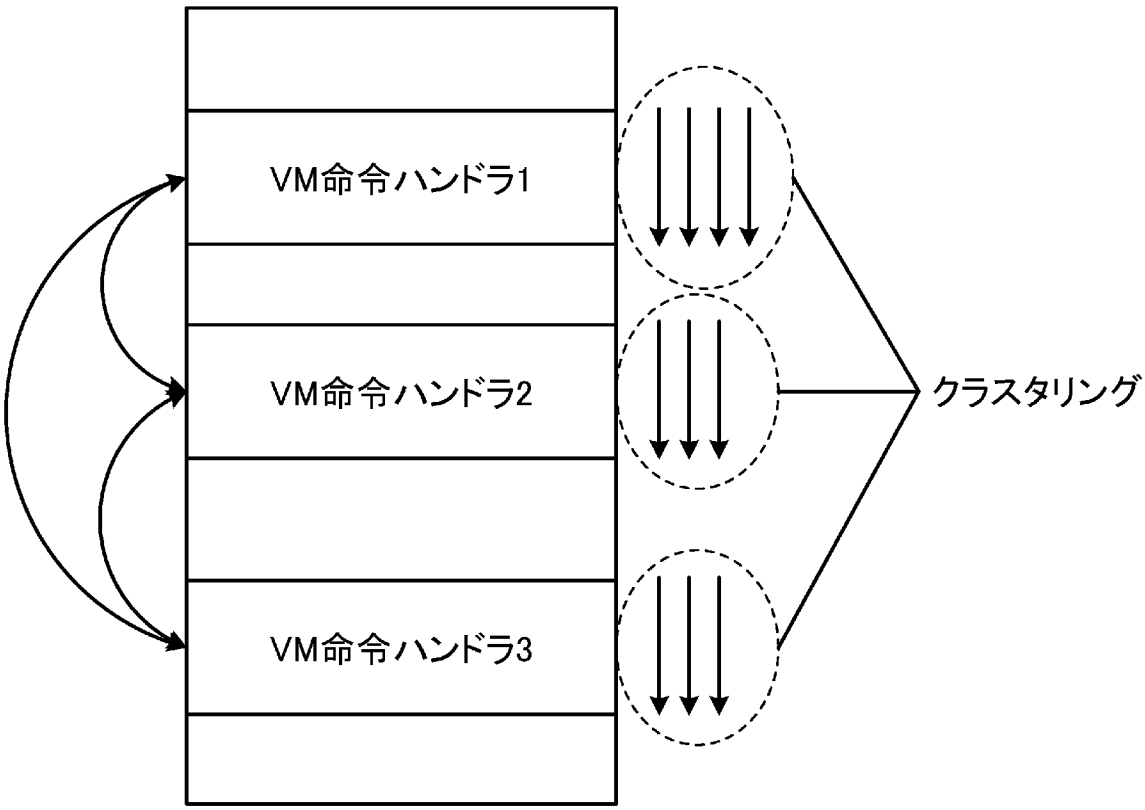
[図14]



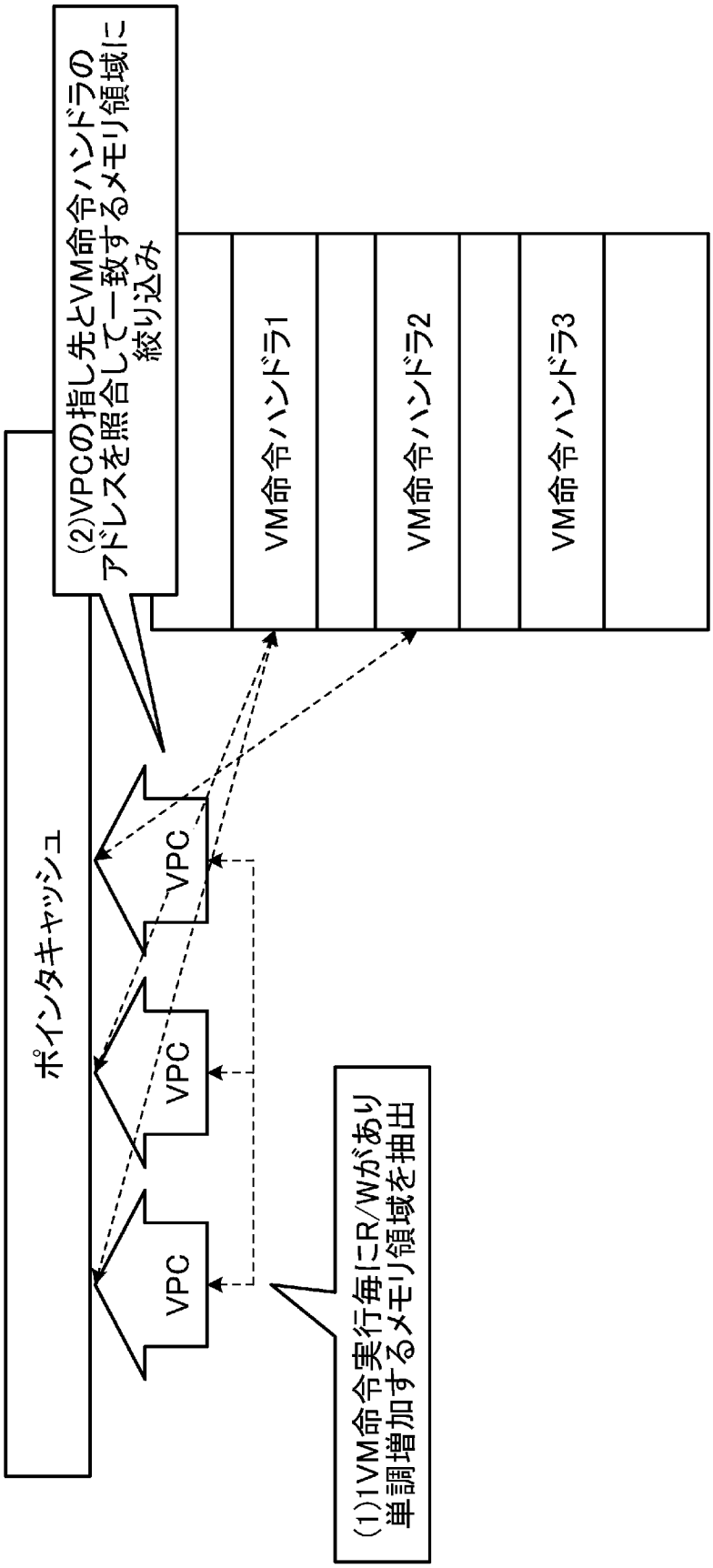
グローバルスコープの記号表

0	a	…
1	1	…
2	b	…
3	2	…

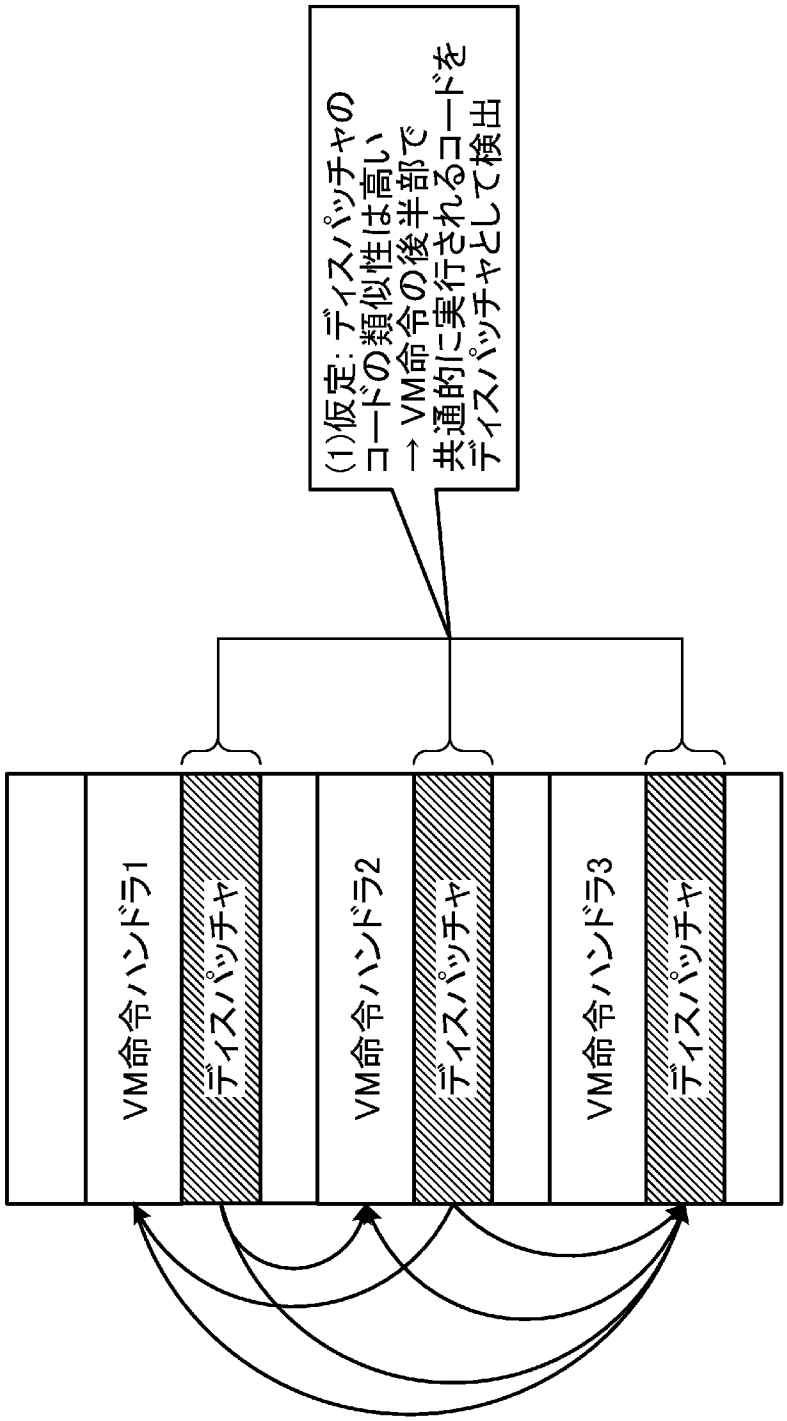
[図15]



