(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6142724号
(P6142724)
(45)発行日　平成29年6月7日(2017.6.7)　(24)登録日　平成29年5月19日(2017.5.19)

(51)Int.Cl.　F I
G06F 11/36 (2006.01)　G06F 11/36 184
G06F 11/36 108

請求項の数 8　(全 29 頁)

(21)出願番号　特願2013-163794(P2013-163794)
(22)出願日　平成25年8月7日(2013.8.7)
(65)公開番号　特開2015-32282(P2015-32282A)
(43)公開日　平成27年2月16日(2015.2.16)
審査請求日　平成28年5月10日(2016.5.10)

(73)特許権者　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号
(74)代理人　100103528
弁理士　原田　一男
(72)発明者　徳本　晋
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
(72)発明者　上原　忠弘
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
(72)発明者　宗像　一樹
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
最終頁に続く

(54)【発明の名称】テストデータ生成プログラム、方法及び装置

(57)【特許請求の範囲】
【請求項１】
プログラムにおいて実行される複数の関数の各々についてシンボリック実行を行うと共に、シンボル変数に関わる分岐を辿った履歴を表す条件と、当該関数の引数及び返却値についての条件との少なくともいずれかを含む実行条件を前記複数の関数の各々について生成し、
生成された複数の実行条件を論理積によって統合し、
統合後の条件を満たすシンボル変数の値を算出し、算出された当該シンボル変数の値を含むテストデータを生成する
処理をコンピュータに実行させるためのテストデータ生成プログラム。
【請求項２】
前記複数の関数の各々について、シンボル変数を設定するためのプログラムと当該関数において呼び出される関数のスタブとを生成する
処理をさらに実行させるための請求項１記載のテストデータ生成プログラム。
【請求項３】
前記実行条件を生成する処理において、
前記関数の引数に関数ポインタが含まれる場合に、当該関数ポインタと当該関数ポインタが指す関数との関係を表す条件を、前記実行条件に追加する
ことを特徴とする請求項１又は２記載のテストデータ生成プログラム。
【請求項４】

前記実行条件を生成する処理において、
前記プログラムにグローバル変数が含まれる場合、当該グローバル変数の読み込みについての条件及び書き込みについての条件を、前記実行条件に追加する
ことを特徴とする請求項１乃至３のいずれか１つ記載のテストデータ生成プログラム。
【請求項５】
前記実行条件を生成する処理において、
前記関数において同一の関数を複数回呼び出す場合、呼び出される当該関数の引数についての条件を前記複数回の呼び出し毎に生成し、前記実行条件に追加する
ことを特徴とする請求項１乃至４のいずれか１つ記載のテストデータ生成プログラム。
【請求項６】
前記実行条件を生成する処理において、
関数間の関係を表すコールツリーを生成し、
前記コールツリーの端に追加された関数が、再帰呼び出しされる関数である場合、前記コールツリーの端に追加された関数について生成された条件を削除する
ことを特徴とする請求項１記載のテストデータ生成プログラム。
【請求項７】
プログラムにおいて実行される複数の関数の各々についてシンボリック実行を行うと共に、シンボル変数に関わる分岐を辿った履歴を表す条件と、当該関数の引数及び返却値についての条件との少なくともいずれかを含む実行条件を前記複数の関数の各々について生成し、
生成された複数の実行条件を論理積によって統合し、
統合後の条件を満たすシンボル変数の値を算出し、算出された当該シンボル変数の値を含むテストデータを生成する
処理をコンピュータが実行するテストデータ生成方法。
【請求項８】
プログラムにおいて実行される複数の関数の各々についてシンボリック実行を行うと共に、シンボル変数に関わる分岐を辿った履歴を表す条件と、当該関数の引数及び返却値についての条件との少なくともいずれかを含む実行条件を前記複数の関数の各々について生成する第１処理部と、
生成された複数の実行条件を論理積によって統合する第２処理部と、
統合後の条件を満たすシンボル変数の値を算出し、算出された当該シンボル変数の値を含むテストデータを生成する第３処理部と、
を有するテストデータ生成装置。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、プログラムのテストに利用するテストデータを生成する技術に関する。
【背景技術】
【０００２】
シンボリック実行とは、プログラム内の変数を具体化せず、変数をシンボル化して（すなわち、記号のまま）プログラムを実行する技術である。シンボル化される変数は、シンボル変数と呼ばれる。シンボリック実行中においては、具体値ではなくシンボル変数が満たすべき条件（以下、パス条件と呼ぶ）がシンボル変数の値としてメモリ等に保持される。そして、シンボリック実行が完了すると、プログラムにおける各パスについてパス条件が得られるため、パス条件を満たすシンボル変数の具体値を求めることで、プログラムをテストするためのテストデータを得ることができる。
【０００３】
例えば、図１に示した、絶対値取得関数として問題があるプログラムについてシンボリック実行を行うことを考える。図１には、シンボリック実行の対象になるプログラムのコードと、そのプログラムの制御構造とが示されている。このプログラムには変数Ｘが含ま

れている。このプログラムを対象としてシンボリック実行を行うと、Ｘ＝－２というテストデータ（パス条件はＸ＜０）と、Ｘ＝１２３４というテストデータ（パス条件はＸ≧０且つＸ＝１２３４）と、Ｘ＝３というテストデータ（パス条件はＸ≧０且つＸ≠１２３４）とを得ることができる。

【０００４】

上で述べたように、通常のシンボリック実行は、プログラム内の全パスを対象範囲とする。例えば図１に示したような単純なプログラムであれば、たとえ全パスについてシンボリック実行を行ったとしても、メモリ容量等の計算資源が足りなくなることは無い。しかしながら、複数のプログラムが連携して実行される場合等についてシンボリック実行を行うと、パスの数が非常に多いことに起因して、計算資源が足りなくなることがある。

【先行技術文献】

【特許文献】

【０００５】

【特許文献１】特開２０１２－０６８８６９号公報

【特許文献２】特開２００９－８７３５５号公報

【発明の概要】

【発明が解決しようとする課題】

【０００６】

従って、本発明の目的は、１つの側面では、シンボリック実行の際に使用する計算資源の量を減らすための技術を提供することである。

【課題を解決するための手段】

【０００７】

本発明に係るテストデータ生成方法は、プログラムにおいて実行される複数の関数の各々についてシンボリック実行を行うと共に、シンボル変数に関わる分岐を辿った履歴を表す条件と当該関数の引数及び返却値についての条件とを含む実行条件を複数の関数の各々について生成し、生成された複数の実行条件を論理積によって統合し、統合後の条件を満たすシンボル変数の値を算出し、算出された当該シンボル変数の値を含むテストデータを生成する処理を含む。

【発明の効果】

【０００８】

１側面では、シンボリック実行の際に使用する計算資源の量を減らせるようになる。

【図面の簡単な説明】

【０００９】

【図１】図１は、シンボリック実行について説明するための図である。

【図２】図２は、本実施の形態における情報処理装置の機能ブロック図である。

【図３】図３は、プログラム格納部に格納されるプログラムの一例を示す図である。

【図４】図４は、コールツリーの一例を示す図である。

【図５】図５は、コールスタックについて説明するための図である。

【図６】図６は、プログラムの実行木を示す図である。

【図７】図７は、条件表の一例を示す図である。

【図８】図８は、条件表の一例を示す図である。

【図９】図９は、条件表の一例を示す図である。

【図１０】図１０は、統合後の条件及び判定結果を示す図である。

【図１１】図１１は、統合後の条件及び判定結果を示す図である。

【図１２】図１２は、統合後の条件及び判定結果を示す図である。

【図１３】図１３は、統合後の条件及び判定結果を示す図である。

【図１４】図１４は、テストデータに格納されるデータの一例を示す図である。

【図１５】図１５は、メインの処理フローを示す図である。

【図１６】図１６は、前処理の処理フローを示す図である。

【図１７】図１７は、スタブの一例を示す図である。

【図１８】図１８は、ドライバの一例を示す図である。
【図１９】図１９は、シンボリック実行の処理フローを示す図である。
【図２０】図２０は、プログラムの一例を示す図である。
【図２１】図２１は、コールツリーの一例を示す図である。
【図２２】図２２は、条件表の一例を示す図である。
【図２３】図２３は、条件表の一例を示す図である。
【図２４】図２４は、条件表の一例を示す図である。
【図２５】図２５は、条件表の一例を示す図である。
【図２６】図２６は、プログラムの一例を示す図である。
【図２７】図２７は、コールツリーの一例を示す図である。
【図２８】図２８は、条件表の一例を示す図である。
【図２９】図２９は、条件表の一例を示す図である。
【図３０】図３０は、統合後の条件及び判定結果を示す図である。
【図３１】図３１は、統合後の条件及び判定結果を示す図である。
【図３２】図３２は、テストデータ格納部に格納されるデータの一例を示す図である。
【図３３】図３３は、追加処理の処理フローを示す図である。
【図３４】図３４は、プログラムの一例を示す図である。
【図３５】図３５は、コールツリーに対する処理について説明するための図である。
【図３６】図３６は、条件表における条件の削除について説明するための図である。
【図３７】図３７は、プログラムの一例を示す図である。
【図３８】図３８は、コールツリーの一例を示す図である。
【図３９】図３９は、条件表の一例を示す図である。
【図４０】図４０は、条件表の一例を示す図である。
【図４１】図４１は、条件表の一例を示す図である。
【図４２】図４２は、条件表の一例を示す図である。
【図４３】図４３は、統合後の条件及び判定結果を示す図である。
【図４４】図４４は、シンボリック実行の処理フローを示す図である。
【図４５】図４５は、統合処理の処理フローを示す図である。
【図４６】図４６は、通常のシンボリック実行における状態保存について説明するための図である。
【図４７】図４７は、本実施の形態の方法における状態保存について説明するための図である。
【図４８】図４８は、プログラムの分岐の一例を示す図である。
【図４９】図４９は、プログラムの一例を示す図である。
【図５０】図５０は、変更後のｆｕｎｃＣの一例を示す図である。
【図５１】図５１は、変更後のｆｕｎｃＣについて生成される条件表の一例を示す図である。
【図５２】図５２は、コンピュータの機能ブロック図である。
【発明を実施するための形態】
【００１０】
図２に、本実施の形態における情報処理装置１の機能ブロック図を示す。情報処理装置１は、入力部１０１と、プログラム格納部１０２と、前処理部１０３と、ドライバ格納部１０４１と、スタブ格納部１０４２と、シンボリック実行部１０５と、状態データ格納部１０６と、条件表格納部１０７と、コールツリー格納部１０８と、統合処理部１０９と、テストデータ生成部１１０と、テストデータ格納部１１１とを含む。
【００１１】
入力部１０１は、テストデータが生成されるべきプログラムの入力を受け付け、プログラム格納部１０２に格納する。前処理部１０３は、プログラム格納部１０２に格納されているプログラムに基づき、シンボリック実行を開始するためのドライバを生成し、ドライバ格納部１０４１に格納する。また、前処理部１０３は、プログラム格納部１０２に格納

