受領書

平成３０年　２月　８日
特　許　庁　長　官

識別番号　　１００１１０５５９
氏名（名称）　　友野　英三　　様

以下の書類を受領しました。

項番	書類名	整理番号	受付番号	提出日	出願番号通知（事件の表示）	アクセスコード
1	特許願	DCT18017	51800263743	平30. 2. 8	特願2018- 21459	EE79
2	特許願	DCT18021	51800263744	平30. 2. 8	特願2018- 21460	EE7A

以　上

【書類名】　特許願
【整理番号】　DCT18021
【提出日】　平成30年 2月 8日
【あて先】　特許庁長官殿
【国際特許分類】　G06F 21/56
【発明者】
　【住所又は居所】　神奈川県鎌倉市山ノ内５９１番地
　【氏名】　根来　文生
【特許出願人】
　【識別番号】　598037422
　【氏名又は名称】　根来　文生
【代理人】
　【識別番号】　100110559
　【弁理士】
　【氏名又は名称】　友野　英三
　【電話番号】　0422-27-7774
【提出物件の目録】
　【物件名】　明細書　1
　【物件名】　特許請求の範囲　1
　【物件名】　要約書　1
　【物件名】　図面　1

【書類名】明細書
【発明の名称】テスト対象プログラムを実行させず、且つテストデータを使用せずにそのプログラムに内在するバグの起因と起因から発症する全バグをそのコンパイル済みのソースから自動抽出する方法。
【技術分野】
【０００１】
プログラムのコンパイル済みのソースプログラムに内在するバグを発症させる起因を、テストデータを用いず且つ実行させずに、そのソースから自動的に抽出するプログラムに関する。
本願発明で言う「バグの起因」は、プログラムの内容そのものが要件と元々異なることが原因で、意図と異なる結果となる原因は除いている。
本願発明で言う「バグの起因」とは、プログラムの作り方に問題があることにより生じる様相である。要件そのものの間違いによる、或いは要件を間違ったプログラム化によるバグの起因は本願発明のバグには含めない。

【背景技術】
【０００２】
本願に関連する背景技術は、本発明者によるもの以外には見当たらない。
本願並びに特許文献１、２の背景に置かれる理論は、本願発明の基本となっている本発明者によるプログラムの同期構造に関する研究の成果が用いられているという理由で、特許文献１、２をここに掲げる。但し、特許文献１、２の「課題とそれを解決する手段」と、本願発明の「課題とそれを解決する手段」は異なる。同期とは段落番号０００７参照。
特許文献３は、今回出願の発明の背景となる理論は同じであるし、今回出願に係わるLYEE空間も主語系譜も登場する。しかし、特許文献３は命令文に属する変数を主語と変数主語に区分する必要がありそのため構文解析が複雑となるので実用上は困難があった。主語とは、その値を生成される変数（例えばA=B+CのA）、変数主語とは主語でない変数（BやC。X>YのXやYも変数主語）である。それに対し今回出願の発明は、これら称呼区別の必要をなくした。これは実用上大きな技術的進歩である。これに応じて今回出願の発明のLYEE空間や主語系譜の内容は特許文献３のそれらとは異なる。また今回出願の発明は「変数の値を生成するタイミング違反によるバグの起因の発見方法」であるが、特許文献３にはその「課題」も「それを解決する手段」も含まれていない。
非特許文献１と本出願の発明の違いは特許文献３と同じである。

【先行技術文献】
【特許文献】
【０００３】
【特許文献１】特許５９９２０７９
【特許文献２】特許６０８６９７７
【特許文献３】特開２０１３－０５８１２７
【非特許文献】
【０００４】
【非特許文献１】根来文生著　コンピュータウィルスを無力化するプログラム革命。２０１４年１１月３日　日本地域社会研究所発行。
【発明の概要】
【発明が解決しようとする課題】
【０００５】
本明細書には本発明に特有な語が用いられている。その索引は段落番号００４２にあるので必要があれば参照されたい。
目次

２．プログラムで使用する名詞数の臨界数

【０００６】
１．プログラムテストの課題
自然法則を利用した物性的存在物であるコンピュータハードウェアと、プログラムが共生するシステムでは、共生機能をテスト用データを用いてプログラム検証が行われる。しかし、この半世紀に及ぶＩＴ分野のプログラム検証の結果、システムの完成度、プログラム保守作業の実情を見れば、このような検証方法ではプログラムに内在する課題（バグの起因）が完全に捉えられていないことは明かである。（「バグ」と「バグの起因」の定義については段落番号００１０参照。）そして、プログラム規模が大きくなるほど、この問題はソフト危機として深刻な問題に発展している。これは１９７０年代に始まる問題で、今日に至るも何も解法されていない状態にある。合わせて、ここ３０年近くは、この問題を直視できる関係者が育っていないことは、この分野の看過できない深刻な問題である。

【０００７】
２．プログラムで使用する名詞数の臨界数
プログラムとは、言い換えると、そのプログラムに属す全名詞（変数のことを本願では名詞と呼ぶ）、同全定数（「定値」と呼ぶ）、同全文字列（「定置」と呼ぶ）が総合的に織りなす脈絡（脈絡の意味については段落番号００１５参照。）を成立させることである。そのプログラムにおいて、全名詞数が或る値を超えるとその脈絡には曖昧さが生じる。これは、本発明人の４０年に及ぶプログラム構造の研究過程で捉えられた現象であるが、概略的にはプログラムで扱う名詞数が２００を超えると、ひとがプログラムの意図を律することが困難になるという現象である。これは観点を変えると、われわれの思考法の同期的限界を意味していた。同期とは、システム内の全ての名詞の値の関係が、矛盾のない依存関係（親子関係）になっている状態を言う。同期するプログラムを「正統なプログラム」という。正統なプログラムについては段落番号００１０参照。
【０００８】
本発明者の研究では、名詞のこの同期的限界数を「名詞数の臨界数」と称した。即ち、名詞の数が臨界数を超えるプログラムでは、正統なプログラムでない可能性があるということである。換言すれば、関係者達の誰もが自覚できないバグを生じさせる原因になっているという事である。このバグの起因はテストデータを用いて実行するプログラムテストでは発見できないのである。なぜならプログラムは実行できるし、結果の数値が想定外の突飛なものでない限り異常を感じないので発見する方法が無いのである。後述する「主語系譜」は、このバグの起因を捉えることが出来る様に作られた全名詞の同期脈絡構造の事である。