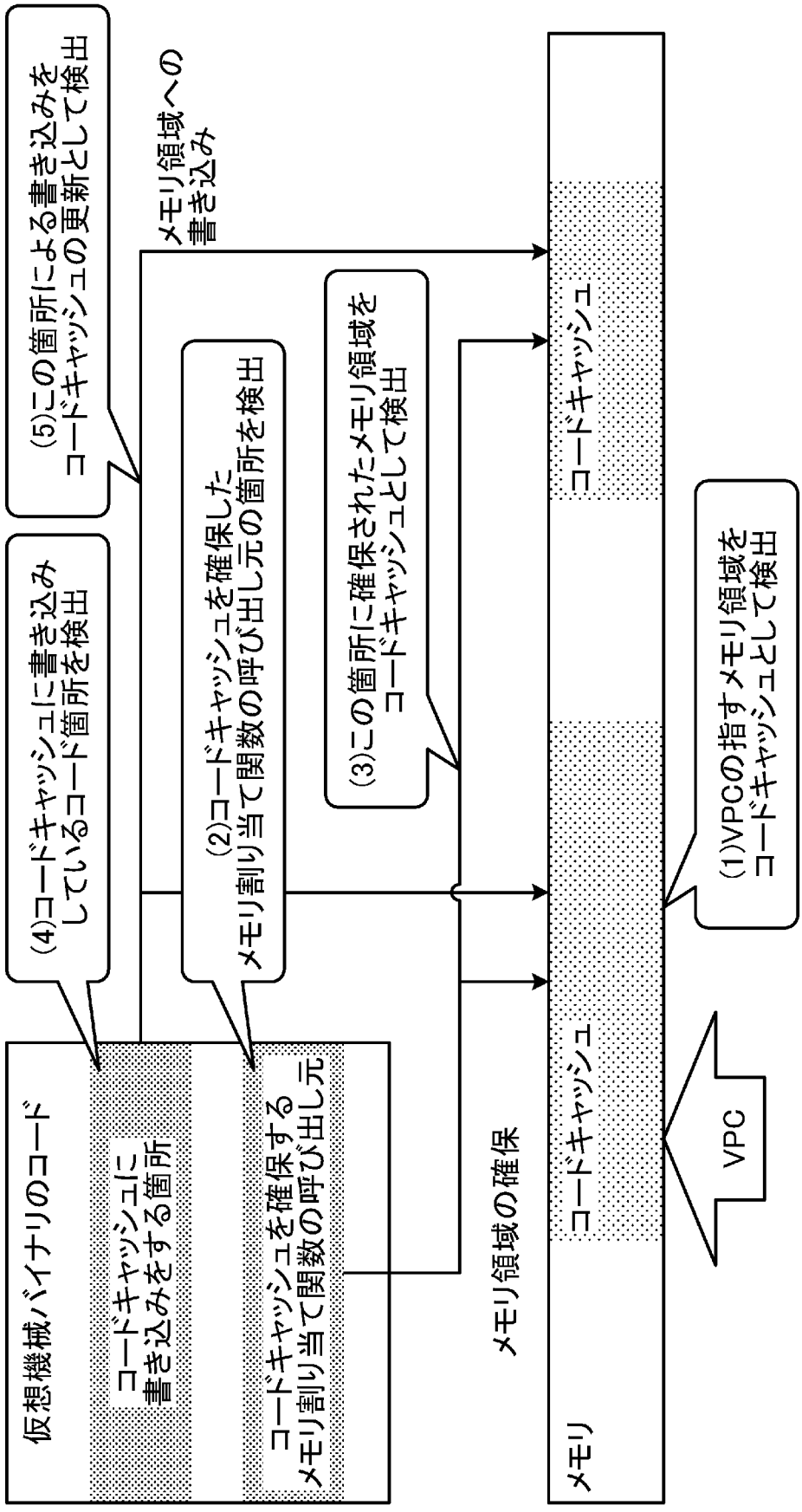
[図16]


(2)VPCの指し先とVM命令ハンドラの
アドレスを照合して一致するメモリ領域に
絞り込み
VM命令ハンドラ1
VM命令ハンドラ2
VM命令ハンドラ3
ポインタキャッシュ
VPC
VPC
VPC
(1)1VM命令実行毎にR/Wがあり
単調増加するメモリ領域を抽出

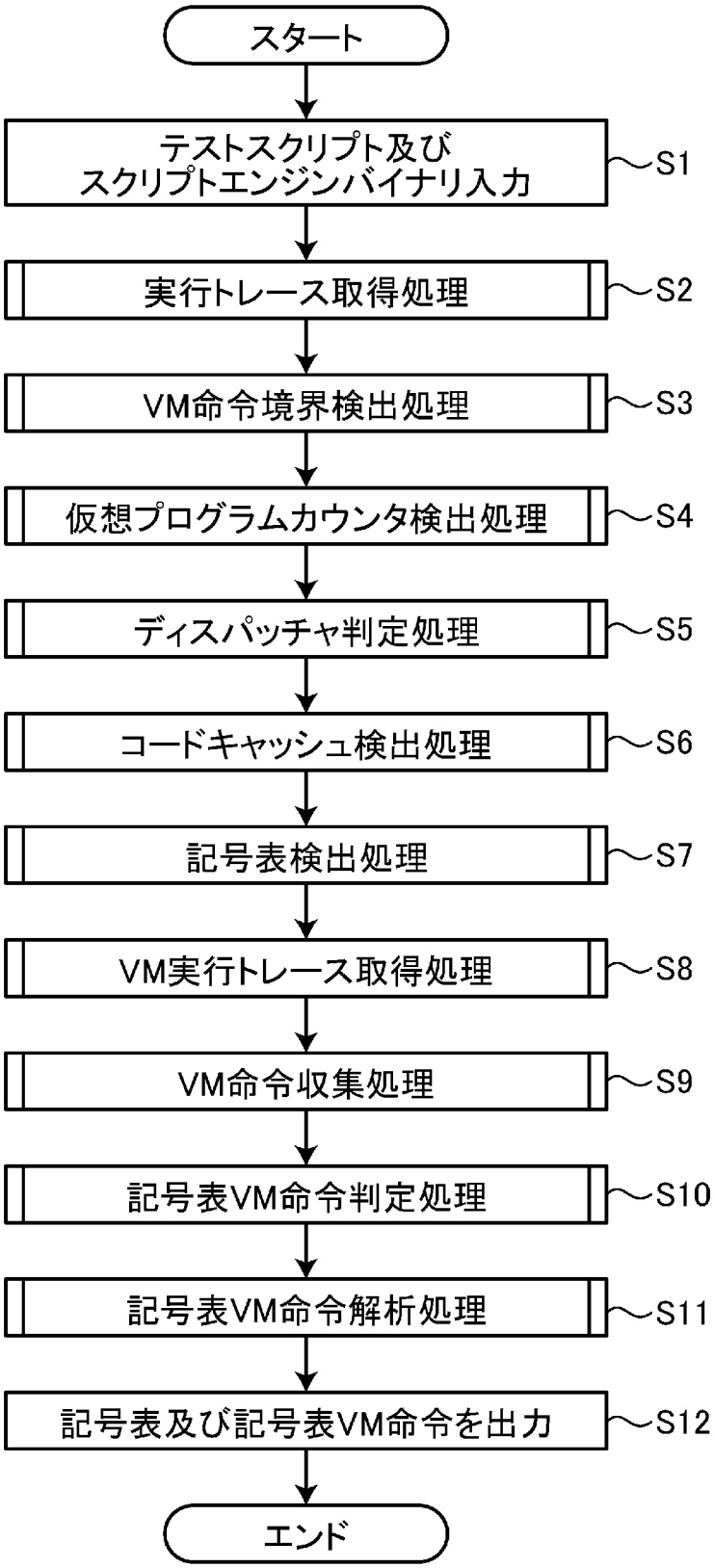
[図17]