されているプログラムに基づきスタブを生成し、スタブ格納部１０４２に格納する。
【００１２】
シンボリック実行部１０５は、プログラム格納部１０２に格納されているプログラム、ドライバ格納部１０４１に格納されているドライバ、及びスタブ格納部１０４２に格納されているスタブを用いてシンボリック実行を行い、シンボリック実行中における実行状態のデータを状態データ格納部１０６に格納する。また、シンボリック実行部１０５は、シンボリック実行時に得られた条件及び返却値（返り値とも呼ばれる）等を含む条件表を条件表格納部１０７に格納する。また、シンボリック実行部１０５は、関数の呼び出し関係を表すコールツリーを生成し、コールツリー格納部１０８に格納する。
【００１３】
統合処理部１０９は、条件表格納部１０７に格納されている条件表及びコールツリー格納部１０８に格納されているコールツリーを用いて、条件を統合する処理を実行し、処理結果を条件表格納部１０７に格納する。テストデータ生成部１１０は、条件表格納部１０７に最終的に格納された条件表における条件を満たすテストデータを生成し、テストデータ格納部１１１に格納する。
【００１４】
図３に、プログラム格納部１０２に格納されるプログラムの一例を示す。図３の例では、関数ｆｕｎｃＡについてのプログラムと、関数ｆｕｎｃＢについてのプログラムと、関数ｆｕｎｃＣについてのプログラムとが格納される。ｆｕｎｃＡは４行目においてｆｕｎｃＢを呼び出し、ｆｕｎｃＢは７行目及び９行目においてｆｕｎｃＣを呼び出す。本実施の形態においては、図３に示すように、複数の関数が実行されるプログラムを処理対象とする。
【００１５】
図４に、コールツリーの一例を示す。図４の例は、図３に示したプログラムについて生成されたコールツリーであり、矢印に指されている関数は呼び出される側の関数である。このコールツリーにおいては、最も上位の関数はｆｕｎｃＡであり、最も下位の関数はｆｕｎｃＣである。コールツリー格納部１０８には、このような関数の呼び出し関係を表すコールツリーのデータが格納される。
【００１６】
図５及び図６を用いて、プログラムの実行状態について説明する。まず、図５を用いて、コールスタックについて説明する。図５に示したプログラムは、メインの処理を表すコードにおいて関数ｆｕｎｃＡを呼び出し、関数ｆｕｎｃＡにおいて関数ａｂｓを呼び出すプログラムである。命令１乃至７におけるいずれかの命令を実行中におけるコールスタックはｓｔａｃｋ３のようになっており、命令８乃至１５におけるいずれかの命令を実行中におけるコールスタックはｓｔａｃｋ２のようになっており、命令１６乃至１９におけるいずれかの命令を実行中におけるコールスタックはｓｔａｃｋ１のようになっている。
【００１７】
図６に、図５に示したプログラムの実行木を示す。図６の実行木においては、各実行状態について、プログラムカウンタの値、コールスタックの識別情報（図５における識別情報に対応している）、メモリの内容、及びパス条件（本実施の形態においては、シンボル変数に関わる分岐を辿った履歴を表す条件）が示されている。例えば、プログラムカウンタの値が１２である場合には、実行状態はｓｔａｔｅ１又はｓｔａｔｅ２である。「＊」は、変数がシンボル変数であることを表す。状態データ格納部１０６には、シンボリック実行中におけるプログラムの実行状態を表すデータが、図６における６０のような形式で格納される。
【００１８】
次に、図７乃至図１４を用いて、本実施の形態における処理の概要を説明する。本実施の形態においては、関数毎に生成されたドライバ及びスタブを用いて、関数毎にシンボリック実行を行う。そして、シンボリック実行によって各関数について生成された条件を論理積によって統合し、統合後の条件を満たすシンボル変数の値によってテストデータを生

成する。以下では、ｆｕｎｃＢの返却値をｆｕｎｃＢ＿ｒｅｔと表し、ｆｕｎｃＣの返却値をｆｕｎｃＣ＿ｒｅｔと表し、ｆｕｎｃＢの引数ｘをｆｕｎｃＢ＿ｘと表し、ｆｕｎｃＣの引数ｘをｆｕｎｃＣ＿ｘと表す。

【００１９】

図７に、図３に示したｆｕｎｃＡについてシンボリック実行を行った場合において、各パスについて生成される条件及び返却値を示す。ＩＤは、パスのＩＤである。条件には、パス条件と、呼び出す関数の引数についての条件とが含まれる。パス条件に該当するのは、Ａ－（１）における「ｘ＜＝０」、Ａ－（２）における「ｘ＞０　∧　ｆｕｎｃＢ＿ｒｅｔ＞０」、及びＡ－（３）における「ｘ＞０　∧　ｆｕｎｃＢ＿ｒｅｔ＜＝０」である。呼び出す引数についての条件は、仮引数と実引数との関係を表しており、Ａ－（２）においては「ｆｕｎｃＢ＿ｘ＝＝ｘ」が該当し、Ａ－（３）においては「ｆｕｎｃＢ＿ｘ＝＝ｘ」が該当する。図７における返却値は、ｆｕｎｃＡの返却値である。

【００２０】

図８に、図３に示したｆｕｎｃＢについてシンボリック実行を行った場合において、各パスについて生成される条件及び返却値を示す。図８に示した条件にも、パス条件と、呼び出す関数の引数についての条件とが含まれる。パス条件に該当するのは、Ｂ－（１）における「ｘ＝＝０」、Ｂ－（２）における「ｘ！＝０　∧　ｘ＞１０　∧　ｆｕｎｃＣ＿ｒｅｔ＞＝０」、Ｂ－（３）における「ｘ！＝０　∧　ｘ＞１０　∧　ｆｕｎｃＣ＿ｒｅｔ＞＝０」、Ｂ－（４）における「ｘ！＝０　∧　ｘ＜＝１０　∧　ｆｕｎｃＣ＿ｒｅｔ＜０」、及びＢ－（５）における「ｘ！＝０　∧　ｘ＜＝１０　∧　ｆｕｎｃＣ＿ｒｅｔ＞＝０」である。呼び出す引数についての条件に該当するのは、Ｂ－（２）における「ｆｕｎｃＣ＿ｘ＝＝ｘ＋１」、Ｂ－（３）における「ｆｕｎｃＣ＿ｘ＝＝ｘ＋１」、Ｂ－（４）における「ｆｕｎｃＣ＿ｘ＝＝ｘ－１」、及びＢ－（５）における「ｆｕｎｃＣ＿ｘ＝＝ｘ－１」である。図８における返却値は、ｆｕｎｃＢの返却値である。

【００２１】

図９に、図３に示したｆｕｎｃＣについてシンボリック実行を行った場合において、各パスについて生成される条件及び返却値を示す。図９に示した条件には、パス条件が含まれる。具体的には、Ｃ－（１）にお「ｘ＝＝２０」と、Ｃ－（２）における「ｘ！＝２０」とがパス条件である。図９における返却値は、ｆｕｎｃＣの返却値である。なお、図９に示したような条件表は、条件表格納部１０７に格納される。

【００２２】

なお、図７乃至図９に示した返却値は、統合処理部１０９によって条件に変換され、図７乃至図９に示した条件に追加される。例えば、Ｂ－（２）における返却値は、「ｆｕｎｃＢ＿ｒｅｔ＝＝－１」という条件に変換され、Ｂ－（２）における条件「ｘ！＝０　∧　ｘ＞１０　∧　ｆｕｎｃＣ＿ｒｅｔ＜０　∧　ｆｕｎｃＣ＿ｘ＝＝ｘ＋１」に追加される。

【００２３】

そして、各関数について生成された条件を、統合処理部１０９が関数間で網羅的に統合する。例えば、Ａ－（２）の条件とｆｕｎｃＢについての各条件とを統合すると、図１０に示すようになる。図１０においては、条件に２つの括弧が含まれており、左側の括弧にはＡ－（２）の条件が含まれ、右側の括弧にはｆｕｎｃＢについての条件が含まれる。２つの括弧内の条件は、論理積によって結合されている。そして、条件の欄の右に設けられている判定の欄には、条件を満たす解が存在するか否かについての判定結果が示されている。ＳＡＴ（Satisfied）である場合には、条件を満たす解が存在する。ＵＮＳＡＴ（UNSATisfied）である場合には、条件を満たす解が存在しない。ＵＮＳＡＴの括弧内には、ＵＮＳＡＴである原因である条件が示されている。例えばパスＡ－（２）Ｂ－（１）については、ｆｕｎｃＢ＿ｒｅｔ＞０とｆｕｎｃＢ＿ｒｅｔ＝＝０とが同時に成立することが無いため、ＵＮＳＡＴである。

【００２４】

但し、図１０に示した条件は、ｆｕｎｃＢの下位関数であるｆｕｎｃＣに依存している

。そこで、図１０においてＳＡＴであると判定された条件（ここでは、Ａ－（２）Ｂ－（３）の条件）と、ｆｕｎｃＣについての各条件とをさらに統合すると、図１１に示すようになる。図１１においては、条件に３つの括弧が含まれており、左側の括弧にはＡ－（２）の条件が含まれ、真ん中の括弧にはＢ－（３）の条件が含まれ、右側の括弧にはｆｕｎｃＣについての条件が含まれる。３つの括弧内の条件は、論理積によって結合されている。そして、条件の欄の右に設けられている判定の欄には、条件を満たす解が存在するか否かについての判定結果が示されている。

【００２５】

また、Ａ－（３）の条件とｆｕｎｃＢについての各条件とを統合すると、図１２に示すようになる。図１２においては、条件に２つの括弧が含まれており、左側の括弧にはＡ－（３）の条件が含まれ、右側の括弧にはｆｕｎｃＢについての条件が含まれる。２つの括弧内の条件は、論理積によって結合されている。そして、条件の欄の右に設けられている判定の欄には、条件を満たす解が存在するか否かについての判定結果が示されている。

【００２６】

但し、図１２に示した条件は、ｆｕｎｃＢの下位関数であるｆｕｎｃＣに依存している。そこで、図１２においてＳＡＴであると判定された条件（ここでは、Ａ－（３）Ｂ－（２））と、ｆｕｎｃＣについての各条件とをさらに統合すると、図１３に示すようになる。図１３においては、条件に３つの括弧が含まれており、左側の括弧にはＡ－（３）の条件が含まれ、真ん中の括弧にはＢ－（２）の条件が含まれ、右側の括弧にはｆｕｎｃＣについての条件が含まれる。３つの括弧は、論理積によって結合されている。そして、条件の欄の右に設けられている判定の欄には、条件を満たす解が存在するか否かについての判定結果が示されている。

【００２７】

以上のようにして条件を統合すると、最終的にＳＡＴである条件は、図１４に示した条件である。すなわち、Ａ－（１）の条件、Ａ－（２）Ｂ－（３）Ｃ－（１）の条件、Ａ－（３）Ｂ－（２）Ｃ－（２）の条件、及びＡ－（３）Ｂ－（４）Ｃ－（２）の条件である。そして、これらの条件を満たすシンボル変数の値を含むテストデータが生成される。なお、このようなテストデータを用いて図３に示したプログラムのテストを実行すると、ｆｕｎｃＢにおける４行目の命令以外を網羅することができる。

【００２８】

なお、条件には、パス条件、引数についての条件、及び返却値についての条件の他に、以下で説明する処理によって別の条件が追加される場合もある。但し、説明を簡単にするため、上ではパス条件、引数についての条件、及び返却値についての条件のみを示した。

【００２９】

次に、図１５乃至図５１を用いて、情報処理装置１が実行する処理について詳細に説明する。

【００３０】

まず、情報処理装置１における入力部１０１は、ユーザからプログラムの入力を受け付け、プログラム格納部１０２に格納する。これに応じ、シンボリック実行部１０５は、処理対象の関数ｆをプログラム格納部１０２から特定する（図１５：ステップＳ１）。ステップＳ１においては、エントリポイントのプログラム（例えば、「ｍａｉｎ」のプログラム）が特定される。

【００３１】

シンボリック実行部１０５は、関数ｆに対して既にシンボリック実行を行ったか判断する（ステップＳ３）。関数ｆに対して既にシンボリック実行を行った場合（ステップＳ３：Ｙｅｓルート）、シンボリック実行部１０５は、関数ｆの最終更新の時点がシンボリック実行の時点よりも前であるか判断する（ステップＳ５）。関数ｆの最終更新の時点がシンボリック実行の時点よりも前である場合（ステップＳ５：Ｙｅｓルート）、既に行われたシンボリック実行の結果を採用することができる。そこで、シンボリック実行部１０５は、関数ｆに対するシンボリック実行の結果を、条件表格納部１０７から特定する（ステ

ップＳ７）。条件表格納部１０７には、例えば、図７乃至図１３に示したような条件表が格納されている。

【００３２】

一方、関数ｆに対して未だシンボリック実行を行っていない場合（ステップＳ３：Ｎｏルート）及び関数ｆの最終更新の時点がシンボリック実行の時点より後である場合（ステップＳ５：Ｎｏルート）、新たにシンボリック実行を行うことが求められる。そこで、シンボリック実行部１０５は、前処理部１０３に前処理の実行を指示する。これに応じ、前処理部１０３は、前処理を実行する（ステップＳ９）。前処理については、図１６乃至図１８を用いて説明する。但し、ドライバ及びスタブの生成はよく知られた処理であるので、簡単に説明する。

【００３３】

まず、前処理部１０３は、関数ｆから、変数及び関数ｆにおいて呼び出される関数（以下、呼び出し関数と呼ぶ）を抽出する（図１６：ステップＳ２１）。例えば図３に示したｆｕｎｃＡの場合、呼び出し関数としてｆｕｎｃＢが抽出される。