【課題を解決するための手段】
【０００９】
目次　段落番号

【００１０】
１．　プログラムテストの課題を解決するための本発明の概要
完全性を持つプログラムを本研究では「正統なプログラム」と呼ぶ。正統なプログラムは「主語系譜」で観察することが出来る。
主語系譜とはプログラムに存在する全名詞、全定値、全定置を本研究でいう後述する「調和脈絡」、「超言語脈絡」と呼ぶ２種の規則で脈絡化したものである。
「完全性を持つプログラム」即ち「正統なプログラム」とは、主語系譜に於ける２種の規則による脈絡が、「バグの起因」を含まずに全て成立されている状態のプログラムのことである。
「バグの起因を含まないプログラム」とは、全名詞が同期している、即ち全名詞の値の関係が矛盾のない依存関係（親子関係）になっている状態であり、たとえば、名詞aの値が生成されるタイミングtにおいて関与する他の名詞bの値が有効・適切であるプログラムである。つまりタイミングtにおいては使われる名詞bの成立条件が既に変わっているのに、変わる前のタイミングt-nの成立条件下のbの値を使っているようなことが無いプログラムである。

名詞の数が臨界点を超えると、バグの起因が含まれてくることが避けられない。結果、実行時エラーを発症する。それを避けるためには、名詞の値を生成するタイミングは一つに限ればよい。即ち名詞の値を生成する名詞の定義文は、１つの名詞に対して１つに限られることが望ましく、そうであればタイミング違反による実行時エラーは発症しない。（「タイミング違反」については段落番号００３０乃至００３３参照。）このように名詞の定義文が単一であるプログラムを完全性を持つプログラムと言う。逆に、同名の名詞の定義文が複数あることが「バグの起因」である。
「プログラムのバグ」とは、主語系譜に於ける調和脈絡と超言語脈絡に「バグの起因」が含まれるとその箇所を起因として発症する症状である。バグとは症状であり、バグの起因とはその症状を引き起こす原因である。
【００１１】
ＬＹＥＥ空間は、テスト対象とするコンパイル済みプログラムのソースからコンピュータが構文解析して自動的に、またはそのプログラムの言語文法を修得済みのひとであれば導き出すことが出来る。ＬＹＥＥ空間ではプログラムソースの実相がＬＹＥＥ空間の構成項目に於いて捉えられる。そして、主語系譜ではＬＹＥＥ空間が捉えるプログラムソースの実相が、２種の規則（調和脈絡と超言語脈絡）で図式化され可視化されることにより、プログラムに属す名詞達の同期的様相が捉えられる。故に、プログラムソースが完全であれば、主語系譜の全脈絡はバグの起因を含まずに自ずと完結し、プログラムソースが完全でなければ、主語系譜を成立させる脈絡にはバグの起因となる名詞が現れる。これがバグを

発症させる起因である。そして、この起因を基に主語系譜の脈絡を辿れば、この起因の影響を受ける名詞が判る。それら名詞がバグを発症させることになる。
因みにLYEE空間はシナリオ関数の研究過程で求められた。シナリオ関数については、特許文献１及び特許文献２を参照。それは命令処理の順序を人が決める必要が無く、且つ全名詞の値の同期を保証するプログラム手法である。

【００１２】

ＬＹＥＥ空間には、テストされるプログラムの記述言語の文法規則に従って、テストされるプログラムが構文解析によって図１－１乃至図１－４の表の項目に展開される。結果、ＬＹＥＥ空間では文法規則に反するプログラムが論理バグとして捉えられる。文法規則に反するとは、例えば、領域定義文が存在しない名詞が使われているソースである。
次に、このＬＹＥＥ空間を基に二つの規則（調和脈絡、超言語脈絡）で導かれる主語系譜では、プログラムに内在する上記論理バグ以外のバグの起因が捉えられる。

【００１３】

２．名詞数の臨界数
本研究によれば、プログラムに属す名詞（記憶領域の番地）の数が、われわれの統治能力の臨界数を超えると、われわれはそれら名詞を完全に統治するプログラムを作ることが困難になる。このことがプログラムに不完全性を生じさせる原因になっている。換言すれば、これがプログラムにバグを誘発させている真因である。
名詞の臨界数は、本発明者の４０年間余に及ぶプログラム構造論の研究期間の中で、継続的に行われた数百本に及ぶ様々な分野の従来プログラムの精密な分析調査で発見された現象である。因みに、ＬＹＥＥ空間及び主語系譜の構造理論は、この分析調査のツールとして用いられたものである。これらは、従来プログラムの未解決の３題問題、即ち、プログラムバグ問題、ウイルス問題、デッドロック問題の解法を自律的に成立させるプログラム構造（シナリオ関数）と共に、２００８年に確立されている。

【００１４】

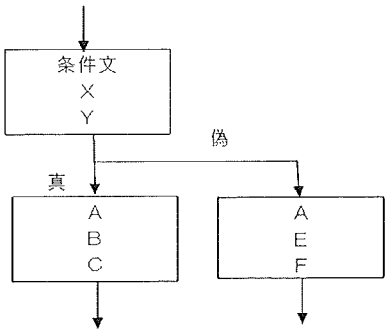
名詞の臨界数はシステムの規模で様々に変わるが、目安として、名詞数が２００個を超えるとプログラム達に生じる不完全性の原因になる。その捉え方については後述の段落番号００２９乃至００３５参照。プログラムの完全性の理解がないまま、即ち、不完全なプログラムの全てのバグの起因を、これまでのテストデータを使用するテスト方法で抽出することは、原理的に成立はしない。故に、おぼろげに、従来プログラムに残存バグの起因が内在したまま稼働していることに気づく人がいるとしても、実際は棚上げされたままで完成とされている。これがＩＴ分野の嘆かわしい現実なのである。

【００１５】

３．調和脈絡、超言語脈絡、主語系譜
プログラムの脈絡には調和脈絡と超言語脈絡がある。調和脈絡とは、処理の流れ、即ち構文の処理の順序に沿って登場する個々の名詞、定値、定置文たちの関係である。例えば、ソースプログラムの中に下記の構文