(1)仮定: ディスパッチャのコードの類似性は高い
→ VM命令の後半部で共通的に実行されるコードをディスパッチャとして検出
VM命令ハンドラ1
ディスパッチャ
VM命令ハンドラ2
ディスパッチャ
VM命令ハンドラ3
ディスパッチャ

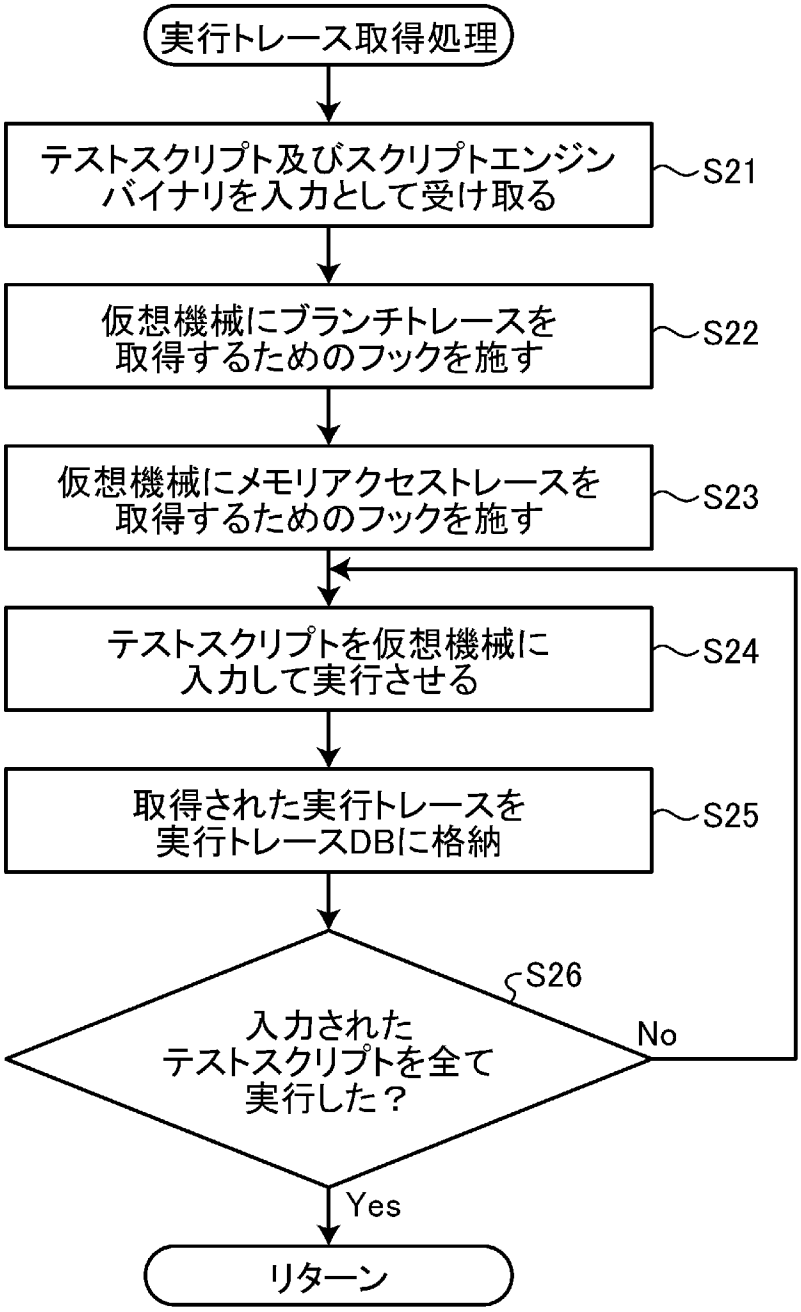
[図18]


仮想機械バイナリのコード
コードキャッシュに
書き込みをする箇所
コードキャッシュを確保する
メモリ割り当て関数の呼び出し元
メモリ領域の確保
メモリ
コードキャッシュ
コードキャッシュ
VPC
(1)VPCの指すメモリ領域を
コードキャッシュとして検出
(2)コードキャッシュを確保した
メモリ割り当て関数の呼び出し元の箇所を検出
(3)この箇所に確保されたメモリ領域を
コードキャッシュとして検出
(4)コードキャッシュに書き込み
しているコード箇所を検出
(5)この箇所による書き込みを
コードキャッシュの更新として検出
メモリ領域への
書き込み

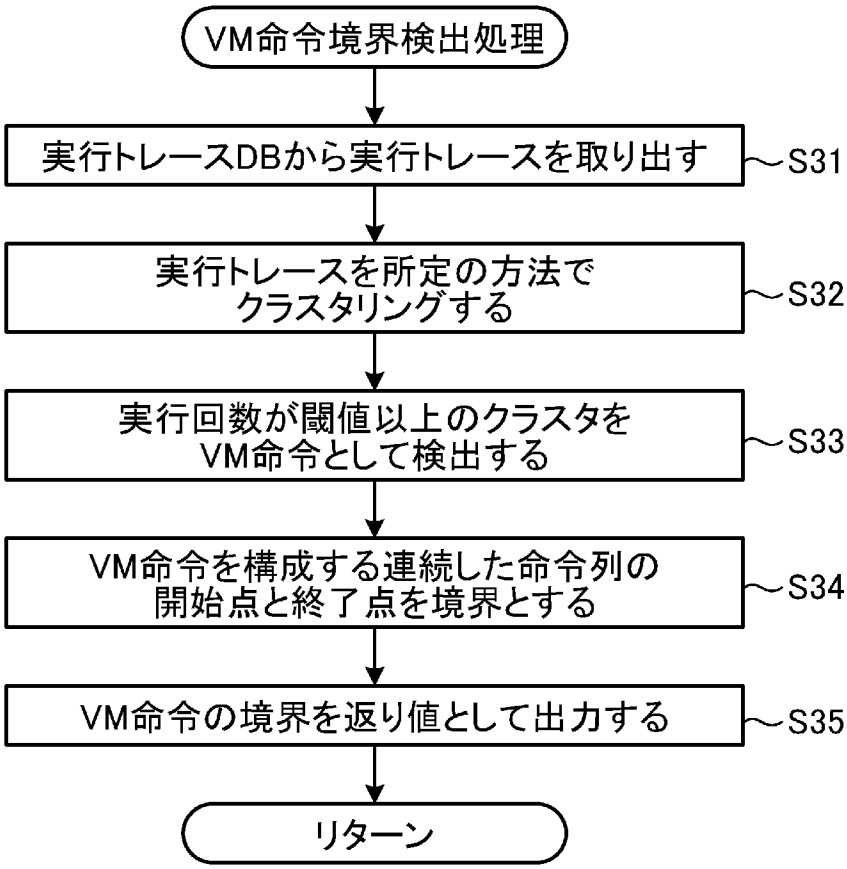
[図19]


スタート
テストスクリプト及び
スクリプトエンジンバイナリ入力
S1
実行トレース取得処理
S2
VM命令境界検出処理
S3
仮想プログラムカウンタ検出処理
S4
ディスパッチャ判定処理
S5
コードキャッシュ検出処理
S6
記号表検出処理
S7
VM実行トレース取得処理
S8
VM命令収集処理
S9
記号表VM命令判定処理
S10
記号表VM命令解析処理
S11
記号表及び記号表VM命令を出力
S12
エンド