【００３４】

前処理部１０３は、ステップＳ２１において抽出された呼び出し関数についてスタブを生成し（ステップＳ２３）、スタブ格納部１０４２に格納する。スタブはドライバから呼び出され、呼び出しに応じてシンボルを返すプログラムである。図１７に、生成されるスタブの一例を示す。図１７に示したスタブの６行目において、ドライバに対してシンボルを返すようになっている。

【００３５】

前処理部１０３は、関数ｆの変数及び呼び出し関数に対応するシンボル変数を設定するためのプログラムであるドライバを生成し（ステップＳ２５）、ドライバ格納部１０４１に格納する。そして処理を終了する。図１８に、生成されるドライバの一例を示す。図１８に示したドライバにおいては、３行目及び４行目においてシンボル変数を設定し、５行目においてスタブを呼び出し、６行目においてｆｕｎｃＡを呼び出している。

【００３６】

以上のような処理を実行すれば、関数単体でシンボリック実行を行えるようになる。

【００３７】

図１５の説明に戻り、ドライバ及びスタブが生成されると、シンボリック実行部１０５は、関数ｆについてシンボリック実行を行う（ステップＳ１１）。ステップＳ１１の処理については、図１９乃至図４４を用いて説明する。なお、図１９にける「シンボリック実行（ｐ）」とは、実行状態ｐについてシンボリック実行を行うことを表している。

【００３８】

まず、シンボリック実行部１０５は、状態データ格納部１０６に格納されている、状態ｐのプログラムカウンタの値に基づき、命令を１つ読み出す（図１９：ステップＳ３１）。例えばプログラムカウンタの値が「２」である場合、関数ｆの２行目の命令を読み出す。以下では、ステップＳ３１において読み出された命令を「処理対象の命令」と呼ぶ。

【００３９】

シンボリック実行部１０５は、処理対象の命令が、関数を呼び出す命令であるか判断する（ステップＳ３３）。関数を呼び出す命令ではない場合（ステップＳ３３：Ｎｏルート）、ステップＳ４３の処理に移行する。

【００４０】

一方、関数を呼び出す命令である場合（ステップＳ３３：Ｙｅｓルート）、シンボリック実行部１０５は、呼び出される関数ｆｆの引数に関数ポインタが含まれるか判断する（ステップＳ３５）。関数ｆｆの引数に関数ポインタが含まれていない場合（ステップＳ３５：Ｎｏルート）、ステップＳ３９の処理に移行する。関数ｆｆの引数に関数ポインタが含まれている場合（ステップＳ３５：Ｙｅｓルート）、シンボリック実行部１０５は、関数ポインタについての条件を、実行中のパスについての条件に追加する（ステップＳ３７）。

【００４１】
図２０乃至図２５を用いて、ステップＳ３７の処理について説明する。例えば、プログラム格納部１０２に格納されているプログラムが図２０に示すプログラムであるとする。このプログラムにおいては、ｆｕｎｃＡがｆｕｎｃＢを呼び出し、ｆｕｎｃＢがｆｕｎｃＣを呼び出す。従って、このプログラムのコールツリーは、図２１に示すようになる。
【００４２】
このプログラムにおいては、ｆｕｎｃＡの４行目においてｆｕｎｃＢを呼び出しているが、ｆｕｎｃＢの引数には関数ポインタ（＊ｐ）が含まれている。このような場合においては、ｆｕｎｃＢ単体についてシンボリック実行を行ったとしても、ｆｕｎｃＢがｆｕｎｃＣに依存しているということを特定することができない。
【００４３】
そこで、図２２に示すように、ｆｕｎｃＡについてのパスのうちｆｕｎｃＢを呼び出すパスについては、「ｆｕｎｃＢ＿ｐ＝＝ｆｕｎｃＣ」という条件を追加する。これにより、関数ポインタが指す関数を、条件の統合時に特定できるようになる。
【００４４】
また、関数ポインタを呼び出す場合を区別できるようにするため、図２３に示すように、ｆｕｎｃＢについての条件において「ｆｕｎｃＣ」を「＊ｐ」に置き換える。
【００４５】
なお、ｆｕｎｃＣについての条件及び返却値は通常どおり生成されるので、図２４に示すようになる。
【００４６】
ここで、ｆｕｎｃＡについての条件と、ｆｕｎｃＢについての条件とを、後述の統合処理によって統合すると、図２５に示すようになる。図２５においては、条件に２つの括弧が含まれており、左側の括弧にはｆｕｎｃＡについての条件が含まれ、右側の括弧にはｆｕｎｃＢについての条件が含まれる。２つの括弧内の条件は、論理積によって結合されている。統合の際には、ｆｕｎｃＡについての条件に含まれる「ｆｕｎｃＢ＿ｐ＝＝ｆｕｎｃＣ」という条件を用いて、ｆｕｎｃＢについての条件に含まれる「＊ｐ」をｆｕｎｃＣに置き換える。
【００４７】
図１９の説明に戻り、シンボリック実行部１０５は、関数ｆｆの実引数と仮引数との関係を表す条件を、実行中のパスについての条件に追加する（ステップＳ３９）。ステップＳ３９において追加される条件は、例えば、図８のＢ－（３）における「ｆｕｎｃＣ＿ｘ＝＝ｘ＋１」が該当する。
【００４８】
なお、関数ｆｆが複数回呼び出される場合、ステップＳ３９においては、引数についての条件が呼び出し毎に追加される。これについては、図２６乃至図３２を用いて説明する。
【００４９】
例えば図２６に示すプログラムにおいては、ｆｕｎｃＡの４行目及び５行目の処理によって、ｆｕｎｃＢが３回呼び出される。従って、このプログラムのコールツリーは、図２７に示すようになる。
【００５０】
このような場合、図２８に示すように、ｆｕｎｃＡについての条件には、ｆｕｎｃＢの引数についての条件がｆｕｎｃＢの呼び出し毎に追加される。すなわち、Ａ－（２）及びＡ－（３）の条件に対しては「ｆｕｎｃＢ＿０＿ｘ＝＝ｘ」という条件、「ｆｕｎｃＢ＿１＿ｘ＝＝ｘ＋１」という条件、及び「ｆｕｎｃＢ＿２＿ｘ＝＝ｘ＋２」という条件が追加される。
【００５１】
ｆｕｎｃＢについての条件は、図２９に示すように、呼び出し毎に条件表を生成する。図２６のプログラムの場合、ｆｕｎｃＢについての条件表は３つ生成される

【００５２】
そして、ｆｕｎｃＡについての条件と、ｆｕｎｃＢについての呼び出し毎の条件とを、後述の統合処理によって統合する。図３０に、Ａ－（２）とｆｕｎｃＢについての条件とを統合することによって得られる条件を示す。図３０においては、条件に４つの括弧が含まれており、左から１番目の括弧にはＡ－（２）の条件が含まれ、左から２番目の括弧には１回目の呼び出しについての条件が含まれ、左から３番目の括弧には２回目の呼び出しについての条件が含まれ、左から４番目の括弧には３回目の呼び出しについての条件が含まれる。４つの括弧内の条件は、論理積によって結合されている。そして、条件の欄の右に設けられている判定の欄には、条件を満たす解が存在するか否かについての判定結果が示されている。

【００５３】
また、Ａ－（３）とｆｕｎｃＢについての呼び出し毎の条件とを、後述の統合処理によって統合することによって得られる条件は、図３１に示すようになる。図３１においては、条件に４つの括弧が含まれており、左から１番目の括弧にはＡ－（３）の条件が含まれ、左から２番目の括弧には１回目の呼び出しについての条件が含まれ、左から３番目の括弧には２回目の呼び出しについての条件が含まれ、左から４番目の括弧には３回目の呼び出しについての条件が含まれる。４つの括弧内の条件は、論理積によって結合されている。そして、条件の欄の右に設けられている判定の欄には、条件を満たす解が存在するか否かについての判定結果が示されている。

【００５４】
よって、判定がＳＡＴである条件は、図３２に示したＡ－（１）の条件、Ａ－（３）Ｂ０－（２）Ｂ１－（２）Ｂ２－（１）の条件、Ａ－（３）Ｂ０－（２）Ｂ１－（１）Ｂ２－（２）の条件、Ａ－（３）Ｂ０－（１）Ｂ１－（２）Ｂ２－（２）の条件、及びＡ－（３）Ｂ０－（２）Ｂ１－（２）Ｂ２－（２）の条件である。これらの５つの条件についてテストデータを求めると、ｘ＝０、１８、１９、２０、１となる。

【００５５】
図１９の説明に戻り、シンボリック実行部１０５は、関数ｆｆをコールツリーに追加するための追加処理を実行する（ステップＳ４１）。追加処理については、図３３乃至図３６を用いて説明する。

【００５６】
まず、シンボリック実行部１０５は、コールツリー格納部１０８に格納されているコールツリーに最も後に追加された関数の位置を現在位置に設定する（図３３：ステップＳ７１）。

【００５７】
シンボリック実行部１０５は、関数ｆｆはコールツリーにおける現在位置より上位の位置に存在するか判断する（ステップＳ７３）。

【００５８】
現在位置より上位の位置に存在しない場合（ステップＳ７３：Ｎｏルート）、再帰呼び出しには該当しないので、シンボリック実行部１０５は、コールツリーにおける現在位置の下に関数ｆｆを追加する（ステップＳ７５）。一方、現在位置より上位の位置に存在する場合（ステップＳ７３：Ｙｅｓルート）、再帰呼び出しに該当する。そこで、シンボリック実行部１０５は、関数ｆｆについての条件を、実行中のパスについての条件から削除する（ステップＳ７７）。そして呼び出し元の処理に戻る。

【００５９】
図３４乃至図３６を用いて、追加処理についてより具体的に説明する。例えば、図３４に示すようなプログラム「ｒｅｃ３」があるとする。ｒｅｃ３においては、４行目においてｒｅｃ１を呼び出している。ｒｅｃ３が呼び出される前にｒｅｃ１が呼び出されていないのであれば、再帰呼び出しには該当しないので問題は無い。しかしながら、例えば図３５の左側のコールツリーに示すように、ｒｅｃ１においてｒｅｃ２を呼び出し、ｒｅｃ２においてｒｅｃ３を呼び出し、ｒｅｃ３においてｒｅｃ１を呼び出すという呼び出し関係

があるとする。このような場合には再帰呼び出しに該当するので、処理が終わらなくなる可能性がある。

【００６０】

そこで、コールツリーにおける２回目のｒｅｃ１を削除することにより、図３５の左側のコールツリーを右側のコールツリーに示すように変える。また、図３６に示すように、直前のステップＳ３９等において追加されたｒｅｃ１についての条件を、実行中のパスについての条件から削除する。具体的には、Ａ－（２）から「ｒｅｃ１＿ｒｅｔ＞０　∧　ｒｅｃ１＿ｘ＝＝ｘ」を削除し、Ａ－（３）から「ｒｅｃ１＿ｒｅｔ＜＝０　∧　ｒｅｃ１＿ｘ＝＝ｘ」を削除する。

【００６１】

以上のような処理を実行すれば、再帰呼び出しによって同じ処理が繰り返し行われてしまうことを防げるようになる。

【００６２】

図１９の説明に戻り、シンボリック実行部１０５は、処理対象の命令がグローバル変数を読み込む命令又は書き込む命令であるか判断する（ステップＳ４３）。グローバル変数を読み込む命令又は書き込む命令のいずれでもない場合（ステップＳ４３：Ｎｏルート）、処理は端子Ａを介して図４４のステップＳ４９に移行する。一方、グローバル変数を読み込む命令又は書き込む命令である場合（ステップＳ４３：Ｙｅｓルート）、ステップＳ４５の処理に移行する。そして、グローバル変数を読み込む命令である場合、シンボリック実行部１０５は、読み込みに対応する書き込みの命令を、コールツリー格納部１０８に格納されているコールツリーによって特定する（ステップＳ４５）。なお、グローバル変数を書き込む命令である場合には、ステップＳ４５の処理は行われない。

【００６３】

シンボリック実行部１０５は、ステップＳ４３においてグローバル変数の読み込みであると判定された場合にはグローバル変数の読み込みについての条件を、ステップＳ４３においてグローバル変数の書き込みであると判定された場合にはグローバル変数の書き込みについての条件を、実行中のパスについての条件に追加する（ステップＳ４７）。処理は端子Ａを介して図４４のステップＳ４９に移行する。