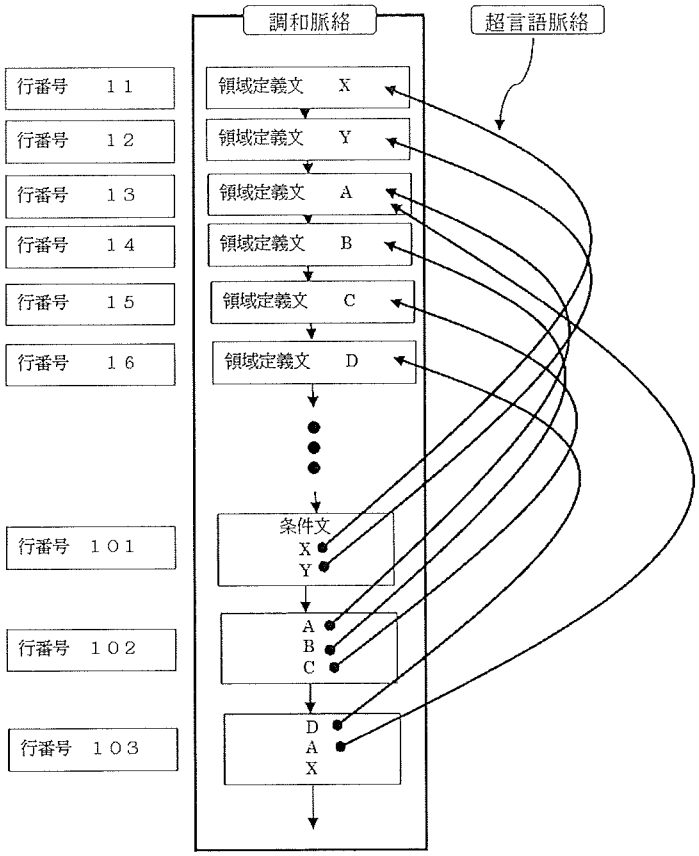
```
if X>Y
A=B+C
else
A=E+F
```

があったとする。この構文は、名詞X 及び名詞Yが係る条件下に置いて、真の場合に名詞A、B、Cの間に、また偽の場合に名詞A、E、Fの間に、特定の関係が成立していることを意味している。これを図示すると下図のようになる。これらの名詞達との関係を示す矢印線が脈絡である。条件文がない命令文の場合は単にその構文に含まれる名詞、定値、定置を含むブロックの前後に引かれる矢印線が脈絡となる。
脈絡図。



この脈絡を構文の流れに沿って書き連ねた構造体が調和脈絡である。以下の図を参照されたい。この以下の図には条件偽が省略されている。

主語系譜図



【００１６】

一方、超言語脈絡とは、同じ名詞の同期関係を捉える関係である。超言語脈絡とは同名の名詞を結んで関係付ける矢印線のことある。その名詞とは記憶領域の番地名を指している。記憶領域の番地名は領域定義文で定義されるので、超言語脈絡とは各名詞とその領域定義文を結んで関係付ける矢印線のことである。これを図示すると上図のようになる。太線が超言語脈絡である。上図には上記構文if X>Y　A=B+Cの直後に構文　D＝A＋Xが加えてある。

調和脈絡と超言語脈絡を合わせて主語系譜と言う。
主語系譜において調和脈絡と超言語脈絡及び構文を観察すると、
「各名詞がいつ、どこで、どのような名詞が係る条件下で、どのような名詞が係って、生成されたのか」
が一目で分かる。主語系譜によってプログラムの意味の可視化が出来るのである。
【００１７】
例えば、行番号１０３における名詞Dと関係のある名詞Aは、その超言語脈絡を辿りAの領域定義文を見ると、そこに調和脈絡の上で上流にある行番号１０２からの超言語脈絡が到達している。その超言語脈絡をたどると、行番号１０１において名詞Xと名詞Yが係っている条件下で、行番号１０２において名詞Bと名詞Cによって値が生成される名詞Aであることが分かる。つまりＢ、Ｃ、Ｘ、Ｙが親でありＡが子である。このように主語系譜によって全ての名詞の親子の関係が明らかにされ、それら全ての名詞が連鎖して全体の系譜を形成する。そこでは全ての個々の名詞がどのような名詞が係って生成されるかが可視化できる。プログラムの可視化によって、各名詞（変数）が個々の成立要件に登場する名詞と合致し、且つ全ての名詞が正しい依存関係で連鎖して生成されているかどうかが分かるのである。
調和脈絡と超言語脈絡が完全に形成することが出来る状態を、「全名詞が同期している状態」という。完全でない脈絡がバグの起因であるが、それを捉える方法については、後述の５．２（主語系譜においてバグの起因と実行エラーの発見方法）（段落番号００２９）で述べる。
【００１８】
なお、調和脈絡と名付けた理由は、調和とはハーモニーであり全体と言う意味を持つので、名詞達の夫々の関係である脈絡の全てを含む全体の脈絡と言う含蓄である。一方、超言語脈絡と名付けた理由は、人間の思考方法は言語であるが、それは一筆書きのように常に一筋の論理を辿る。言語思考は構造的に以前の思考と常に照合しているわけではない。超言語脈絡とは、前記のように各名詞とその領域定義文の関係付けであるが、それは各名詞の出生を領域定義文を介して常に構造的に関係付けて照合していることである。だから言語（人の思考）を超えると言う含蓄で超言語脈絡と名付けた。

【００１９】
４．　ＬＹＥＥ空間
LYEE空間とは、ソースコードの各構文を単一機能しかもたない単元文に分解し、夫々の単元文に、ソースコードの行番号又はその枝番号を付けてそれを座標とし、それら単元文毎に、次に実行される単元文の座標、単元文が条件文の場合は、真偽夫々の行き先にある単元文の座標、条件文を終了させる単元文の座標とその次に実行される単元文の座標、の諸座標を付し、更に単元文に含まれる名詞、定値、定置を抽出し、その名詞にはその領域定義文の座標を付し、これらを一覧表に展開したものである。詳細は下記４．１（ＬＹＥＥ空間の構成項目）参照。
LYEE空間には、更に別の項目を付加しても良い。例えば構文解析して各名詞を主語（値を生成される変数）と変数主語（主語でない変数）に区別しその欄を設けて表示しても良い。