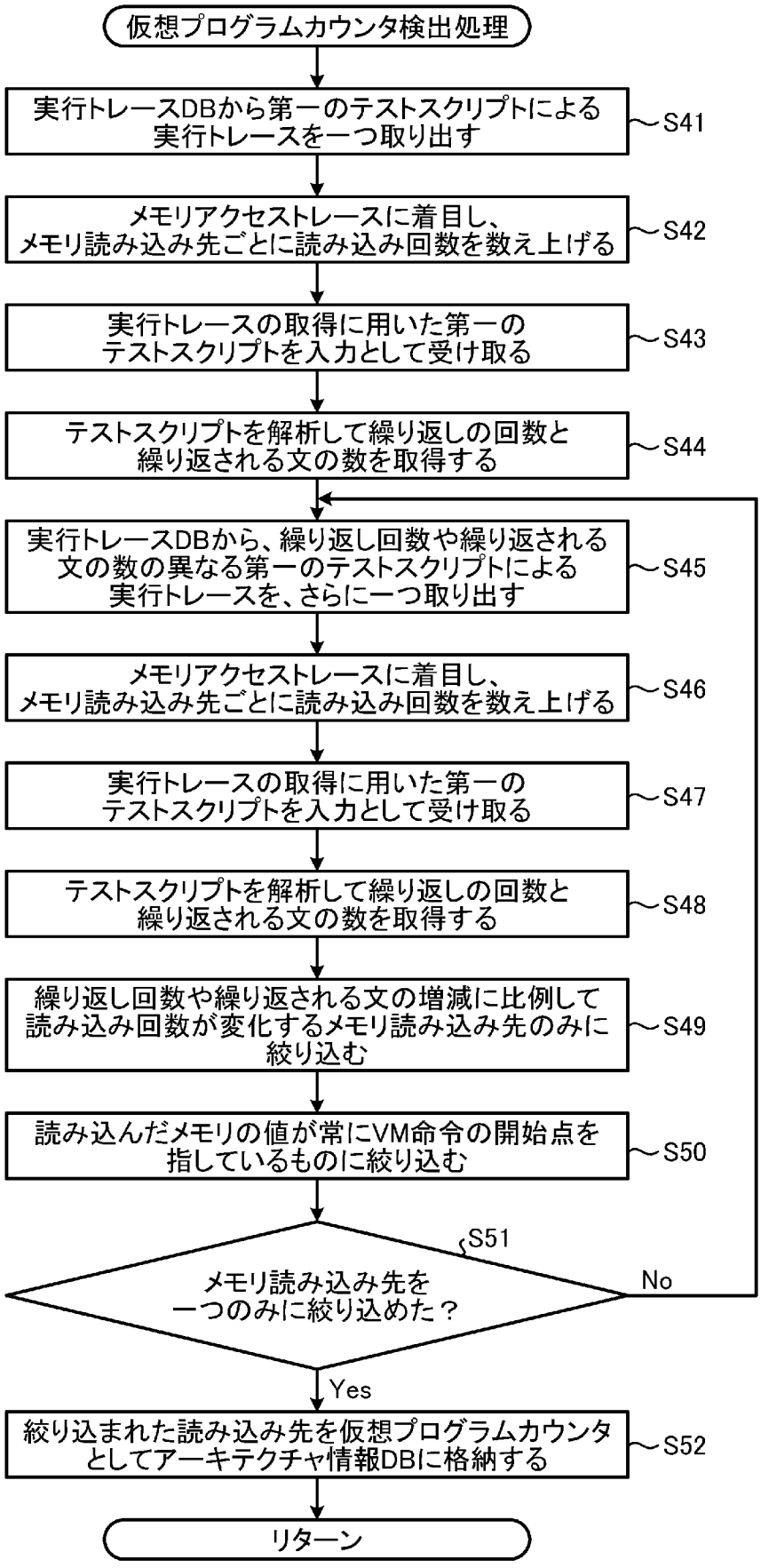
[図20]


実行トレース取得処理
テストスクリプト及びスクリプトエンジンバイナリを入力として受け取る
S21
仮想機械にブランチトレースを取得するためのフックを施す
S22
仮想機械にメモリアクセストレースを取得するためのフックを施す
S23
テストスクリプトを仮想機械に入力して実行させる
S24
取得された実行トレースを実行トレースDBに格納
S25
S26
入力されたテストスクリプトを全て実行した？
No
Yes
リターン

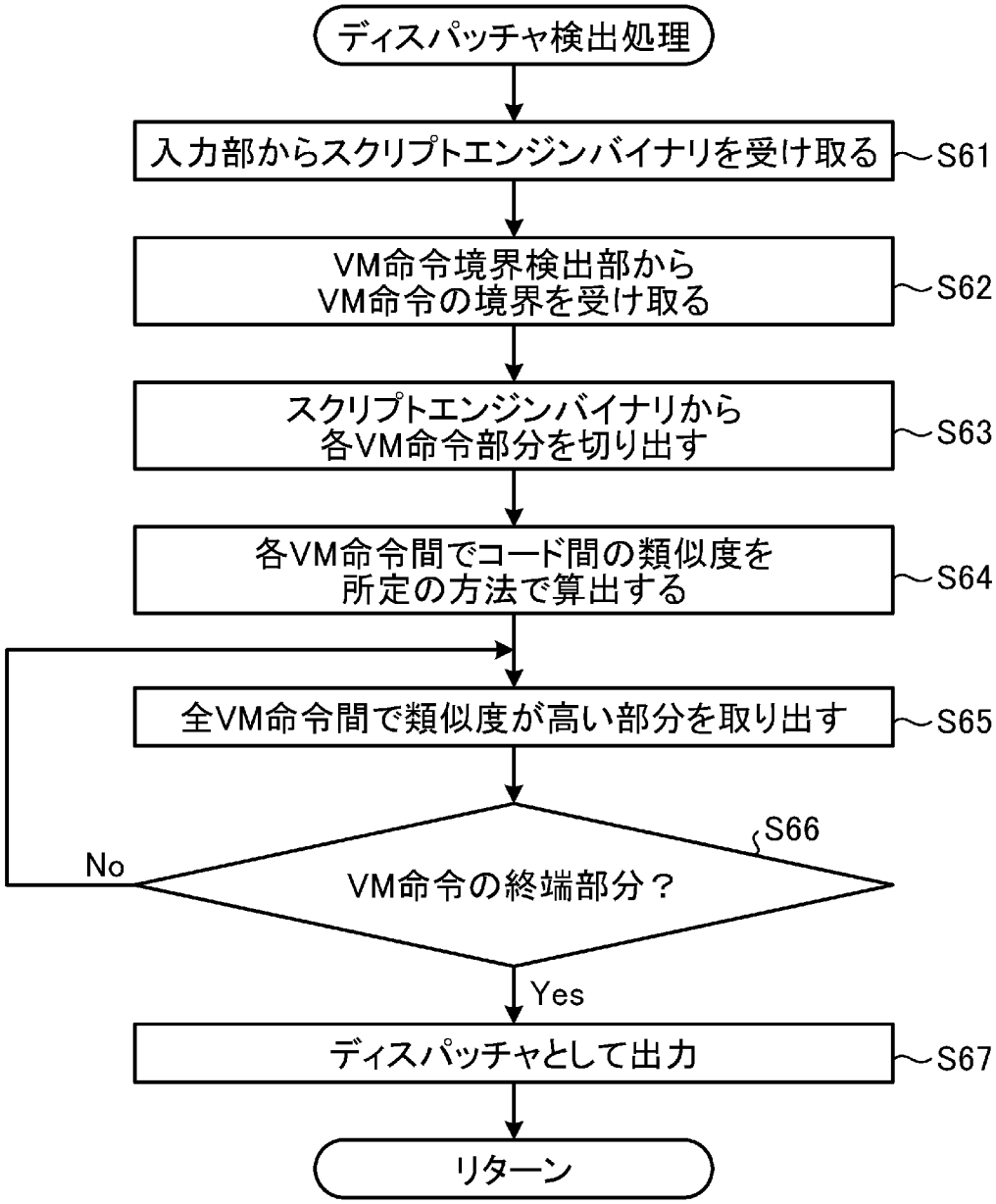
[図21]


VM命令境界検出処理
実行トレースDBから実行トレースを取り出す
S31
実行トレースを所定の方法で
クラスタリングする
S32
実行回数が閾値以上のクラスタを
VM命令として検出する
S33
VM命令を構成する連続した命令列の
開始点と終了点を境界とする
S34
VM命令の境界を返り値として出力する
S35
リターン

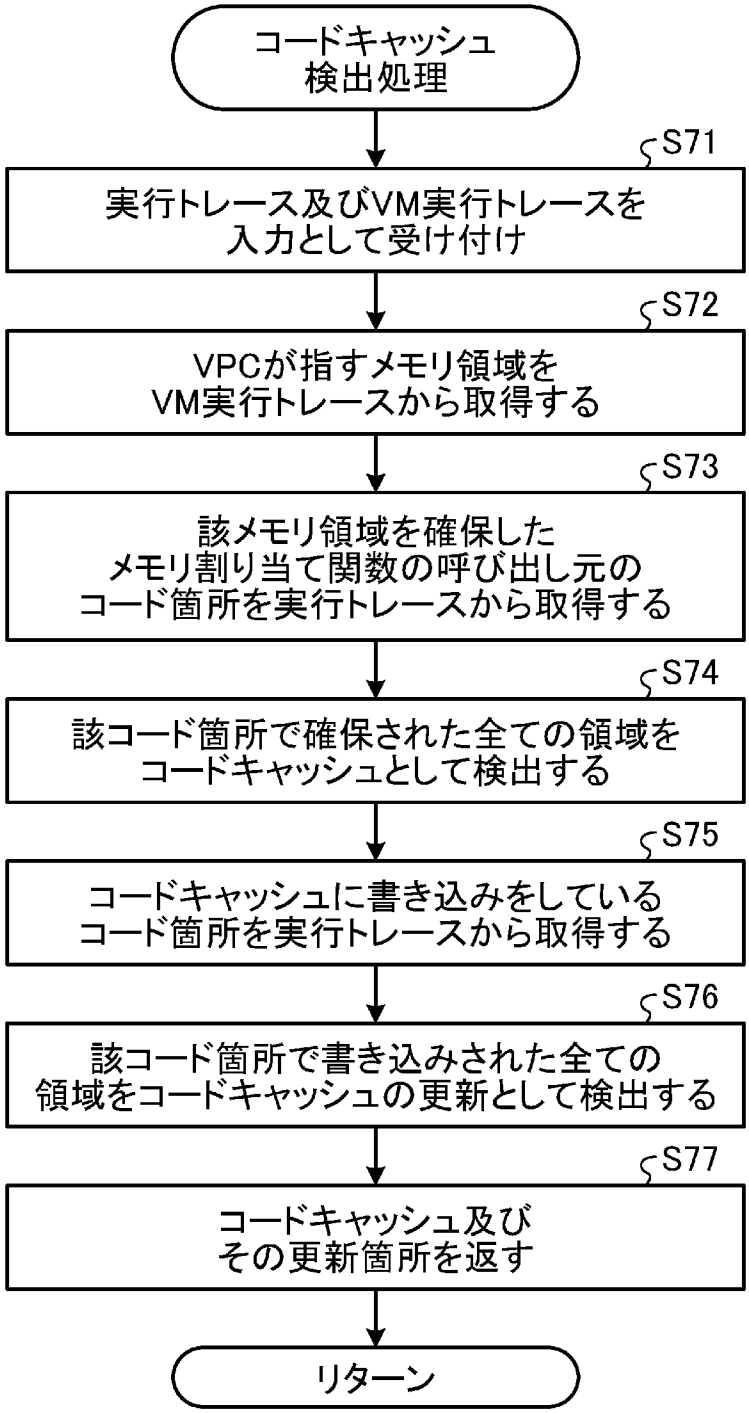
[図22]