【００６４】

図３７乃至図４３を用いて、ステップＳ４３乃至Ｓ４７において実行される処理について具体的に説明する。例えば図３７に示すようなプログラムが有るとする。図３７のプログラムにおいては、プログラム３７０においてグローバル変数ｚが定義されている。また、ｆｕｎｃＡの３行目乃至５行目の処理によってａｄｄｚ及びｆｕｎｃＢが２回呼び出され、ｆｕｎｃＢにおいてｆｕｎｃＣが呼び出される。従って、このプログラムのコールツリーは、図３８に示すようになる。

【００６５】

ステップＳ４５においては、図３８に示すように、グローバル変数ｚの読み込みに対応する書き込みを特定する。これにより、グローバル変数ｚの書き込みが複数回行われる場合であっても、それぞれの書き込みに対応する読み込みの条件を生成できるようになる。なお、コールツリーは左側から順に生成されることを前提とする。

【００６６】

従って、図３７に示したプログラム３７０について生成される条件は図３９に示すようになり、図３７に示したｆｕｎｃＡについて生成される条件及び返却値は図４０に示すようになり、図３７に示したｆｕｎｃＢについて生成される返却値は図４１に示すようになり、図３７に示したｆｕｎｃＣについて生成される条件及び返却値は図４２に示すようになる。本例においては、プログラム３７０、ｆｕｎｃＢ、及びｆｕｎｃＣが２回呼び出されているため、呼び出し毎に条件が生成されるようになっている。また、ステップＳ４５の処理によって、書き込みについての条件であるＺ０－（１）には読み込みについての条件であるＣ０－（１）及びＣ０－（２）が対応付けられ、書き込みについての条件であるＺ１－（１）には読み込みについての条件であるＣ１－（１）及びＣ１－（２）が対応付

けられる。

【００６７】

そして、図３９乃至図４２に示した条件及び返却値についての条件を、後述の統合処理によって統合すると、図４３に示すようになる。図４３においては、ｆｕｎｃＡについての条件と、プログラム３７０についての呼び出し毎の条件と、ｆｕｎｃＢについての呼び出し毎の条件と、ｆｕｎｃＣについての呼び出し毎の条件とが統合されている。そして、条件の欄の右に設けられている判定の欄には、条件を満たす解が存在するか否かについての判定結果が示されている。

【００６８】

図４４の説明に移行し、シンボリック実行部１０５は、処理対象の命令がシンボルに関わる分岐の命令であるか判断する（図４４：ステップＳ４９）。シンボルに関わる分岐の命令とは、例えば、ｉｆ文にシンボル変数が含まれている命令である。

【００６９】

シンボルに関わる分岐の命令である場合（ステップＳ４９：Ｙｅｓルート）、シンボリック実行部１０５は、新たにシンボリック実行が行われる実行状態Ｐに実行状態ｐ（すなわち、状態データ格納部１０６にデータが格納されている現在の実行状態）をコピーする（ステップＳ５１）。

【００７０】

シンボリック実行部１０５は、処理対象の命令において未処理の分岐先を１つ特定する（ステップＳ５３）。そして、シンボリック実行部１０５は、分岐の条件（すなわち、パス条件）を、実行中のパスについての条件に追加する（ステップＳ５５）。

【００７１】

シンボリック実行部１０５は、実行状態Ｐについてのシンボリック実行を再帰的に行う（ステップＳ５７）。

【００７２】

ステップＳ５７におけるシンボリック実行が完了すると、シンボリック実行部１０５は、処理対象の命令において未処理の分岐先が有るか判断する（ステップＳ５９）。未処理の分岐先が有る場合（ステップＳ５９：Ｙｅｓルート）、次の分岐先について処理するため、ステップＳ５３の処理に戻る。一方、未処理の分岐先が無い場合（ステップＳ５９：Ｎｏルート）、呼び出し元の処理に戻る。

【００７３】

一方、処理対象の命令がシンボルに関わる分岐ではない場合（ステップＳ４９：Ｎｏルート）、新たにシンボリック実行を行わなくてもよい。従って、シンボリック実行部１０５は、処理対象の命令を実行し、状態データ格納部１０６に格納されているプログラムカウンタの値を１進める（ステップＳ６１）。

【００７４】

シンボリック実行部１０５は、実行状態ｐは終了の状態であるか判断する（ステップＳ６３）。終了の状態とは、例えば、プログラムカウンタの値がプログラムにおける最後の行の番号に達している状態である。終了の状態ではない場合（ステップＳ６３：Ｎｏルート）、次の命令について処理するため、ステップＳ３１の処理に戻る。

【００７５】

一方、実行状態ｐは終了の状態である場合（ステップＳ６３：Ｙｅｓルート）、シンボリック実行部１０５は、シンボリック実行によって得られた、各パスについての条件及び返却値を含む条件表を条件表格納部１０７に格納する（ステップＳ６５）。そして呼び出し元の処理に戻る。

【００７６】

以上のような処理を実行すれば、各関数についてシンボリック実行を行い、満たすべき条件及び関数の返却値についてのデータを条件表格納部１０７に保存できるようになる。なお、各関数についてのシンボリック実行の結果（すなわち、条件表）は、条件表格納部１０７に保存され、ステップＳ７の処理において再利用することができる。

【００７７】
図１５の説明に戻り、シンボリック実行部１０５は、統合処理の実行を統合処理部１０９に指示する。これに応じ、統合処理部１０９は、条件表格納部１０７に格納されている条件表を用いて統合処理を実行する（ステップＳ１３）。統合処理については、図４５を用いて説明する。なお、上で述べたように、統合処理部１０９は、統合処理を実行するにあたり条件表格納部１０７における返却値を条件に変換する。
【００７８】
まず、統合処理部１０９は、条件表格納部１０７に、統合すべき２つの条件表（ここでは、条件表Ａ及び条件表Ｂとする）が有るか判断する（図４５：ステップＳ８１）。統合すべき２つの条件表Ａ及びＢが存在しない場合（ステップＳ８１：Ｎｏルート）、呼び出し元の処理に戻る。
【００７９】
一方、統合すべき２つの条件表Ａ及びＢが存在する場合（ステップＳ８１：Ｙｅｓルート）、統合処理部１０９は、条件表Ａから未処理の条件（ここでは、条件ａとする）を１つ特定する（ステップＳ８３）。また、統合処理部１０９は、条件表Ｂから未処理の条件（ここでは、条件ｂとする）を１つ特定する（ステップＳ８３）。ステップＳ８３及びＳ８５において、統合処理部１０９は、パス単位で条件を特定する。
【００８０】
統合処理部１０９は、条件ａと条件ｂとの論理積を求めることにより、条件ｃを生成し（ステップＳ８７）、メインメモリ等の記憶装置に格納する。
【００８１】
統合処理部１０９は、条件ｃを満たす解が存在するか、例えばソルバによって判断する（ステップＳ８９）。条件ｃを満たす解が存在しない場合（ステップＳ８９：Ｎｏルート）、ステップＳ９３の処理に移行する。一方、条件ｃを満たす解が存在する場合（ステップＳ８９：Ｙｅｓルート）、統合処理部１０９は、条件ｃを条件表Ｃに追加する（ステップＳ９１）。なお、条件表Ｃが存在しない場合には、ステップＳ９１において新たに条件表Ｃが生成される。
【００８２】
統合処理部１０９は、条件表Ｂに未処理の条件が有るか判断する（ステップＳ９３）。未処理の条件が有る場合（ステップＳ９３：Ｙｅｓルート）、次の条件について処理するため、ステップＳ８５の処理に戻る。一方、条件表Ｂに未処理の条件が無い場合（ステップＳ９３：Ｎｏルート）、統合処理部１０９は、条件表Ｂにおける条件を全て未処理に設定する（ステップＳ９５）。
【００８３】
統合処理部１０９は、条件表Ａに未処理の条件が有るか判断する（ステップＳ９７）。未処理の条件が有る場合（ステップＳ９７：Ｙｅｓルート）、次の条件について処理するため、ステップＳ８３の処理に戻る。一方、条件表Ａに未処理の条件が無い場合（ステップＳ９７：Ｎｏルート）、統合処理部１０９は、条件表Ｃを条件表格納部１０７に格納し、呼び出し元の処理に戻る。
【００８４】
以上のような処理を実行すれば、各関数について生成された条件をまとめることができるようになる。
【００８５】
図１５の説明に戻り、統合処理部１０９は、シンボリック実行部１０５に、統合処理の完了を通知する。これに応じ、シンボリック実行部１０５は、統合後の条件が下位関数に依存しているか判断する（ステップＳ１５）。統合後の条件が下位関数に依存しているとは、統合後の条件に下位関数が含まれるということである。統合後の条件が下位関数に依存している場合（ステップＳ１５：Ｙｅｓルート）、処理対象の関数ｆをその下位関数とし（ステップＳ１７）、ステップＳ３の処理に戻る。
【００８６】

統合後の条件が下位関数に依存していない場合（ステップＳ１５：Ｎｏルート）、シンボリック実行部１０５は、テストデータ生成部１１０に処理の実行を指示する。これに応じ、テストデータ生成部１１０は、統合後の条件を満たすシンボル変数の値をソルバによって算出し、算出されたシンボル変数の値を含むテストデータを生成し（ステップＳ１９）、テストデータ格納部１１１に格納する。そして処理を終了する。なお、テストデータ格納部１１１に格納されるデータは、例えば図１４に示したようなデータである。

【００８７】

以上のような処理を実行すれば、シンボリック実行時に使用する計算資源の量を減らせるようになる。具体的には、シンボリック実行中においてメモリに保持するデータの量を減らすことができるようになる。

【００８８】

これについて、図４６及び図４７を用いて説明する。本実施の形態においては、引数毎に生成された条件及び返却値についての条件によってメモリ使用量は増えるが、状態保存のためのメモリ使用量は減る。

【００８９】

まず、本実施の形態におけるメモリ使用量の増加について検討する。関数ｆにおける引数の数をn_arg_fとし、関数ｆの返却値に含まれる変数の数をn_ret_fとし、関数ｆにおけるパス条件の数を$n_pathcond_f$とし、１つの条件ごとのメモリ使用量をＭとする。すると、メモリ使用量の増加分は以下の式によって算出される。

【００９０】

【数１】

$$\sum_f n\_pathcond_f \times \left(n\_\arg_f + n\_ret_f\right) \times M$$

【００９１】

但し、状態保存のためのメモリ使用量が支配的であれば、上記の要因で増えるメモリ使用量の増加は無視できる。そこで、（１）状態保存のためのメモリ使用量が支配的であること、（２）深さ優先の探索によってプログラムを実行することを前提として、メモリ使用量の減少について検討する。

【００９２】

以下では、１番目の関数を関数f_1とし、ｋ番目に呼ばれる関数f_kの深さをd_kとし、関数f_kにおける各パスの状態を、

【数２】

$$s_{(f_1 \cdots f_k),1}, \cdots, s_{(f_1 \cdots f_k),d_k}$$

とする。また、状態ｓにおけるメモリ使用量をＭ（ｓ）とする。

【００９３】

まず、通常のシンボリック実行について検討する。通常のシンボリック実行において最もメモリ使用量が多くなるのは、一番深い関数についてシンボリック実行が行われている時である。この時、図４６に示すように、深さd_1において状態$s_{f1,1}$・・・$s_{f1,d1}$についてメモリを使用しており、・・・、深さd_nにおいて状態$s_{(f1\cdots fn),1}$・・・$s_{(f1\cdots fn),dn}$についてメモリを使用している。なお、ｆ及びｄの添え字は本来は下付きであるが、ここではファイル形式の都合により下付きにはしていない。この場合、メモリ使用量は以下の式によって算出される。

【００９４】

【数３】

$$\sum_{i=1}^{n} \sum_{j=1}^{d_i} M\left(s_{(f_1 \cdots f_i),j}\right)$$

【００９５】
次に、本実施の形態の方法について検討する。本実施の形態の方法において最もメモリ使用量が多くなるのは、$f_1 \cdots f_n$の中で最も深い位置にある関数f_mについてシンボリック実行を行っている時である。この時、図４７に示すように、深さd_mにおいて状態$s_{f_m,1} \cdots s_{f_m,d_m}$についてメモリを使用している。この場合、メモリ使用量は以下の式によって算出される。