【００２０】
４．１　LYEE空間の構成項目
図１－１乃至図１－４参照。そのソースプログラムは図１－５参照。
ＬＹＥＥ空間の項目は次の通りである。
（１）TCX。自然数で与えられる構文間の実行順位である。調和座標と総称される座標であり行番号又はその枝番号。
（２）プログラムの構文のソースコード
（３）ソースコード構文解析情報

１）ソースプログラムをその言語規則で構文解析して得られる単元化された構文の要素
２）構文の要素のTCX。構文が複数の単元文に分解される時は枝番号を付す。
(a) 構文の要素が名詞の場合は、そのＴＣＸと記憶領域(領域定義文)の番地（TCX）
(b)構文の要素が定値の場合は、そのTCX
(c) 構文の要素が定置（文字列）の場合は、そのTCX
３）構文の要素の種別。構文はその代表要素で文種化される。
文種には以下の１０種がある。注釈文、翻訳文、領域文（名詞文）、算術文、制御文、入力文、出力文、条件文、定値文、定置文（文字列文）がある。
（４）構文または構文要素の調和座標
調和座標はＴＣＸ、ＴＣＹ、ＴＣＺ１、ＴＣＺ２，ＴＣＺ３，ＴＣＺ４の６種で構成される。
１）ＴＣＸは構文に付与される。自然数で与えられる構文間の実行順位である。
２）ＴＣＹは当該構文の次に実行される構文のＴＣＸを指す。
３）ＴＣＺ１は当該構文が条件文なら、その条件文が真で実行されるＴＣＸを指す。
４）ＴＣＺ２は当該構文が条件文なら、その条件文が偽で実行されるＴＣＸを指す。
５）ＴＣＺ３は当該構文が条件文なら、その条件文の終了として実行される構文のＴＣＸを指す。
６）ＴＣＺ４は当該構文が条件文なら、その条件文の終了の構文の次に実行される構文のＴＣＸを指す。
因みに、上記の５）及び６）を設けるのは、どんな言語でもそのＬＹＥＥ空間と主語系譜を作成することができるようにするためである。

【００２１】
４．２　LYEE空間の作成方法
図１－１乃至図１－４参照。
ＬＹＥＥ空間は対象とするソースプログラムの構文を解析し、前記４．１（ＬＹＥＥ空間の構成項目）の各項目を抽出してＬＹＥＥ空間表を作成する。人が解析して抽出作成しても、自動的に行うアルゴリズムを用いて自動的に作成しても良い。

【００２２】
４．３　LYEE空間の超言語座標の付与方法
構文解析により求められる「構文を構成する要素」に付与されるTCXが超言語座標である。即ち、ＬＹＥＥ空間の項目「構文の要素のTCX欄」のことである。超言語座標は調和座標ＴＣＸの転用である。成立する超言語座標を以下に示す。即ち、
（１）領域定義文の名詞の超言語座標は、その領域定義文のＴＣＸである。
（２）領域定義文以外の名詞の超言語座標は、その名詞が属す構文のＴＣＸ及びその名詞の領域定義文のＴＣＸである。
（３）定値の超言語座標は、その定値が属す構文のＴＣＸである。
（４）定置の超言語座標は、その定置が属す構文のＴＣＸである。
（５）分岐先を示すラベルは、定置として扱われる。

【００２３】
４．４　LYEE空間でバグの起因を捉える方法
テストされるプログラムは、構文解析によって図１－１乃至図１－４のLYEE空間の項目に展開される。各項目の値が得られないと、それが文法規則に反するバグの起因として捉えられる。文法規則に反するとは、例えば、領域定義文が存在しない名詞が使われているソースである。LYEE空間において各項目が成立させられない個所があるとプログラムは不完全となる。その状況は、名詞の数が臨界数を超えるプログラムや、プログラムの製造を分担して行いそれらを合体するプログラムで起こり得る。

ＬＹＥＥ空間は、もし、どのプログラム言語のプログラムでもコンパイルできる万能コンパイラが存在するとすれば、そのコンパイラが生成する翻訳テーブルとして位置づけられる。この概念は本発明者がプログラムの完全系（シナリオ関数）を求める研究の中で、形而上学的論考を起源として導出されたものである。

【００２４】
５．　主語系譜
５．１　主語系譜の作成方法
前記３．（調和脈絡、超言語脈絡、主語系譜。段落番号００１５，００１６）及び図２－１、図２－２参照。
主語系譜とは、システムが統治する全名詞、全定値、全定置文（定置文とは領域を持たない文字列）の同期的成立関係を捉える脈絡図のことである。即ち、主語系譜では、先ず調和脈絡が、外部情報を取得する領域の名詞を始祖（終点）とし、プログラムの外部に送り出される情報を起点として、起点から終点までを結ぶ間に連鎖して位置する矢印線と名詞達で成立する。
調和脈絡の作成方法は、ＬＹＥＥ空間である図１－１乃至図１－４から各構文毎に含まれる名詞、定値、定置を夫々のブロックとし、そのブロックを各構文のTCX、TCY、TCZ1、TCZ２、TCZ３、TCZ４に従って矢印線で結ぶ。
次に、各名詞を超言語脈絡で結ぶ（段落番号００１６）。
主語系譜はこれらの二つの操作で求められる。これら二つの操作は人が行っても良いし、コンピュータで自動的に行っても良い。