仮想プログラムカウンタ検出処理
実行トレースDBから第一のテストスクリプトによる実行トレースを一つ取り出す
S41
メモリアクセストレースに着目し、メモリ読み込み先ごとに読み込み回数を数え上げる
S42
実行トレースの取得に用いた第一のテストスクリプトを入力として受け取る
S43
テストスクリプトを解析して繰り返しの回数と繰り返される文の数を取得する
S44
実行トレースDBから、繰り返し回数や繰り返される文の数の異なる第一のテストスクリプトによる実行トレースを、さらに一つ取り出す
S45
メモリアクセストレースに着目し、メモリ読み込み先ごとに読み込み回数を数え上げる
S46
実行トレースの取得に用いた第一のテストスクリプトを入力として受け取る
S47
テストスクリプトを解析して繰り返しの回数と繰り返される文の数を取得する
S48
繰り返し回数や繰り返される文の増減に比例して読み込み回数が変化するメモリ読み込み先のみに絞り込む
S49
読み込んだメモリの値が常にVM命令の開始点を指しているものに絞り込む
S50
S51
メモリ読み込み先を一つのみに絞り込めた？
No
Yes
絞り込まれた読み込み先を仮想プログラムカウンタとしてアーキテクチャ情報DBに格納する
S52
リターン

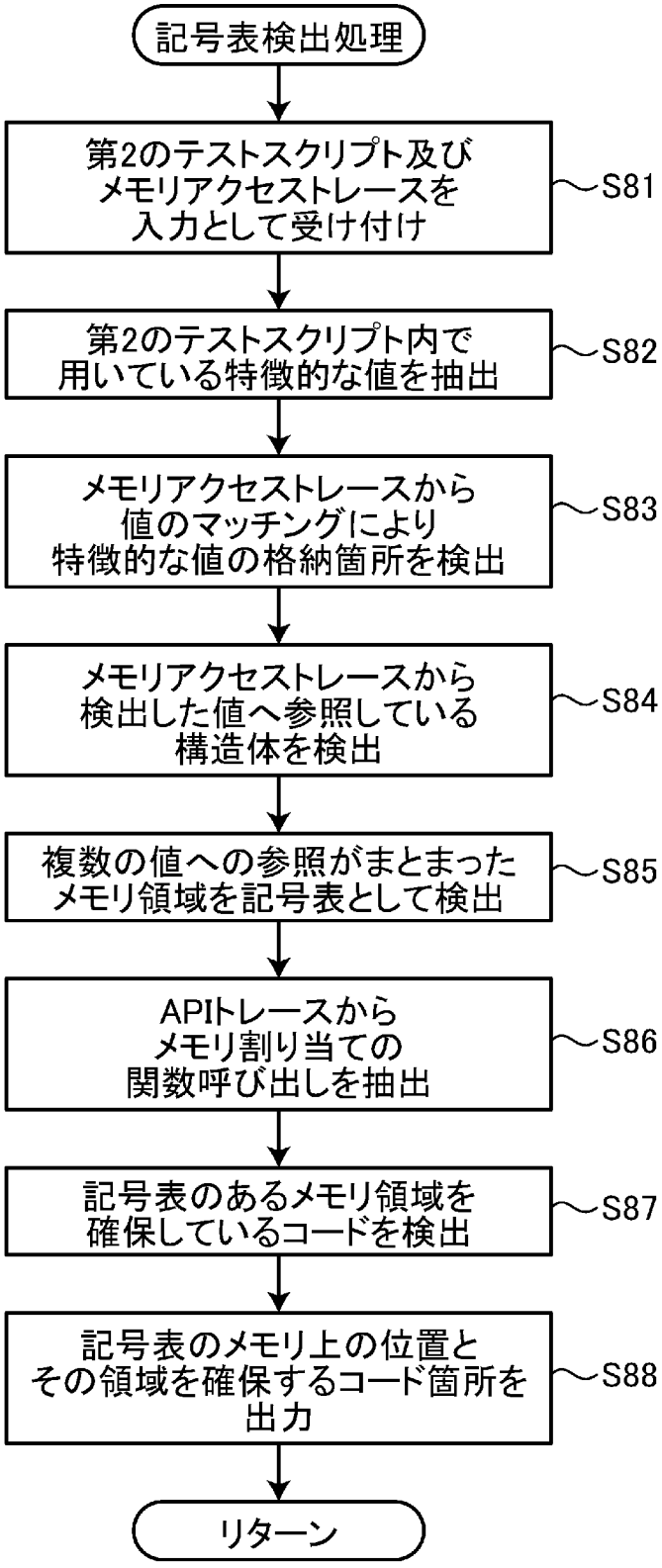
[図23]


ディスパッチャ検出処理
入力部からスクリプトエンジンバイナリを受け取る
S61
VM命令境界検出部から
VM命令の境界を受け取る
S62
スクリプトエンジンバイナリから
各VM命令部分を切り出す
S63
各VM命令間でコード間の類似度を
所定の方法で算出する
S64
全VM命令間で類似度が高い部分を取り出す
S65
S66
No
VM命令の終端部分？
Yes
ディスパッチャとして出力
S67
リターン

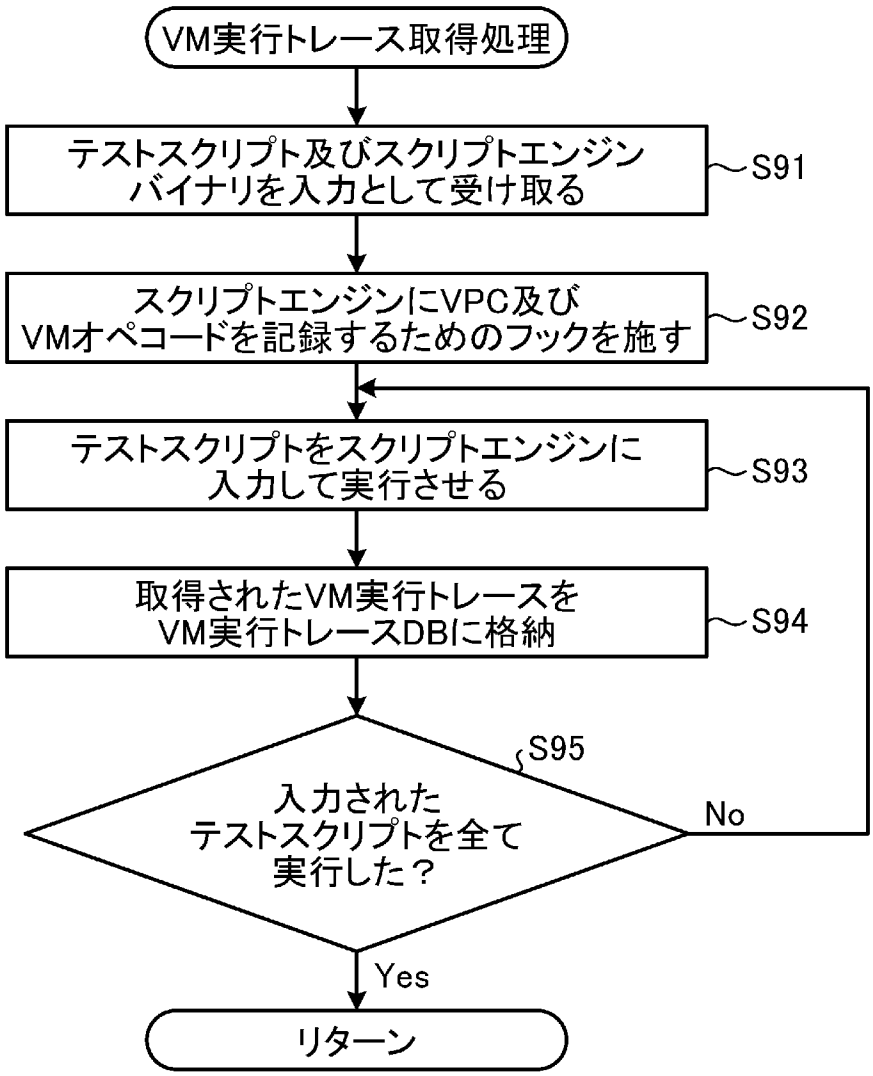
[図24]