【００９６】
【数４】

$$\sum_{j=1}^{d_m} M\left(s_{f_m,j}\right)$$

【００９７】
ここで、両者を比較する。或るｋ≦ｎにおいて、状態$s_{(f_1 \cdots f_k),d_k}$はスタックに$f_1 \cdots f_k$のコンテキストを持つため、状態s_{f_k,d_k}よりもメモリ使用量が多くなり、以下の式が成立する。

【００９８】
【数５】

$$\sum_{j=1}^{d_k} M\left(s_{(f_1 \cdots f_k),j}\right) \geq \sum_{j=1}^{d_k} M\left(s_{f_k,j}\right)$$

【００９９】
また、スタック$f_1 \cdots f_n$のコンテキストを保持することになるf_nにおいて深さが最大となる場合、以下の式によって算出される値が最大になる。

【０１００】
【数６】

$$\sum_{j=1}^{d_n} M\left(s_{(f_1 \cdots f_n),j}\right) - \sum_{j=1}^{d_n} M\left(s_{f_n,j}\right)$$

【０１０１】
よって、最悪の場合であっても、本実施の形態の方が以下の量だけメモリ使用量が少ない。

【０１０２】
【数７】

$$\sum_{i=1}^{n}\sum_{j=1}^{d_i} M\left(s_{(f_1 \cdots f_i),j}\right) - \sum_{j=1}^{d_m} M\left(s_{f_m,j}\right)$$

$$\geq \sum_{i=1}^{n}\sum_{j=1}^{d_i} M\left(s_{(f_1 \cdots f_i),j}\right) - \sum_{j=1}^{d_n} M\left(s_{(f_1 \cdots f_n),j}\right)$$

$$= \sum_{i=1}^{n-1}\sum_{j=1}^{d_i} M\left(s_{(f_1 \cdots f_i),j}\right)$$

【０１０３】
また、本実施の形態によれば、シンボリック実行の量を減らすことができるので、ＣＰＵの負荷が減少する。

【０１０４】
これについて、図４８及び図４９を用いて説明する。まず、図４８に示すような、ｎ（ｎは自然数）個の分岐があるプログラムがあるとする。このプログラムにおいては、ｎ回の分岐を経た後に、関数ｆｕｎｃ（ｘ）を実行する。通常のシンボリック実行においては

、それぞれのパスにおいて関数ｆｕｎｃ（ｘ）を実行するため、関数ｆｕｎｃ（ｘ）の実行回数はｎ回である。一方、本実施の形態の方法であれば、関数毎にシンボリック実行を行うため、関数ｆｕｎｃ（ｘ）の実行回数は１回であるので、実行回数を減らすことができる。

【０１０５】

また、図４９に示すような、同じ関数をｎ回実行するプログラムがあるとする。このプログラムにおいては、ｆｕｎｃ（ｘ＋ｉ）がｎ回実行される。通常のシンボリック実行においては、コードの実行回数はｎ回である。これに対し、本実施の形態の方法であれば、コードの実行回数は１回であるので、実行回数を減らすことができる。但し、関数の入力がそれぞれの実行において異なる場合には、それぞれの条件（図４９の例の場合、ｆｕｎｃ＿ｘ＝＝ｘ、ｆｕｎｃ＿ｘ＝＝ｘ＋１・・・）を保存することになるため、メモリ使用量は増加することがある。

【０１０６】

さらに、本実施の形態においては、関数毎の実行結果が条件表格納部１０７に保存されているため、関数のコードを変更した場合においてもシンボリック実行を最小限の計算でやり直すことができる。

【０１０７】

例えば、図３に示したｆｕｎｃＣを、図５０に示すように変更したとする。図５０のｆｕｎｃＣは、図３と比較すると、４行目及び５行目に分岐が追加されている。このような場合には、変更後のｆｕｎｃＣについてのみ再度シンボリック実行を行い、図５１に示すような条件表を生成する。これに対し、ｆｕｎｃＡ及びｆｕｎｃＢについては変更が無いので、条件表格納部１０７に保存されている実行結果を再利用する。そして、ｆｕｎｃＡ及びｆｕｎｃＢについての条件と、変更後のｆｕｎｃＣについての条件とを統合する。また、ｆｕｎｃＡについての条件とｆｕｎｃＢについての条件とを統合した結果が条件表格納部１０７に格納されている場合には、統合後の条件と変更後のｆｕｎｃＣについての条件とを統合すれば済むので、統合の処理をも減らすことができるようになる。

【０１０８】

以上本発明の一実施の形態を説明したが、本発明はこれに限定されるものではない。例えば、上で説明した情報処理装置１の機能ブロック構成は実際のプログラムモジュール構成に一致しない場合もある。

【０１０９】

また、上で説明した各テーブルの構成は一例であって、上記のような構成でなければならないわけではない。さらに、処理フローにおいても、処理結果が変わらなければ処理の順番を入れ替えることも可能である。さらに、並列に実行させるようにしても良い。

【０１１０】

なお、上で述べた情報処理装置１は、コンピュータ装置であって、図５２に示すように、メモリ２５０１とＣＰＵ（Central Processing Unit）２５０３とハードディスク・ドライブ（ＨＤＤ：Hard Disk Drive）２５０５と表示装置２５０９に接続される表示制御部２５０７とリムーバブル・ディスク２５１１用のドライブ装置２５１３と入力装置２５１５とネットワークに接続するための通信制御部２５１７とがバス２５１９で接続されている。オペレーティング・システム（ＯＳ：Operating System）及び本実施例における処理を実施するためのアプリケーション・プログラムは、ＨＤＤ２５０５に格納されており、ＣＰＵ２５０３により実行される際にはＨＤＤ２５０５からメモリ２５０１に読み出される。ＣＰＵ２５０３は、アプリケーション・プログラムの処理内容に応じて表示制御部２５０７、通信制御部２５１７、ドライブ装置２５１３を制御して、所定の動作を行わせる。また、処理途中のデータについては、主としてメモリ２５０１に格納されるが、ＨＤＤ２５０５に格納されるようにしてもよい。本発明の実施例では、上で述べた処理を実施するためのアプリケーション・プログラムはコンピュータ読み取り可能なリムーバブル・ディスク２５１１に格納されて頒布され、ドライブ装置２５１３からＨＤＤ２５０５にインストールされる。インターネットなどのネットワーク及び通信制御部２５１７を経由し

て、ＨＤＤ２５０５にインストールされる場合もある。このようなコンピュータ装置は、上で述べたＣＰＵ２５０３、メモリ２５０１などのハードウエアとＯＳ及びアプリケーション・プログラムなどのプログラムとが有機的に協働することにより、上で述べたような各種機能を実現する。

【０１１１】

以上述べた本発明の実施の形態をまとめると、以下のようになる。

【０１１２】

本実施の形態に係るテストデータ生成方法は、（Ａ）プログラムにおいて実行される複数の関数の各々についてシンボリック実行を行うと共に、シンボル変数に関わる分岐を辿った履歴を表す条件と、当該関数の引数及び返却値についての条件とを含む実行条件を複数の関数の各々について生成し、（Ｂ）生成された複数の実行条件を論理積によって統合し、（Ｃ）統合後の条件を満たすシンボル変数の値を算出し、算出された当該シンボル変数の値を含むテストデータを生成する処理を含む。

【０１１３】

このようにすれば、シンボリック実行の際に保持するデータの量及び計算量が少なくなるので、計算資源の使用量を減らすことができるようになる。

【０１１４】

また、上で述べた複数の関数の各々について、シンボル変数を設定するためのプログラムと当該関数において呼び出される関数のスタブとを生成してもよい。このようにすれば、単一の関数について適切にシンボリック実行を行えるようになる。

【０１１５】

また、上で述べた実行条件を生成する処理において、（ａ１）関数の引数に関数ポインタが含まれる場合に、当該関数ポインタと当該関数ポインタが指す関数との関係を表す条件を、実行条件に追加してもよい。このようにすれば、条件の統合時に、関数ポインタが指す関数を特定できるようになる。

【０１１６】

また、上で述べた実行条件を生成する処理において、（ａ２）プログラムにグローバル変数が含まれる場合、当該グローバル変数の読み込みについての条件及び書き込みについての条件を、実行条件に追加してもよい。このようにすれば、グローバル変数が使用される場合にも適切に対処できるようになる。

【０１１７】

また、上で述べた実行条件を生成する処理において、（ａ３）関数において同一の関数を複数回呼び出す場合、呼び出される当該関数の引数についての条件を複数回の呼び出し毎に生成し、実行条件に追加してもよい。このようにすれば、考慮すべき条件の取りこぼしを防げるようになる。

【０１１８】

また、上で述べた実行条件を生成する処理において、（ａ４）関数間の関係を表すコールツリーを生成し、（ａ５）コールツリーの端に追加された関数が、再帰呼び出しされる関数である場合、コールツリーの端に追加された関数について生成された条件を削除してもよい。このようにすれば、不要な条件が追加されることを防げるようになる。

【０１１９】

なお、上記方法による処理をコンピュータに行わせるためのプログラムを作成することができ、当該プログラムは、例えばフレキシブルディスク、ＣＤ－ＲＯＭ、光磁気ディスク、半導体メモリ、ハードディスク等のコンピュータ読み取り可能な記憶媒体又は記憶装置に格納される。尚、中間的な処理結果はメインメモリ等の記憶装置に一時保管される。

【０１２０】

以上の実施例を含む実施形態に関し、さらに以下の付記を開示する。

【０１２１】

（付記１）

プログラムにおいて実行される複数の関数の各々についてシンボリック実行を行うと共

に、シンボル変数に関わる分岐を辿った履歴を表す条件と、当該関数の引数及び返却値についての条件との少なくともいずれかを含む実行条件を前記複数の関数の各々について生成し、
生成された複数の実行条件を論理積によって統合し、
統合後の条件を満たすシンボル変数の値を算出し、算出された当該シンボル変数の値を含むテストデータを生成する
処理をコンピュータに実行させるためのテストデータ生成プログラム。
【０１２２】
（付記２）
前記複数の関数の各々について、シンボル変数を設定するためのプログラムと当該関数において呼び出される関数のスタブとを生成する
処理をさらに実行させるための付記１記載のテストデータ生成プログラム。
【０１２３】
（付記３）
前記実行条件を生成する処理において、
前記関数の引数に関数ポインタが含まれる場合に、当該関数ポインタと当該関数ポインタが指す関数との関係を表す条件を、前記実行条件に追加する
ことを特徴とする付記１又は２記載のテストデータ生成プログラム。
【０１２４】
（付記４）
前記実行条件を生成する処理において、
前記プログラムにグローバル変数が含まれる場合、当該グローバル変数の読み込みについての条件及び書き込みについての条件を、前記実行条件に追加する
ことを特徴とする付記１乃至３のいずれか１つ記載のテストデータ生成プログラム。
【０１２５】
（付記５）
前記実行条件を生成する処理において、
前記関数において同一の関数を複数回呼び出す場合、呼び出される当該関数の引数についての条件を前記複数回の呼び出し毎に生成し、前記実行条件に追加する
ことを特徴とする付記１乃至４のいずれか１つ記載のテストデータ生成プログラム。
【０１２６】
（付記６）
前記実行条件を生成する処理において、
関数間の関係を表すコールツリーを生成し、
前記コールツリーの端に追加された関数が、再帰呼び出しされる関数である場合、前記コールツリーの端に追加された関数について生成された条件を削除する
ことを特徴とする付記１記載のテストデータ生成プログラム。
【０１２７】
（付記７）
プログラムにおいて実行される複数の関数の各々についてシンボリック実行を行うと共に、シンボル変数に関わる分岐を辿った履歴を表す条件と、当該関数の引数及び返却値についての条件との少なくともいずれかを含む実行条件を前記複数の関数の各々について生成し、
生成された複数の実行条件を論理積によって統合し、
統合後の条件を満たすシンボル変数の値を算出し、算出された当該シンボル変数の値を含むテストデータを生成する
処理をコンピュータが実行するテストデータ生成方法。
【０１２８】
（付記８）
プログラムにおいて実行される複数の関数の各々についてシンボリック実行を行うと共

に、シンボル変数に関わる分岐を辿った履歴を表す条件と、当該関数の引数及び返却値についての条件との少なくともいずれかを含む実行条件を前記複数の関数の各々について生成する第１処理部と、