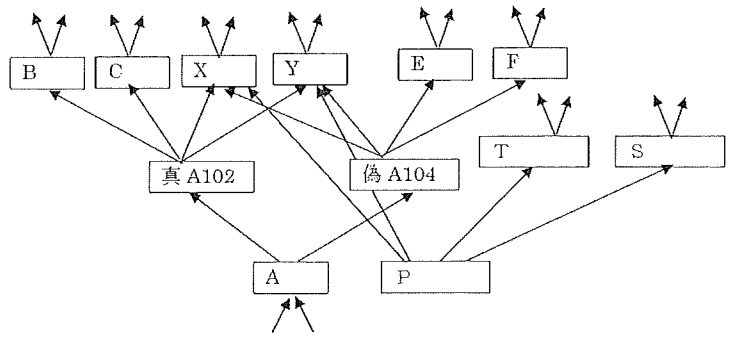
【００２５】
上記の主語系譜は、実は主語系譜の原形である。主語系譜の最終形は、主語系譜における調和脈絡と超言語脈絡を合体させたものである。その方法は、調和脈絡に登場する名詞、定値、定置達を、同名の領域定義文の名詞を介さずに、親子間で直接結び付けるものである。即ち、前記の主語系譜の原形において、各構文に対応する単元文を構文解析して、その中に含まれる名詞を「値を生成される変数」である主語（子）と、主語の「値の生成に関与する変数」である変数主語（親）に区分けして、これらの個々の間に存在する親子関係を、始祖である全名詞、定値、定置から子孫の全名詞を含む家系図である名詞の全系譜を表示するものを主語系譜の最終形（以後単に主語系譜という）という。個々の親子関係は前記段落番号００１５に示した脈絡である。

【００２６】
なお、主語系譜の作成に当たって特に注意する点を述べる。主語系譜はソースプログラムに登場する名詞、定値、定置達の値が、相互にどのような依存関係にあるかを観察し分析することを目的に、プログラムを可視化したものである。従って登場するそれらのエレメントの値に影響のない命令や操作は主語系譜には入らない。例えば、入出力命令、ファイルの開閉命令、真偽の区別や指標（true、false）等の動作は上記のエレメントの値に影響しないので主語系譜には入れない。主語系譜は、プログラムの構造の可視化ではなく、あくまでプログラムに登場する名詞、定値、定置達のみに着目し、それらが起点から終点までどのように相互依存しているかを可視化するものである。
勿論、上記の入出力命令、ファイルの開閉命令、真偽の区別や指標（true、false）等の動作をそれらの動作命令を特別名詞化して主語系譜に含めることもできる。そうすればプログラム構造も可視化することが出来る。しかしプログラムに登場する名詞以外の、要件とは関係のないそれらの特別名詞が相当数加わるので主語系譜は複雑にはなる。

【００２７】
主語系譜の作成方法を下記に例を以て示す。図２－３「ソースプログラム例の主語系譜」参照。

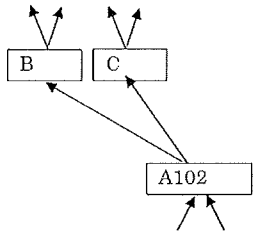
段落番号００１５に記載した下記の構文の脈絡（脈絡については同段落番号参照）は下記の図になる。名詞にはその値が生成される座標を記す。Ａ以外の名詞の座標は省略する。

```
101  if X>Y
102  A=B+C
103  else
104  A=E+F
105  P=S+T
106  end-if
```



注記：＊B, C, X, Y は A102（子）の親、E, F, X, Y は A104（子）の親。
＊end-if の下流で A の値が使われるかもしれないので　真 A102、偽 A104 は A にまとめる

若し脈絡Ａに条件文が無ければ、脈絡Ａは下記のようになる。



名詞の値を生成するために係る名詞、定値、定置は「親」であり、値を生成される名詞は「子」である。条件文に含まれる名詞、定値、定置も親である。
全ての名詞、定値、定置について、上記の脈絡の親子関係を連鎖すれば、起点である外部出力名詞から終点（始祖）である名詞、定値、定置までの系譜を表す主語系譜が出来る。

【００２８】
図１－１乃至図１－４及び図１－５に示したソースプログラムの例についての主語系譜を図２－３に示す。起点には入力線は無い。入力線がない名詞は、numとfp2である。但しfp2はプログラムでは出力しない。fp2は主語系譜上で他の名詞の値に影響を与えないことが分かる。
下記の定置は出力されるだけで他の名詞に影響を与えない文字列なので基本的には主語系譜には登場しない。登場させても良いが孤立して存在しているだけで系譜はない。
“Start.￥n”
“Usage: SampleC infilename outfilename￥n”
勿論上記段落番号００２６に記した通り、主語系譜に入出力命令等を特別名詞化して入れても良い。その主語系譜を自動作成ツールを用いて出力した図を図２－４に示す。

【００２９】

５．２　主語系譜においてバグの起因と実行エラーの発見法

５．２．１　主語系譜からバグの起因を発見する方法。

主語系譜において、上記の通り、起点である外部出力名詞から、終点である入力名詞、定値、定置（始祖）までの間に全ての名詞が親子関係で連なって含まれ、全ての個々の名詞から必ず始祖まで脈絡（矢印線）を辿って到達することが出来なければならない。また矢印線は座標の大きい名詞から座標の小さい名詞に向かって結べなければならない。始祖まで到達できない名詞、或いは矢印線の両端の座標が大から小でなく、少から大へと逆転している名詞はバグの起因である。これらの探索作業は人が行っても良いし、コンピュータプログラムに実行させても良い。

【００３０】

５．２．２　バグの起因とそれが実行エラーかどうかの判定方法。

例を以て説明する。図３参照。

構文と図３の主語系譜の原形から導出された主語系譜は下図の通りである。

101　a=b+c

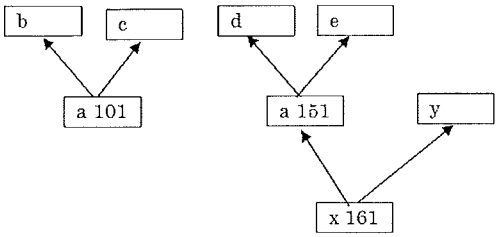
・

151　a=d+e

・

161　x=a+y

主語系譜の各名詞にはその値が生成される座標を付与する。名詞x161（子）の変数主語（親）として主語系譜の中に同一名詞が複数ある場合（a 101、 a 151）は、その内TCXの大きい方をxの変数主語(親)とする。