コードキャッシュ
検出処理
S71
実行トレース及びVM実行トレースを
入力として受け付け
S72
VPCが指すメモリ領域を
VM実行トレースから取得する
S73
該メモリ領域を確保した
メモリ割り当て関数の呼び出し元の
コード箇所を実行トレースから取得する
S74
該コード箇所で確保された全ての領域を
コードキャッシュとして検出する
S75
コードキャッシュに書き込みをしている
コード箇所を実行トレースから取得する
S76
該コード箇所で書き込みされた全ての
領域をコードキャッシュの更新として検出する
S77
コードキャッシュ及び
その更新箇所を返す
リターン

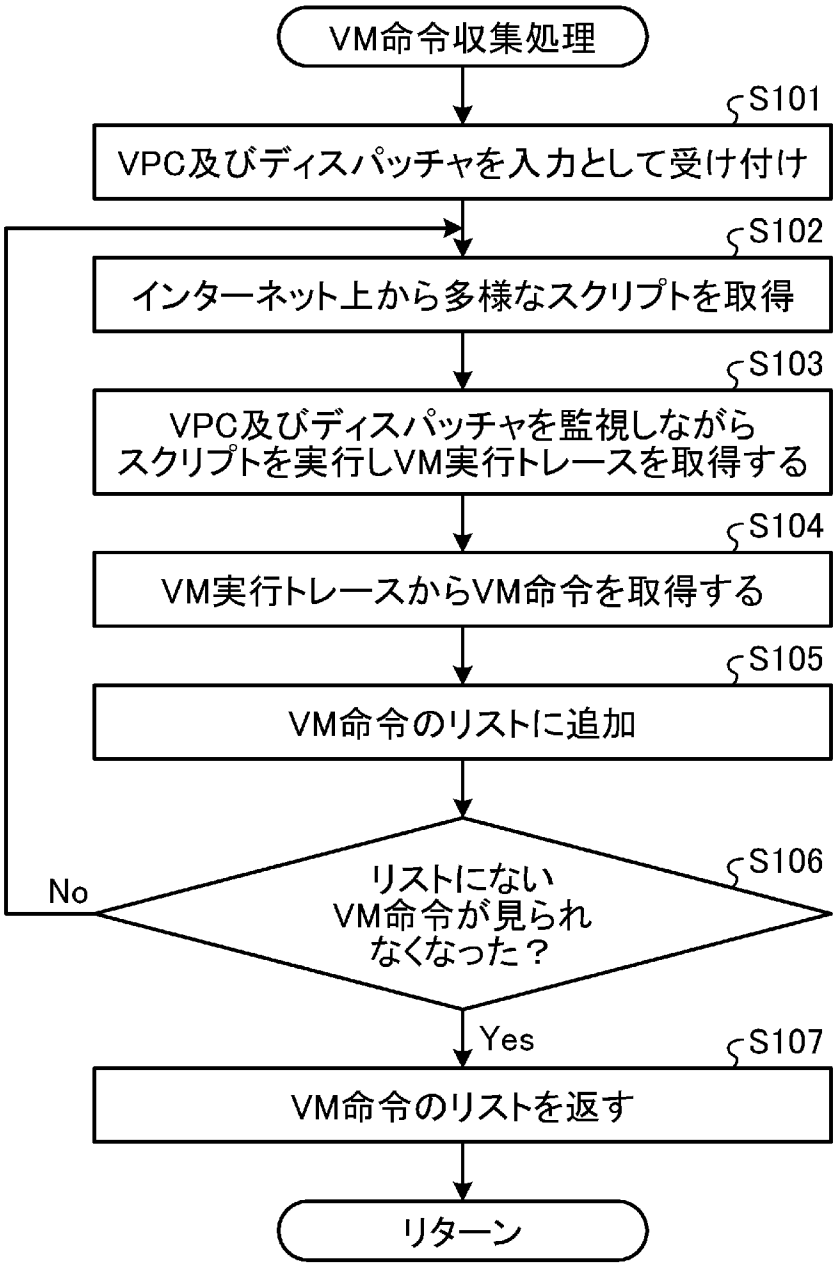
[図25]


記号表検出処理
第2のテストスクリプト及び
メモリアクセストレースを
入力として受け付け
S81
第2のテストスクリプト内で
用いている特徴的な値を抽出
S82
メモリアクセストレースから
値のマッチングにより
特徴的な値の格納箇所を検出
S83
メモリアクセストレースから
検出した値へ参照している
構造体を検出
S84
複数の値への参照がまとまった
メモリ領域を記号表として検出
S85
APIトレースから
メモリ割り当ての
関数呼び出しを抽出
S86
記号表のあるメモリ領域を
確保しているコードを検出
S87
記号表のメモリ上の位置と
その領域を確保するコード箇所を
出力
S88
リターン

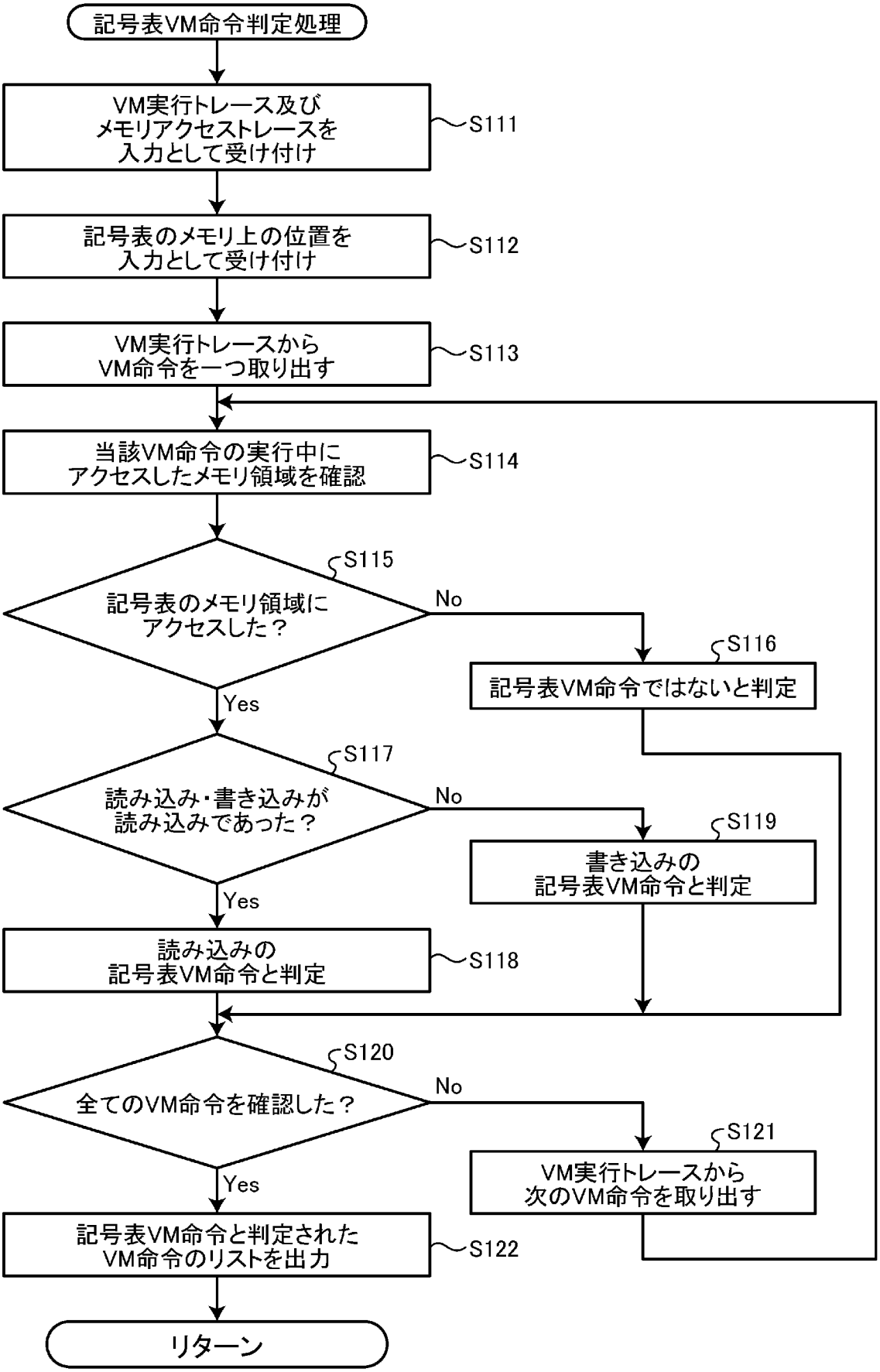
[図26]


VM実行トレース取得処理
テストスクリプト及びスクリプトエンジン
バイナリを入力として受け取る
S91
スクリプトエンジンにVPC及び
VMオペコードを記録するためのフックを施す
S92
テストスクリプトをスクリプトエンジンに
入力して実行させる
S93
取得されたVM実行トレースを
VM実行トレースDBに格納
S94
S95
入力された
テストスクリプトを全て
実行した？
No
Yes
リターン