生成された複数の実行条件を論理積によって統合する第２処理部と、

統合後の条件を満たすシンボル変数の値を算出し、算出された当該シンボル変数の値を含むテストデータを生成する第３処理部と、

を有するテストデータ生成装置。

【符号の説明】

【０１２９】

１　情報処理装置　　１０１　入力部

１０２　プログラム格納部　　１０３　前処理部

１０４１　ドライバ格納部　　１０４２　スタブ格納部

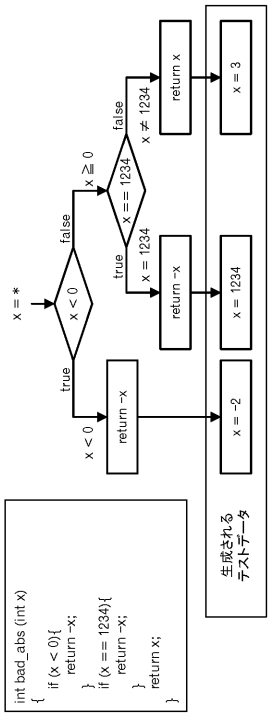
１０５　シンボリック実行部　　１０６　状態データ格納部

１０７　条件表格納部　　１０８　コールツリー格納部

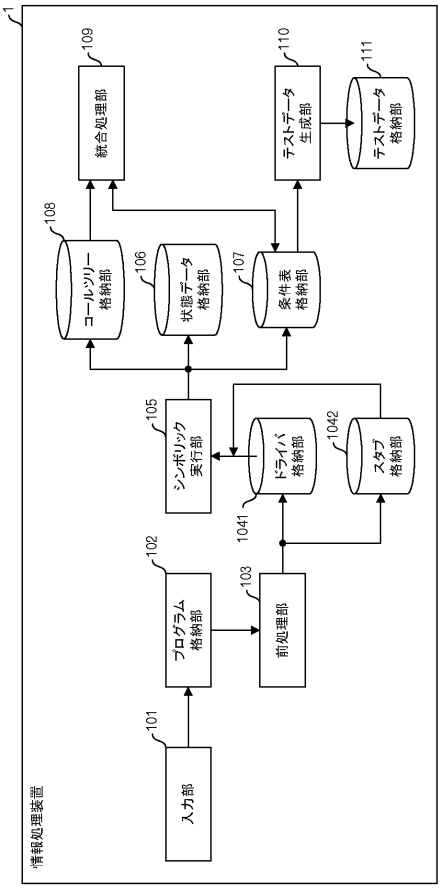
１０９　統合処理部　　１１０　テストデータ生成部

１１１　テストデータ格納部

【図１】



【図２】



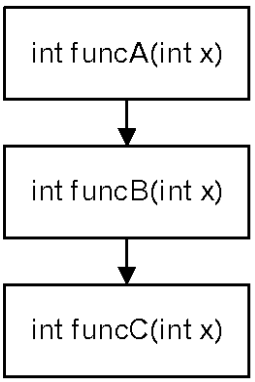
【図３】

```
1: int funcA(int x){
2:   int y = 0;
3:   if(x > 0){
4:     y = funcB(x);
5:   }
6:   if(y > 0){
7:     return 1;
8:   }
9:   return 0;
10: }
```

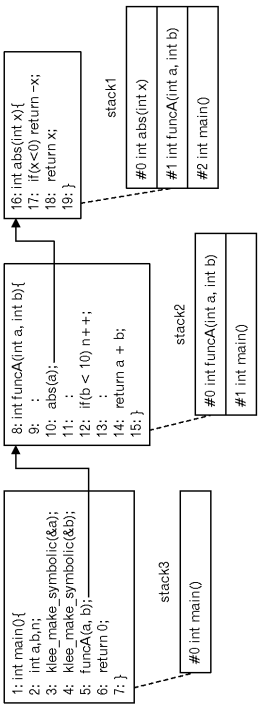
```
1: int funcB(int x){
2:   int y = 0;
3:   if(x == 0){
4:     return 0;
5:   }
6:   if(x > 10){
7:     y = funcC(x+1);
8:   } else {
9:     y = funcC(x-1);
10:  }
11:  if(y < 0){
12:    return -1;
13:  }
14:  return y;
15: }
```

```
1: int funcC(int x){
2:   if(x == 20){
3:     return 1;
4:   }
5:   return -1;
6: }
```

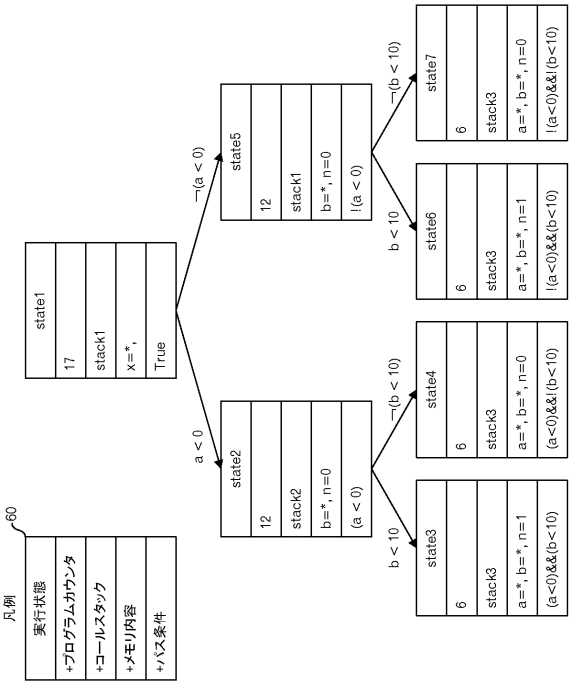
【図４】



【図５】



【図６】



【図７】

ID	条件	返却値
A-(1)	x<=0	0
A-(2)	x>0 ∧ funcB_ret>0 ∧ funcB_x==x	1
A-(3)	x>0 ∧ funcB_ret<=0 ∧ funcB_x==x	0

【図８】

ID	条件	返却値
B-(1)	x==0	0
B-(2)	x!=0 ∧ x>10 ∧ funcC_ret<0 ∧ funcC_x==x+1	-1
B-(3)	x!=0 ∧ x>10 ∧ funcC_ret>=0 ∧ funcC_x==x+1	funcC_ret
B-(4)	x!=0 ∧ x<=10 ∧ funcC_ret<0 ∧ funcC_x==x-1	-1
B-(5)	x!=0 ∧ x<=10 ∧ funcC_ret>=0 ∧ funcC_x==x-1	funcC_ret

【図９】

ID	条件	返却値
C-(1)	x==20	1
C-(2)	x!=20	-1

【図１０】

ID	条件	判定
A-(2)B-(1)	(x>0 ∧ funcB_ret>0 ∧ funcB_x==x) ∧ (funcB_x==0 ∧ funcB_ret==0)	UNSAT (funcB_ret>0 ∧ funcB_ret==0)
A-(2)B-(2)	(x>0 ∧ funcB_ret>0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret<0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==-1)	UNSAT (funcB_ret>0 ∧ funcB_ret==-1)
A-(2)B-(3)	(x>0 ∧ funcB_ret>0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret>=0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==funcC_ret)	SAT
A-(2)B-(4)	(x>0 ∧ funcB_ret>0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x<=10 ∧ funcC_ret<0 ∧ funcC_x==funcB_x-1 ∧ funcB_ret==-1)	UNSAT (funcB_ret>0 ∧ funcB_ret==-1)
A-(2)B-(5)	(x>0 ∧ funcB_ret>0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x<=10 ∧ funcC_ret>=0 ∧ funcC_x==funcB_x-1 ∧ funcB_ret==funcC_ret)	SAT

【図１１】

ID	条件	判定
A-(2)B-(3) C-(1)	(x>0 ∧ funcB_ret>0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret>=0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==funcC_ret) ∧ (funcC_x==20 ∧ funcC_ret==1)	SAT
A-(2)B-(3) C-(2)	(x>0 ∧ funcB_ret>0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret>=0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==funcC_ret) ∧ (funcC_x!=20 ∧ funcC_ret==-1)	UNSAT (funcC_ret>=0 ∧ funcC_ret==-1)

【図１２】

ID	条件	判定
A-(3)B-(1)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x==0 ∧ funcB_ret==0)	UNSAT (x>0∧ funcB_x==x ∧ funcB_x==0)
A-(3)B-(2)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret<0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==-1)	SAT
A-(3)B-(3)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret>=0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==funcC_ret)	SAT
A-(3)B-(4)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x<=10 ∧ funcC_ret<0 ∧ funcC_x==funcB_x-1 ∧ funcB_ret==-1)	SAT
A-(3)B-(5)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x<=10 ∧ funcC_ret>=0 ∧ funcC_x==funcB_x-1 ∧ funcB_ret==funcC_ret)	SAT

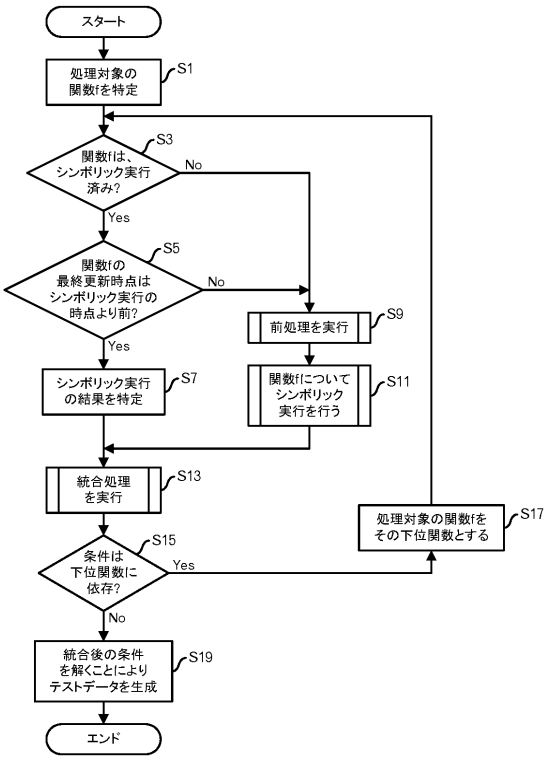
【図１３】

ID	条件	判定
A-(3)B-(2) C-(1)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret<0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==-1) ∧ (funcC_x==20 ∧ funcC_ret==1)	UNSAT (funcC_ret<0 ∧ funcC_ret==1)
A-(3)B-(2) C-(2)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret<0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==-1) ∧ (funcC_x!=20 ∧ funcC_ret==1)	SAT

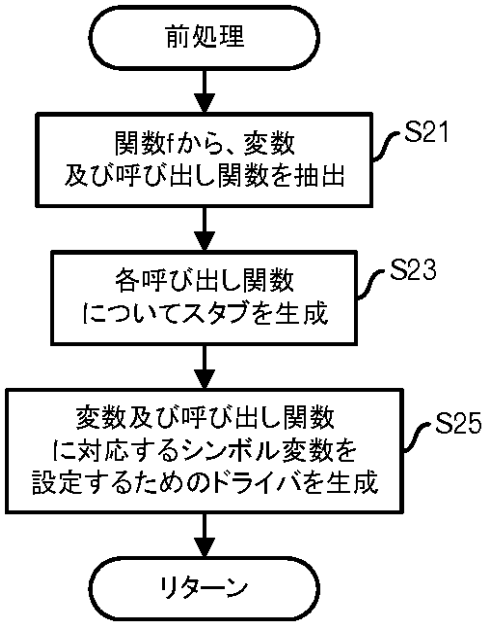
【図１４】

ID	条件	テストデータ
A-(1)	x<=0	x=0
A-(2)B-(3) C-(1)	(x>0 ∧ funcB_ret>0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret>=0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==funcC_ret) ∧ (funcC_x==20 ∧ funcC_ret==1)	x=19
A-(3)B-(2) C-(2)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret<0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==-1) ∧ (funcC_x!=20 ∧ funcC_ret==-1)	x=11
A-(3)B-(4) C-(2)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x<=10 ∧ funcC_ret<0 ∧ funcC_x==funcB_x-1 ∧ funcB_ret==-1) ∧ (funcC_x!=20 ∧ funcC_ret==-1)	x=10