行番号（TCX）１６１のxの値を生成するaの値として、OSは直近のTCX１５１で生成されたaの値を勝手に自動的に使う。しかし、同一処理経路の上流でaの値はTCX１０１でも生成されており、いずれの値が要件に合致しているのかを人が確認しなければならない。換言すれば、TCX１５１のタイミングで生成されたaの値で良いのか、それともTCX１０１のタイミングで生成されたaの値が正しいのか、いずれが要件に合致しているのかは人が判断しなければならない。名詞数が臨界数を超えるプログラム、或いは、プログラムを別々な人が分担作成した後に合体させるプログラムの場合は、TCX１６１のxを生成するaは、要件上はTCX１０１で生成されたaの値を使わなければならないのに、プログラムの実行上はTCX１５１で生成されたaの値を使ってしまうと言う誤りがないとは言えない。これを「タイミング違反」と言う。その場合は実行エラーとなる。しかしそれを人は気が付かないし、テストデータでテストしても検出は難しい。テストの実行結果が想定外の突飛な値でない限りプログラムは正常に見えるからである。

勿論、要件上でも実行されるTCX１５１で生成されたaの値を使うことが正しいことが多い。その場合は実行エラーとはならない。いずれにしても、同一名詞の値を生成する構文（名詞の値の定義文）が処理経路に2つ以上ある場合は、それが実行エラーとなる可能性があるので、それが２つ以上ある事、即ち２つの名詞がバグの起因である。

【００３１】

バグの起因の発見とそれが実行エラーとなるかどうかの判定方法について述べる。主語系

譜を、人がまたはコンピュータプログラムによって観察し、aの定義文が２つあり、それらが同一処理経路上にあるかどうかを調べる。その方法は調和脈絡上aの一方から他方へ辿れるかどうかを調べる方法である。例示の構文では調和脈絡において辿ることが出来るので、それらは同一処理経路上にあることが分かる。その場合、ＴＣＸ１６１のタイミングで親として使う名詞は自動的にa151であるが、要件上ではa101を親として使わなくてよいかどうかを人が要件に照合して判定する。この判定は、テストデータなしで判定できるし、テストデータでは判定できない。このプログラムエラーは主語系譜で発見する以外に方法が無い。テストデータでテストしても検出は難しい。テストの実行結果が想定外の突飛な値でない限りプログラムは正常に見えるからである。

【００３２】

５．２．３　実行エラーとなるバグの起因とその発見方法

例を以て説明する。図４参照。

構文と図４の主語系譜の原形から導出された主語系譜は下図の通りである。

101　a=b+c

102　x=a+b

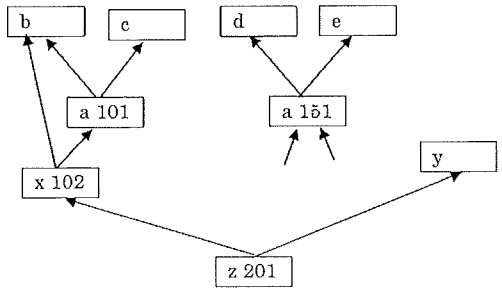
・

151　a=d+e

・

201　x=a+y

上記の構文の主語系譜は下図となる。各名詞にはその値が生成される座標を付与する。主語系譜の中に同一名詞が複数あるが、この例の場合のようにx 102のタイミングにおいては、a 101しかないことに注意。



行番号（TCX）２０１のzの値を生成するxの値として、OSは直近のTCX１０２で生成されたxの値を自動的に使う。そのxの値は、TCX１０１で生成されたaの値（a－１０１）を使う。しかし、TCX２０１が実行される前にTCX１５１でaの値は別の値（a－１５１）に変わっている。要件上は、zの値はそれが生成されるタイミング（TCX２０１）におけるxの値を使って生成されなければならないし、そのxの値はタイミング（TCX２０１）におけるaの値（a－１５１）を使って生成しなければならない。しかしこのプログラムの実行では、xの値はその前のタイミングのaの値（a－１０１）を使って生成される。だから要件に合致しない間違った実行をしてしまうので実行エラーである。即ち、名詞の値を生成する「タイミング」の違反がバグの起因である。このことを名詞の値が同期していないとも言う。

【００３３】

この種のプログラムエラーは、テストデータを使用してプログラムエラーを発見しようとしても不可能である。なぜならプログラムは実行できるし、出力値も突飛な想定外な値でない限り異常として発見できないからである。この種のプログラムエラーは、名詞数が臨界数を超えるプログラム、或いは、プログラムを別々な人が分担作成した後に合体させるプログラムの場合に避けられない。このプログラムエラーは主語系譜で発見する以外に方

法が無い。
【００３４】
この種のプログラムのソースには下記A、B、C、の特徴がある。
A.　(定義が異なる同名の名詞が複数有ること)
定義が異なる同一名の名詞の値を生成する構文（名詞の値を生成する文を主語定義文という。）が同一処理経路上に２つ以上あり、（図４のTCX１０１とTCX１５１におけるaの主語定義文）、
B.　(上記AのTCXの間にその名詞を変数主語として使う他の名詞の主語定義文が有ること)
上記Aの当該名詞aを変数主語とする主語定義文（x=a+b）が、Aの2つ以上のTCX（１０１と１５１）の間のどこかに存在し（TCX１０２のxの定義文）、
C.　(Aよりも下流でBで定義した名詞が変数主語として使われること)
上記Bの当該主語xを変数主語とする主語定義文（z=x+y）のTCXが、上記Bの主語定義文（TCX１０２のxの定義文）をはさむ上記Aの主語定義文のTCX（１０１、１５１）の内の大きい方よりも大きい。
【００３５】
このバグの起因を主語系譜から抽出する方法は下記の通りである。
主語系譜を、人またはコンピュータプログラムによって観察し、aの定義文が2以上あり、それらが同一処理経路上にあるかどうか調べる。その方法は調和脈絡上aの一方から他方へ辿れるかどうかを調べる方法である。例示の構文では調和脈絡において辿ることが出来るので、それらは同一処理経路上にあることが分かる。更にaのTCX（101, 151）の間にaを変数主語とする名詞x102の定義文があり、且つ該名詞x102を変数主語として使う名詞zのＴＣＸが２つのaのTCXの大きい方よりも大きい。従って段落番号００３４に記載したＡ，Ｂ，Ｃの条件を全て満たすので実行エラーとなる。バグの起因となる名詞はaである。