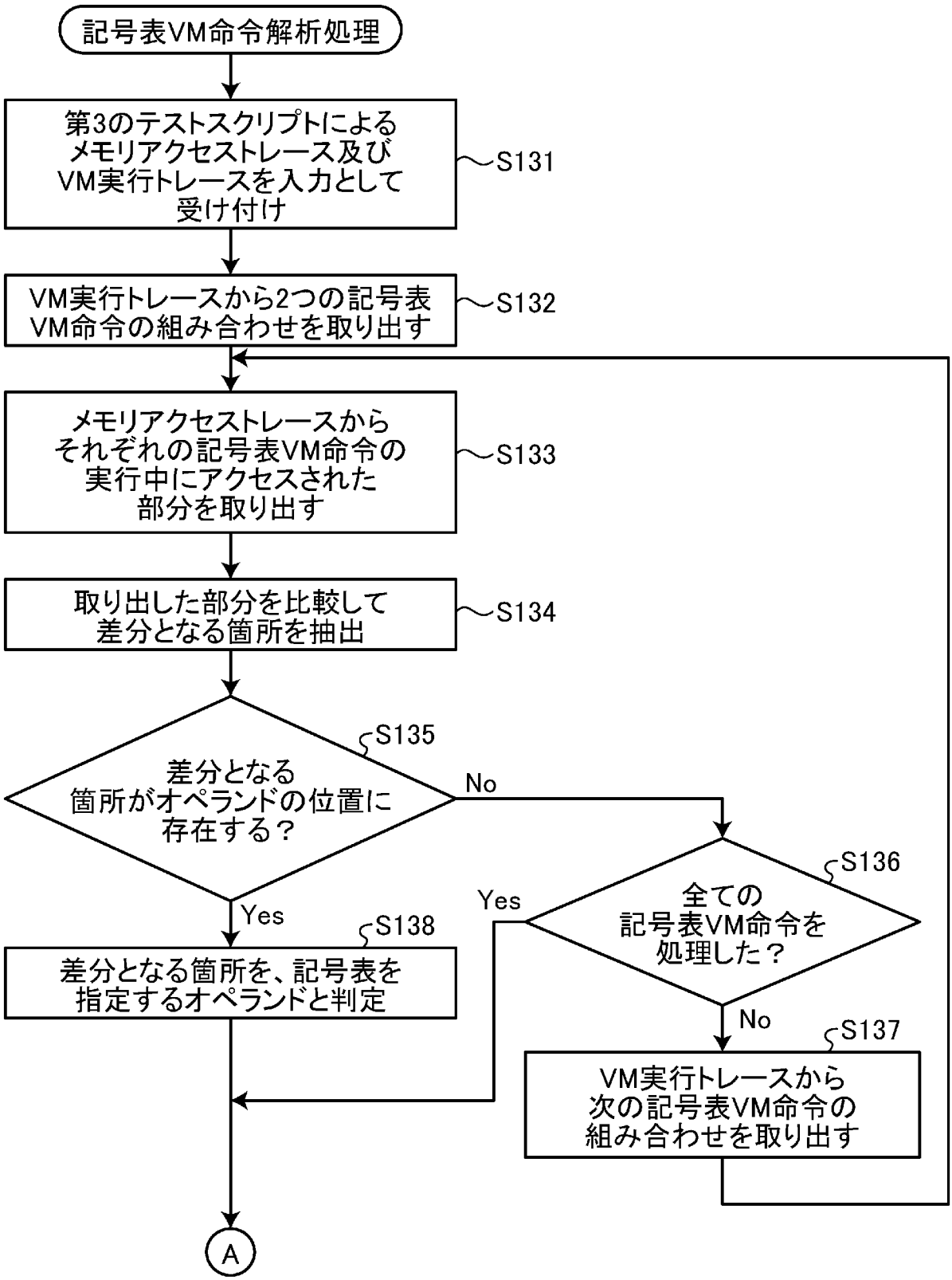
[図27]


VM命令収集処理
S101
VPC及びディスパッチャを入力として受け付け
S102
インターネット上から多様なスクリプトを取得
S103
VPC及びディスパッチャを監視しながら
スクリプトを実行しVM実行トレースを取得する
S104
VM実行トレースからVM命令を取得する
S105
VM命令のリストに追加
S106
リストにない
VM命令が見られ
なくなった？
No
Yes
S107
VM命令のリストを返す
リターン

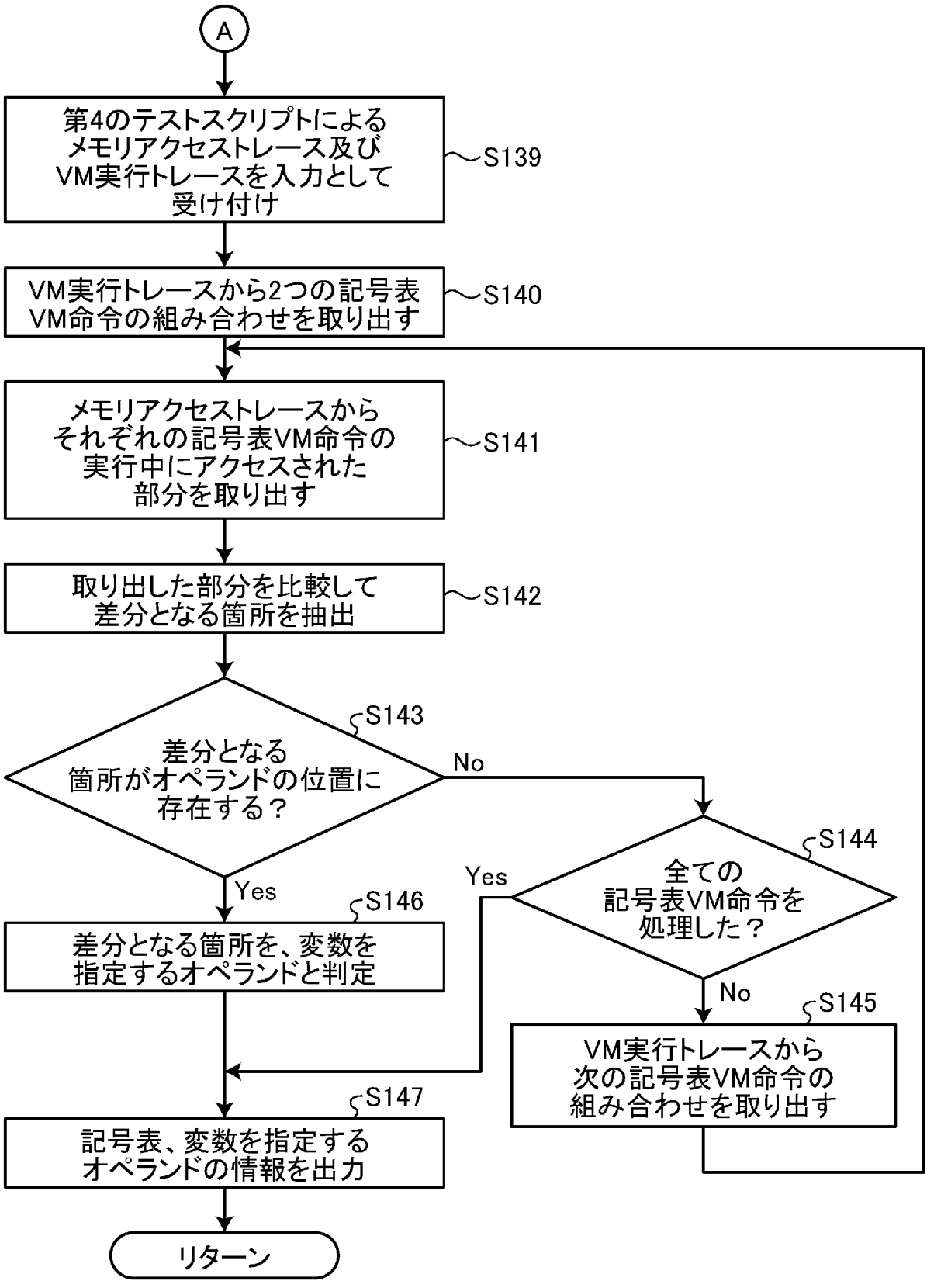
[図28]


記号表VM命令判定処理
VM実行トレース及び
メモリアクセストレースを
入力として受け付け
S111
記号表のメモリ上の位置を
入力として受け付け
S112
VM実行トレースから
VM命令を一つ取り出す
S113
当該VM命令の実行中に
アクセスしたメモリ領域を確認
S114
S115
記号表のメモリ領域に
アクセスした？
No
S116
記号表VM命令ではないと判定
Yes
S117
読み込み・書き込みが
読み込みであった？
No
S119
書き込みの
記号表VM命令と判定
Yes
読み込みの
記号表VM命令と判定
S118
S120
全てのVM命令を確認した？
No
S121
VM実行トレースから
次のVM命令を取り出す
Yes
記号表VM命令と判定された
VM命令のリストを出力
S122
リターン