【図１５】



【図１６】



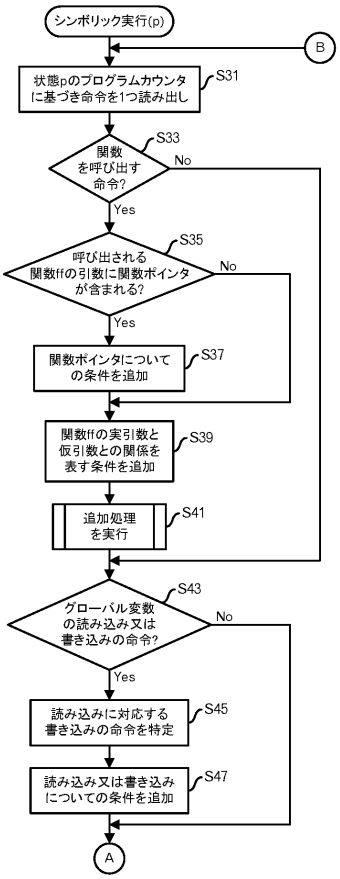
【図１７】

```
1: static int stub_funcB = 0;
2: void setStub_funcB(int symbol){
3:   stub_funcB = symbol;
4: }
5: int funcB(int x){
6:   return stub_funcB;
7: }
```

【図１８】

```
1: int main(){
2:   int x;
3:   klee_make_symbolic(&a,sizeof(a), "a" );
4:   klee_make_symbolic(&b,sizeof(b), "b" );
5:   setStub_funcB(b);
6:   funcA(a);
7:   return 0;
8: }
```

【図１９】



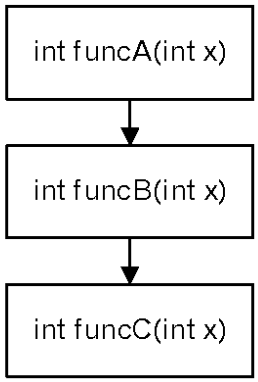
【図２０】

```
1: int funcA(int x){
2:   int y = 0;
3:   if(x > 0){
4:     y = funcB(x,funcC);
5:   }
6:   if(y > 0){
7:     return 1;
8:   }
9:   return 0;
10: }
```

```
1: int funcB(int x, int (*p)(int)){
2:   int y = 0;
3:   if(x == 0){
4:     return 0;
5:   }
6:   if(x > 10){
7:     y = p(x+1);
8:   } else {
9:     y = p(x-1);
10:  }
11:  if(y < 0){
12:    return -1;
13:  }
14:  return y;
15: }
```

```
1: int funcC(int x){
2:   if(x == 20){
3:     return 1;
4:   }
5:   return -1;
6: }
```

【図２１】



【図２２】

ID	条件	返却値
A-(1)	x<=0	0
A-(2)	x>0 ∧ funcB_ret>0 ∧ funcB_x==x ∧ funcB_p==funcC	1
A-(3)	x>0 ∧ funcB_ret<=0 ∧ funcB_x==x ∧ funcB_p==funcC	0

【図２３】

ID	条件	返却値
B-(1)	x==0	0
B-(2)	x!=0 ∧ x>10 ∧ *p_ret<0 ∧ *p_x==x+1	-1
B-(3)	x!=0 ∧ x>10 ∧ *p_ret>=0 ∧ *p_x==x+1	*p_ret
B-(4)	x!=0 ∧ x<=10 ∧ *p_ret<0 ∧ *p_x==x-1	-1
B-(5)	x!=0 ∧ x<=10 ∧ *p_ret>=0 ∧ *p_x==x-1	*p_ret

【図２４】

ID	条件	返却値
C-(1)	x==20	1
C-(2)	x!=20	-1

【図２５】

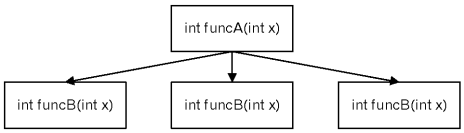
ID	条件
A-(1)	x<=0
A-(2)B-(3)	(x>0 ∧ funcB_ret>0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret>=0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==funcC_ret)
A-(2)B-(5)	(x>0 ∧ funcB_ret>0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x<=10 ∧ funcC_ret>=0 ∧ funcC_x==funcB_x-1 ∧ funcB_ret==funcC_ret)
A-(3)B-(2)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret<0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==-1)
A-(3)B-(3)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x>10 ∧ funcC_ret>=0 ∧ funcC_x==funcB_x+1 ∧ funcB_ret==funcC_ret)
A-(3)B-(4)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x<=10 ∧ funcC_ret<0 ∧ funcC_x==funcB_x-1 ∧ funcB_ret==-1)
A-(3)B-(5)	(x>0 ∧ funcB_ret<=0 ∧ funcB_x==x) ∧ (funcB_x!=0 ∧ funcB_x<=10 ∧ funcC_ret>=0 ∧ funcC_x==funcB_x-1 ∧ funcB_ret==funcC_ret)

【図２６】

```
1: int funcA(int x){
2:   int i, y = 0;
3:   if(x>0)
4:     for(i=0;i<3;i++)
5:       y += funcB(x+i);
6:   if(y > 0){
7:     return 1;
8:   }
9:   return 0;
10: }
```

```
1: int funcB(int x){
2:   if(x == 20){
3:     return 1;
4:   }
5:   return -1;
6: }
```

【図２７】



【図２８】

ID	条件	返却値
A-(1)	x<=0	0
A-(2)	x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret>0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2	1
A-(3)	x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2	0

【図２９】

ID	条件	返却値
B0-(1)	x==20	1
B0-(2)	x!=20	-1

ID	条件	返却値
B1-(1)	x==20	1
B1-(2)	x!=20	-1

ID	条件	返却値
B2-(1)	x==20	1
B2-(2)	x!=20	-1

【図３０】

ID	条件	判定
A-(2)B0-(1) B1-(1)B2-(1)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret>0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x==20 ∧ funcB_0_ret==1) ∧ (funcB_1_x==20 ∧ funcB_0_ret==1) ∧ (funcB_2_x==20 ∧ funcB_2_ret==1)	UNSAT
A-(2)B0-(2) B1-(1)B2-(1)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret>0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_1_x==20 ∧ funcB_0_ret==1) ∧ (funcB_2_x==20 ∧ funcB_2_ret==1)	UNSAT
A-(2)B0-(1) B1-(2)B2-(1)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret>0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x==20 ∧ funcB_0_ret==1) ∧ (funcB_1_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_2_x==20 ∧ funcB_2_ret==1)	UNSAT
A-(2)B0-(2) B1-(2)B2-(1)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret>0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_1_x==20 ∧ funcB_0_ret==1) ∧ (funcB_2_x==20 ∧ funcB_2_ret==1)	UNSAT
A-(2)B0-(1) B1-(1)B2-(2)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret>0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x==20 ∧ funcB_0_ret==1) ∧ (funcB_1_x==20 ∧ funcB_0_ret==1) ∧ (funcB_2_x!=20 ∧ funcB_2_ret==-1)	UNSAT
A-(2)B0-(2) B1-(1)B2-(2)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret>0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_1_x==20 ∧ funcB_0_ret==1) ∧ (funcB_2_x!=20 ∧ funcB_2_ret==-1)	UNSAT
A-(2)B0-(1) B1-(2)B2-(2)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret>0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x==20 ∧ funcB_0_ret==1) ∧ (funcB_1_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_2_x!=20 ∧ funcB_2_ret==-1)	UNSAT
A-(2)B0-(2) B1-(2)B2-(2)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret>0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_1_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_2_x!=20 ∧ funcB_2_ret==-1)	UNSAT

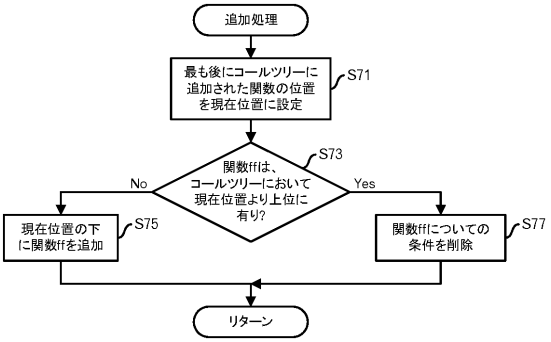
【図３１】

ID	条件	判定
A-(3)B0-(1) B1-(1)B2-(1)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x==20 ∧ funcB_0_ret==1) ∧ (funcB_1_x==20 ∧ funcB_0_ret==1) ∧ (funcB_2_x==20 ∧ funcB_2_ret==1)	UNSAT
A-(3)B0-(2) B1-(1)B2-(1)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_1_x==20 ∧ funcB_0_ret==1) ∧ (funcB_2_x==20 ∧ funcB_2_ret==1)	UNSAT
A-(3)B0-(1) B1-(2)B2-(1)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x==20 ∧ funcB_0_ret==1) ∧ (funcB_1_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_2_x==20 ∧ funcB_2_ret==1)	UNSAT
A-(3)B0-(2) B1-(2)B2-(1)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x==20 ∧ funcB_0_ret==1) ∧ (funcB_1_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_2_x!=20 ∧ funcB_2_ret==-1)	SAT
A-(3)B0-(1) B1-(1)B2-(2)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x==20 ∧ funcB_0_ret==1) ∧ (funcB_1_x==20 ∧ funcB_0_ret==1) ∧ (funcB_2_x!=20 ∧ funcB_2_ret==-1)	UNSAT
A-(3)B0-(2) B1-(1)B2-(2)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x==20 ∧ funcB_0_ret==1) ∧ (funcB_1_x==20 ∧ funcB_0_ret==-1) ∧ (funcB_2_x!=20 ∧ funcB_2_ret==-1)	SAT
A-(3)B0-(1) B1-(2)B2-(2)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x==20 ∧ funcB_0_ret==1) ∧ (funcB_1_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_2_x!=20 ∧ funcB_2_ret==-1)	SAT
A-(3)B0-(2) B1-(2)B2-(2)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_1_x!=20 ∧ funcB_0_ret==-1) ∧ (funcB_2_x!=20 ∧ funcB_2_ret==-1)	SAT

【図３２】

ID	条件	テストデータ
A−(1)	x<=0	x=0
A−(3)B0−(2) B1−(2)B2−(1)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x!=20 ∧ funcB_0_ret==−1) ∧ (funcB_1_x!=20 ∧ funcB_0_ret==−1) ∧ (funcB_2_x==20 ∧ funcB_2_ret==1)	x=18
A−(3)B0−(2) B1−(1)B2−(2)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x!=20 ∧ funcB_0_ret==−1) ∧ (funcB_1_x==20 ∧ funcB_0_ret==1) ∧ (funcB_2_x!=20 ∧ funcB_2_ret==−1)	x=19
A−(3)B0−(1) B1−(2)B2−(2)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x==20 ∧ funcB_0_ret==1) ∧ (funcB_1_x!=20 ∧ funcB_0_ret==−1) ∧ (funcB_2_x!=20 ∧ funcB_2_ret==−1)	x=20
A−(3)B0−(2) B1−(2)B2−(2)	(x>0 ∧ funcB_0_ret+funcB_1_ret+funcB_2_ret<=0 ∧ funcB_0_x==x ∧ funcB_1_x==x+1 ∧ funcB_2_x==x+2) ∧ (funcB_0_x!=20 ∧ funcB_0_ret==−1) ∧ (funcB_1_x!=20 ∧ funcB_0_ret==−1) ∧ (funcB_2_x!=20 ∧ funcB_2_ret==−1)	x=1

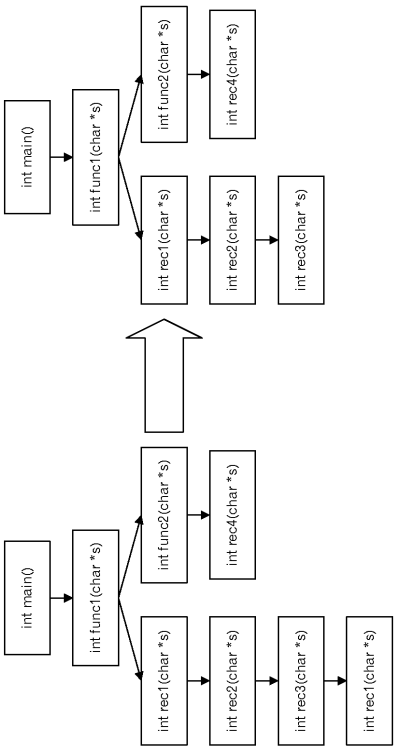
【図３３】



【図３４】

```
1: int rec3(int x){
2:   int y = 0;
3:   if(x > 0){
4:     y = rec1(x);
5:   }
6:   if(y > 0){
7:     return 1;
8:   }
9:   return 0;
10: }
```