【００３６】
５．３　従来はバグの定義は無かった
これまで、この種のバグの定義はできていなかった。その理由は全てのバグの定義を行うとすれば、数百、数千の定義が必要になり誰もそれを公理的に整理することが出来ないからである。本方法の様にバグの起因を普遍的なＬＹＥＥ空間の欠陥、並びに主語系譜の欠陥として捉え直せばバグの起因は公理的に定義することが出来る。これにより、プログラムに内在するバグの起因は全て捉えられる。従来はこのようにしてバグの起因を発見することは出来なかった。これまでのテスト方法では調和脈絡のテストしか行われていない。超言語脈絡のテストが偶然に行われることがあっても、従来プログラムの作成者がこの脈絡を意識していないので超言語脈絡のテストは行われていない。

【００３７】
６．全脈絡に占める超言語脈絡の割合。保守でプログラムはカオスとなる。
主語系譜上でプログラムの様相を観察すると、平均的に調和脈絡は７０％、超言語脈絡は３０％を占めている。そして、従来プログラムのテストではこの３０％はほぼテストされずに見過ごされている。即ち、信じられないかもしれないが、従来プログラムはこれら未テストプログラムを内在したまま稼働しているという事である。つまり、従来プログラムは７０％の完成度で稼働していると考えられる。故に、超言語脈絡のことを知らない作業者が超言語脈絡に干渉する内容のプログラム箇所に保守を行うと、結果的にバグの起因を誘導することになりそのシステムは実行中、突発的に、彼には想定外のデータ撹動、データ破壊、プログラム破壊の類の事態が生じる。そして、その事故の改修は超言語脈絡の知識がないので、そこを修正するのではなく、別のロジックがドンと加えられる。こうして、プログラムは次第に不安定なカオス世界に向かうのである。この深刻な課題を解決する手段が今までは無かったが、本願発明はその課題の解決の根本的手段となる。開発後保守が長きにわたり積み重ねられればバグの起因が必ず多数潜在する。これが稼働中のプログ

ラムの実情である。この様な実情でＡＩ、自動化、ウイルス対策が行われても実行障害が出ない筈がない。

【００３８】
７．本プログラムがテストデータを不要とするわけ
本発明を実行するプログラムはテスト対象プログラムの言語に依存して作成される。しかし、本プログラムのアルゴリズムは普遍的である。普遍的とは、そのアルゴリズムは唯一であるし例外がないということである。これはＬＹＥＥ空間、主語系譜の効果である。そして、この背景にはシナリオ関数を成立させる仕組みが置かれている。因みに、ＬＹＥＥ空間はシナリオ関数の解を捉える形而上学的３次元空間の事である。シナリオ関数については段落番号００１１参照。
本アルゴリズムを用いて、ひとが机上でテスト対象のソースプログラムを
処理すれば、そのひとがテスト対象プログラムの言語特性を問わず、そのプログラムに内在する全バグの起因を捉えることが出来る。もし、ひとがこの机上作業を行えばその人は本アルゴリズムを完全に理解することが出来る。その意味で本アルゴリズムはプログラム作成法の経典（バイブル）の役割をはたしている。

【００３９】
８．特筆される本プログラムの特徴
（１）本発明の方法は単体テスト、組み合わせテスト、総合テストのいずれにも適用出来る。
（２）本発明の方法をテスト対象のプログラム記述言語に合わせてプログラムし、ＰＣに搭載すれば、そのＰＣはその言語の任意のコンパイル済みのプログラムのソースを唯一の入力データとして、そのプログラムの全てのバグの起因の探索を自動的に行うツールとなる。
（３）プログラム言語を問わずどの分野のプログラムにも本発明の方法でテストができる。
（４）本発明の方法でソースプログラムの完全系が求められる。

【００４０】
９．ＬＹＥＥ空間及び主語系譜が捉えるソースプログラムの欠陥とソースプログラムを紐付ける方法
段落番号００１９に記載したように、ＬＹＥＥ空間にはテストされるプログラムの記述言語の文法規則に従ってテストされるプログラムが図１－１乃至図１－４の表の項目に展開される。結果、ＬＹＥＥ空間では文法規則に反するプログラムの不完全性が論理バグとして捉えられる。これらの論理バグはLYEE空間の各項目の値が得られない項目として出て来るので、値が得られない項目の構文のTCX即ち行番号を見れば、論理バグを持つ構文の位置と紐付けすることが出来る。
一方、主語系譜においてプログラムの欠陥は、前記のバグの起因として捉えられる。バグの起因は、段落番号００３０で述べた通り、実行エラーになるものと、実行エラーにはならないものがあるが、いずれのバグの起因も起因となる名詞のTCXで示されるので、それらが属する構文の位置と紐付けることが出来る。

【００４１】
１０．　バグの波及範囲
バグの波及範囲を特定する方法は従来存在しなかった。本願発明の主語系譜によって初めてそれを特定することが出来る。
バグの波及範囲は、バグの起因となり実行エラーとなると判断された名詞をaとする時、aを変数主語として生成される名詞達（ｂ＝a+cのｂ。aはｂの親、ｂはaの子。段落番号００１７参照。）、更にaを先祖とする全ての名詞達に及ぶ。

【００４２】
１２．本明細書の語の索引
＊　主語と変数主語　段落番号０００２
＊　バグとバグの起因　段落番号０００１、００１０、００３０
＊　名詞、定値、定置　段落番号０００７
＊　脈絡　段落番号００１５
＊　終点、起点、始祖、子孫　段落番号００２４
＊　同期　段落番号０００７
＊　正統なプログラム　段落番号００１０
＊　完全性を持つプログラム　段落番号００１０
＊　バグの起因を含まないプログラム　段落番号００１０
＊　タイミング違反　段落番号００３０、００３２
＊　調和脈絡、超言語脈絡、主語系譜（原形）
段落番号００１５
＊主語系譜の最終形　段落番号００２５
＊　座標　段落番号００１９
＊　ＴＣＸ、ＴＣＹ、ＴＣＺ１～４　段落番号００２０
＊　親と子　段落番号００１７
＊　主語定義文　段落番号００３４
＊　シナリオ関数　段落番号００１１
＊　同一処理経路　段落番号００３５