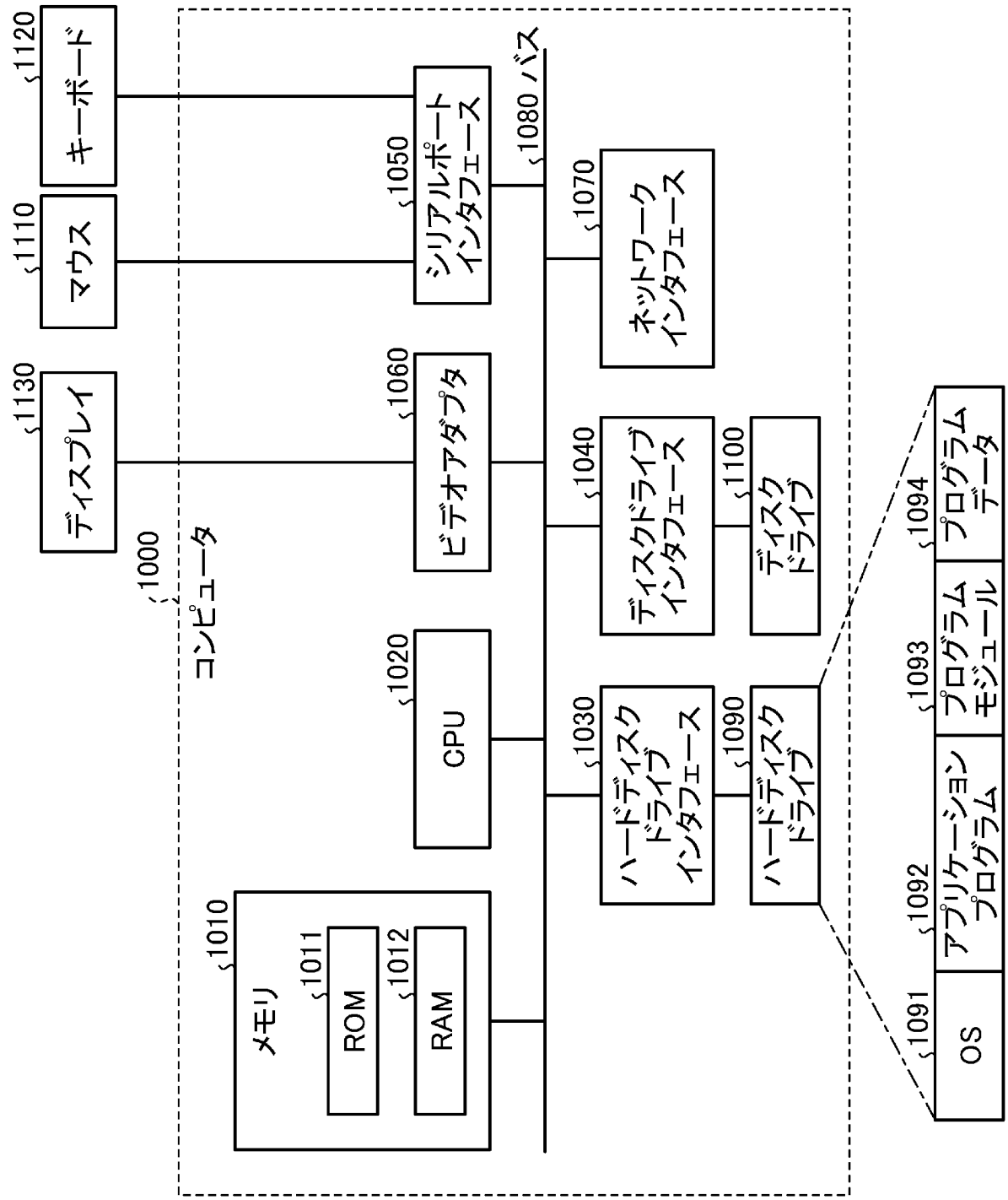
[図29]


記号表VM命令解析処理
第3のテストスクリプトによる
メモリアクセストレース及び
VM実行トレースを入力として
受け付け
S131
VM実行トレースから2つの記号表
VM命令の組み合わせを取り出す
S132
メモリアクセストレースから
それぞれの記号表VM命令の
実行中にアクセスされた
部分を取り出す
S133
取り出した部分を比較して
差分となる箇所を抽出
S134
S135
差分となる
箇所がオペランドの位置に
存在する？
No
Yes
S136
全ての
記号表VM命令を
処理した？
Yes
No
S138
差分となる箇所を、記号表を
指定するオペランドと判定
S137
VM実行トレースから
次の記号表VM命令の
組み合わせを取り出す
A

[図30]


A
第4のテストスクリプトによる
メモリアクセストレース及び
VM実行トレースを入力として
受け付け
S139
VM実行トレースから2つの記号表
VM命令の組み合わせを取り出す
S140
メモリアクセストレースから
それぞれの記号表VM命令の
実行中にアクセスされた
部分を取り出す
S141
取り出した部分を比較して
差分となる箇所を抽出
S142
S143
差分となる
箇所がオペランドの位置に
存在する？
No
Yes
S144
全ての
記号表VM命令を
処理した？
Yes
No
S145
VM実行トレースから
次の記号表VM命令の
組み合わせを取り出す
S146
差分となる箇所を、変数を
指定するオペランドと判定
S147
記号表、変数を指定する
オペランドの情報を出力
リターン

[図31]


1000
コンピュータ
1120
キーボード
1110
マウス
1130
ディスプレイ
1050
シリアルポート
インタフェース
1080 バス
1070
ネットワーク
インタフェース
1060
ビデオアダプタ
1040
ディスクドライブ
インタフェース
1100
ディスク
ドライブ
1020
CPU
1030
ハードディスク
ドライブ
インタフェース
1090
ハードディスク
ドライブ
1010
メモリ
1011
ROM
1012
RAM
1094
プログラム
データ
1093
プログラム
モジュール
1092
アプリケーション
プログラム
1091
OS

INTERNATIONAL SEARCH REPORT

International application No.
PCT/JP2023/015088

A. CLASSIFICATION OF SUBJECT MATTER

G06F 11/34(2006.01)i; ***G06F 21/56***(2013.01)i
FI: G06F11/34 166; G06F21/56

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)

G06F11/34; G06F21/56

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

Published examined utility model applications of Japan 1922-1996
Published unexamined utility model applications of Japan 1971-2023
Registered utility model specifications of Japan 1996-2023
Published registered utility model applications of Japan 1994-2023

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.
X	WO 2022/079840 A1 (NIPPON TELEGRAPH AND TELEPHONE CORPORATION) 21 April 2022 (2022-04-21) paragraphs [0025]-[0130], fig. 1-20	1-2, 7-8
A		3-6
A	WO 2022/180702 A1 (NIPPON TELEGRAPH AND TELEPHONE CORPORATION) 01 September 2022 (2022-09-01) entire text, all drawings	1-8
A	JP 2013-254320 A (NEC CORPORATION) 19 December 2013 (2013-12-19) entire text, all drawings	1-8

☐ 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
12 June 2023	**20 June 2023**
Name and mailing address of the ISA/JP **Japan Patent Office (ISA/JP)** **3-4-3 Kasumigaseki, Chiyoda-ku, Tokyo 100-8915** **Japan**	Authorized officer Telephone No.

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/JP2023/015088

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
WO 2022/079840 A1	21 April 2022	(Family: none)	
WO 2022/180702 A1	01 September 2022	(Family: none)	
JP 2013-254320 A	19 December 2013	(Family: none)	

国際調査報告

国際出願番号
PCT/JP2023/015088

A. 発明の属する分野の分類（国際特許分類（ＩＰＣ））

G06F 11/34(2006.01)i; G06F 21/56(2013.01)i
FI: G06F11/34 166; G06F21/56

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））

G06F11/34; G06F21/56

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	１９７１－２０２３年
日本国実用新案登録公報	１９９６－２０２３年
日本国登録実用新案公報	１９９４－２０２３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
X	WO 2022/079840 A1 (日本電信電話株式会社) 21.04.2022 (2022-04-21) 段落［００２５］－［０１３０］、［図１］－［図２０］	1-2, 7-8
A		3-6
A	WO 2022/180702 A1 (日本電信電話株式会社) 01.09.2022 (2022-09-01) 全文、全図	1-8
A	JP 2013-254320 A (日本電気株式会社) 19.12.2013 (2013-12-19) 全文、全図	1-8

☐ 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抵触するものではなく、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
12.06.2023	20.06.2023

名称及びあて先
日本国特許庁(ISA/JP)
〒100-8915
日本国
東京都千代田区霞が関三丁目４番３号

権限のある職員（特許庁審査官）
松平　英 5B 3146

電話番号 03-3581-1101 内線 3546

国際調査報告
パテントファミリーに関する情報

国際出願番号
PCT/JP2023/015088

引用文献	公表日	パテントファミリー文献	公表日
WO 2022/079840 A1	21.04.2022	（ファミリーなし）	
WO 2022/180702 A1	01.09.2022	（ファミリーなし）	
JP 2013-254320 A	19.12.2013	（ファミリーなし）	