【図３５】

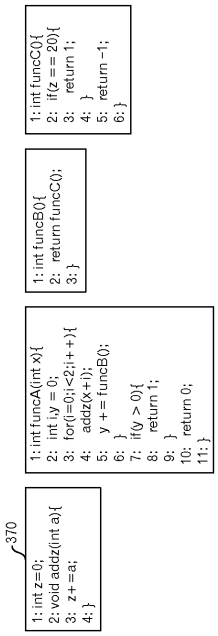


【図３６】

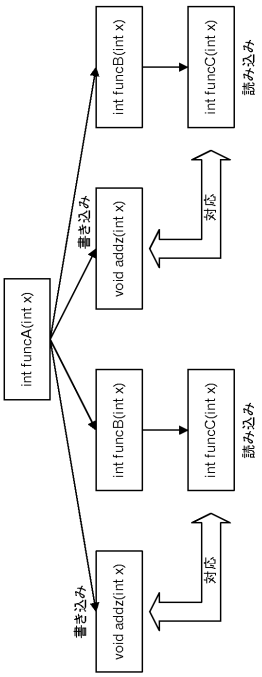
ID	条件	返却値
A−(1)	x<=0	0
A−(2)	x>0 ∧ rec1_ret>0 ∧ rec1_x==x	1
A−(3)	x>0 ∧ rec1_ret<=0 ∧ rec1_x==x	0

ID	条件	返却値
A−(1)	x<=0	0
A−(2)	x>0	1
A−(3)	x>0	0

【図３７】



【図３８】



【図３９】

ID	条件	返却値
Z0-(1)	z==a	

ID	条件	返却値
Z1-(1)	z==a	

【図４０】

ID	条件	返却値
A-(1)	addz_0_a==x ∧ addz_1_a==x+1 ∧ funcB_0_ret+funcB_1_ret>0	1
A-(2)	addz_0_a==x ∧ addz_1_a==x+1 ∧ funcB_0_ret+funcB_1_ret<=0	0

【図４１】

ID	条件	返却値
B0-(1)		funcC_ret

ID	条件	返却値
B1-(1)		funcC_ret

【図４２】

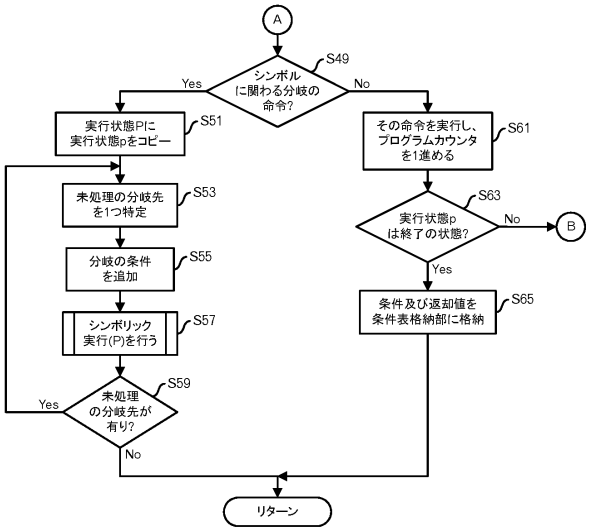
ID	条件	返却値
C0-(1)	z==20	1
C0-(2)	z!=20	-1

ID	条件	返却値
C1-(1)	z==20	1
C1-(2)	z!=20	-1

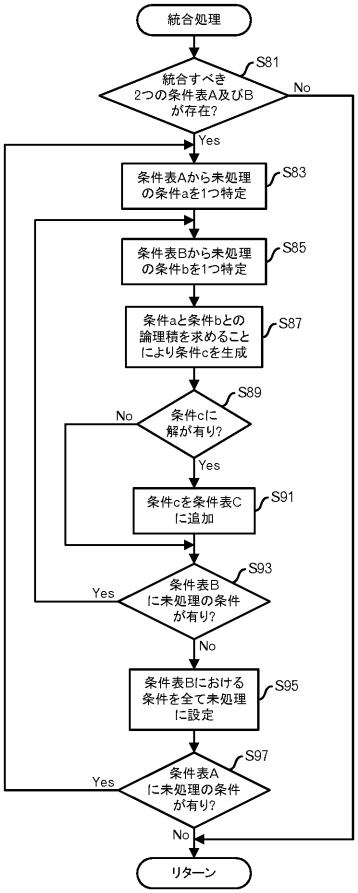
【図４３】

ID	条件	判定
A-(1)B0-(1)C0-(1) Z0-(1)B1-(1)C1-(1) Z1-(1)	(addz_0_a==x ∧ addz_1_a==x+1 ∧ funcB_0_ret+funcB_1_ret>0) ∧ (funcB_0_ret==funcC_0_ret) ∧ (z_0==20 ∧ funcC_0_ret==1) ∧ (z_0==addz_0_a) ∧ (funcB_1_ret==funcC_1_ret) ∧ (z_1==20 ∧ funcC_1_ret==1) ∧ (z_1==addz_1_a)	SAT
A-(1)B0-(1)C0-(2) Z0-(1)B1-(1)C1-(1) Z1-(1)	(addz_0_a==x ∧ addz_1_a==x+1 ∧ funcB_0_ret+funcB_1_ret>0) ∧ (funcB_0_ret==funcC_0_ret) ∧ (z_0!=20 ∧ funcC_0_ret==-1) ∧ (z_0==addz_0_a) ∧ (funcB_1_ret==funcC_1_ret) ∧ (z_1==20 ∧ funcC_1_ret==1) ∧ (z_1==addz_1_a)	UNSAT
A-(1)B0-(1)C0-(1) Z0-(1)B1-(1)C1-(2) Z1-(1)	(addz_0_a==x ∧ addz_1_a==x+1 ∧ funcB_0_ret+funcB_1_ret>0) ∧ (funcB_0_ret==funcC_0_ret) ∧ (z_0==20 ∧ funcC_0_ret==1) ∧ (z_0==addz_0_a) ∧ (funcB_1_ret==funcC_1_ret) ∧ (z_1!=20 ∧ funcC_1_ret==-1) ∧ (z_1==addz_1_a)	UNSAT
A-(1)B0-(1)C0-(2) Z0-(1)B1-(1)C1-(2) Z1-(1)	(addz_0_a==x ∧ addz_1_a==x+1 ∧ funcB_0_ret+funcB_1_ret>0) ∧ (funcB_0_ret==funcC_0_ret) ∧ (z_0!=20 ∧ funcC_0_ret==-1) ∧ (z_0==addz_0_a) ∧ (funcB_1_ret==funcC_1_ret) ∧ (z_1!=20 ∧ funcC_1_ret==-1) ∧ (z_1==addz_1_a)	UNSAT
A-(2)B0-(1)C0-(1) Z0-(1)B1-(1)C1-(1) Z1-(1)	(addz_0_a==x ∧ addz_1_a==x+1 ∧ funcB_0_ret+funcB_1_ret<=0) ∧ (funcB_0_ret==funcC_0_ret) ∧ (z_0==20 ∧ funcC_0_ret==1) ∧ (z_0==addz_0_a) ∧ (funcB_1_ret==funcC_1_ret) ∧ (z_1==20 ∧ funcC_1_ret==1) ∧ (z_1==addz_1_a)	UNSAT
A-(2)B0-(1)C0-(2) Z0-(1)B1-(1)C1-(1) Z1-(1)	(addz_0_a==x ∧ addz_1_a==x+1 ∧ funcB_0_ret+funcB_1_ret<=0) ∧ (funcB_0_ret==funcC_0_ret) ∧ (z_0!=20 ∧ funcC_0_ret==-1) ∧ (z_0==addz_0_a) ∧ (funcB_1_ret==funcC_1_ret) ∧ (z_1==20 ∧ funcC_1_ret==1) ∧ (z_1==addz_1_a)	SAT
A-(2)B0-(1)C0-(1) Z0-(1)B1-(1)C1-(2) Z1-(1)	(addz_0_a==x ∧ addz_1_a==x+1 ∧ funcB_0_ret+funcB_1_ret<=0) ∧ (funcB_0_ret==funcC_0_ret) ∧ (z_0==20 ∧ funcC_0_ret==1) ∧ (z_0==addz_0_a) ∧ (funcB_1_ret==funcC_1_ret) ∧ (z_1!=20 ∧ funcC_1_ret==-1) ∧ (z_1==addz_1_a)	SAT
A-(2)B0-(1)C0-(2) Z0-(1)B1-(1)C1-(2) Z1-(1)	(addz_0_a==x ∧ addz_1_a==x+1 ∧ funcB_0_ret+funcB_1_ret<=0) ∧ (funcB_0_ret==funcC_0_ret) ∧ (z_0!=20 ∧ funcC_0_ret==-1) ∧ (z_0==addz_0_a) ∧ (funcB_1_ret==funcC_1_ret) ∧ (z_1!=20 ∧ funcC_1_ret==-1) ∧ (z_1==addz_1_a)	SAT

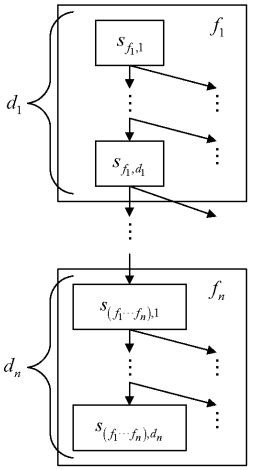
【図４４】



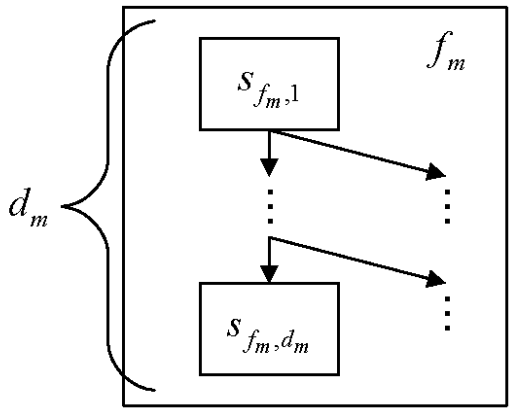
【図４５】



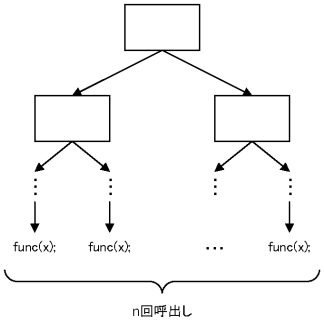
【図４６】



【図４７】



【図４８】



【図４９】

```
for(i=0;i<n;i++){
  func(x+i);
}
```

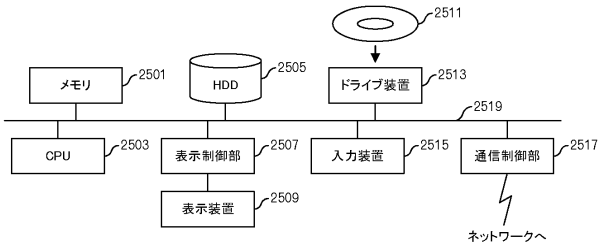
【図５０】

```
1: int funcC(int x){
2:   if(x == 20){
3:     return 1;
4:   }else if(x == 30){
5:     return 2;
6:   }
7:   return -1;
8: }
```

【図５１】

ID	条件	返却値
C-(1)	x==20	1
C-(2)	x!=20 ∧ x==30	2
C-(3)	x!=20 ∧ x!=30	-1

【図５２】



フロントページの続き

(72)発明者　藤原　翔一朗
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
(72)発明者　モンプラターンチャイ　スッパシット
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
(72)発明者　前田　芳晴
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
(72)発明者　片山　朝子
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

審査官　多賀　実

(56)参考文献　国際公開第２０１３／１６１１９５（WO，Ａ１）
橋本　祐介　外１名，「有界モデル検査法を用いたモジュラー検証のテストケース生成による補完」，電子情報通信学会技術研究報告，社団法人電子情報通信学会，２０１１年　2月２８日，第１１０巻，第４５８号，ｐｐ．９１－９６
橋本　祐介　外１名，「有界モデル検査法を用いたＣプログラムのモジュラー検証」，情報処理学会論文誌　論文誌ジャーナル［ＣＤ－ＲＯＭ］，一般社団法人情報処理学会，２０１１年　８月１５日，第５２巻，第８号，ｐｐ．２４２２－２４３０

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ１１／３６