【図面の簡単な説明】
【００４３】
【図１－１】ソースプログラム例のＬＹＥＥ空間　その１ＬＹＥＥ空間はソースプログラムの全構文を含む表であるので縦に非常に長い。特許の図面様式のサイズに制限があり、1ページに記載できる行数に限りがあるので、制限範囲に入るように構文を図１－１（その1）から図１－４（その４）まで4つの表に分割して表示した。各表の項目欄は共通であるが各表毎に表示した。
【図１－２】ソースプログラム例のＬＹＥＥ空間　その２
【図１－３】ソースプログラム例のＬＹＥＥ空間　その３
【図１－４】ソースプログラム例のＬＹＥＥ空間　その４
【図１－５】ソースプログラムの例
【図２－１】ＬＹＥＥ空間から求められる主語系譜の原形の例　上部特許の図面様式のサイズに制限があり、その制限に合わせて本主語系譜図を縮小すると文字が読みにくくなるので、本主語系譜図を上下に２分割した。つながるべき線同志には同じ符号を付した。
【図２－２】ＬＹＥＥ空間から求められる主語系譜の原形の例　下部
【図２－３】ソースプログラム例の主語系譜
【図２－４】ソースプログラム例で命令文等を特別名詞化した主語系譜
【図３】バグの起因を含むが実行エラーとならないプログラムの主語系譜
【図４】バグの起因を含み実行エラーとなるプログラムの主語系譜
【発明を実施するための形態】
【００４４】
ＬＹＥＥ空間（図１－１乃至図１－４）、主語系譜の原形（図２－１、図２－２）、主語系譜（図２－３）参照。
テストされるべきソースプログラムがあれば、本発明の方法でプログラムテストが実施できる。図１－１乃至図１－４は事例として示すコンパイル済みの正統なＣ言語ソースプロ

グラムとそのＬＹＥＥ空間である。図２－１、図２－２はその主語系譜の原形である。図２－３はその主語系譜である。
図１－１乃至図１－４のＬＹＥＥ空間、並びに図２－１、図２－２及び図２－３の主語系譜にはバグの起因は見当たらない。
そのわけはこのプログラムに属す名詞数が、われわれの思考を曖昧にさせる臨界数以下だからである。しかし、この様な規模のプログラムでも幾つかがシステムとして集合化されれば、その名詞の総数は増大し、そして、臨界を超えれば、その全名詞のどれかはバグの起因となる。バグの起因は、単に同名の直近で成立した番地の内容を利用するコンピュータの動作だけでは捉えられない。元来コンパイラにはこの問題を検証する能力はない。
プログラムがバグの起因を含むかどうか、実行エラーとなるかどうかを検証できる方法は、主語系譜による段落番号００２９乃至００３２で記載した通りの方法しかない。
テストデータによるプログラムの検査方法には、検査できない部分が多く残っており、検査を完了してもその製品は未完成品である。

【産業上の利用可能性】
【００４５】
現在、プログラムのテストはテストデータを使う方法でしか行われていない。しかしその方法はバグの起因の検出方法としては原理的に不完全であるので未検出の潜在バグが必ず残存している。更に長年の保守によって潜在バグは累積し多くのプログラムの内容は手が付けられないカオスになっているのが現状である。この状況でプログラムが稼働しているのだから稼働中にプログラムのバグによる重大な事故が発生しないとは保証できない。いつどんな不具合・事故が発生するか予測もつかない道具を使って公的機関・産業が運営されているのである。一旦事故が起こればそれは企業の存亡を左右しかねないし、産業上の重大な問題である。しかし現在まで解決の手段がなかった。今後は本発明の方法を使えば、全てのプログラムの潜在するバグが全て検出されるのであるから、政府機関・企業は安心して運営することが出来るので、国や産業上の利用可能性は計り知れないものがある。

【書類名】特許請求の範囲
【請求項１】
コンパイル済みプログラムのソースコードからＬＹＥＥ空間を自動生成し、そのＬＹＥＥ空間の完成を妨げている名詞或いは構文を、そのプログラムに内在するコンパイラーが捉えられない論理矛盾に係るバグの起因として、テストデータを用いてプログラムを実行することをせずに、自動検出することを特徴とする方法。

【請求項２】
請求項１記載のＬＹＥＥ空間から主語系譜を自動生成し、その主語系譜を用いてそのプログラムに内在するバグの起因を、テストデータを用いてプログラムを実行することをせずに、自動検出することを特徴とする方法。

【請求項３】
請求項２記載の主語系譜を観察し、
（１）定義が異なる同名の名詞aの主語定義文が同一処理経路上に複数存在するかどうか、
（２）それら名詞a達の座標の間に、名詞aを変数主語として使う他の名詞ｂの主語定義文が存在するかどうか、
（３）該名詞bをはさむ二つの名詞aの定義文の下流に名詞bが変数主語として使われているかどうか、
を、全名詞について、テストデータを用いてプログラムを実行することをせずに、自動判定してバグの起因と実行時エラーを自動検出することを特徴とする請求項２記載の方法。

【請求項４】
請求項１乃至請求項３で捉えるプログラムに内在するバグの起因を、テスト対象プログラムのソース構文の行位置と関連付けて表示することを特徴とする請求項１乃至請求項３記載の方法。

【請求項５】
請求項２乃至請求項３記載の方法によって自動検出されたバグの起因による起こるバグが波及する全名詞の範囲を、テストデータを用いてプログラムを実行することをせずに、主語系譜を用いて自動的に抽出することを特徴とする方法。

【書類名】要約書
【要約】
【課題】プログラムのテストは、テストデータを使用してコンパイル済のプログラムを実行させて行うが、テストで行われるべきことが関係者には十分理解がされていないので、プログラムエラーが残存する。たとえば、全変数の値が夫々の要件に合致した条件とタイミングで得られた値であるかどうか、即ち同期性の検証は不可欠であるのに行われていない。だからプログラムは、処理の正しさや稼働中のトラブルが起こらない保証がないまま運営されている。
【解決手段】本発明の方法は、コンパイル済みのプログラムソースから、「ＬＹＥＥ空間」を自動生成し、且つＬＹＥＥ空間から「主語系譜」を自動生成し、それらを自動解析して、プログラムの実行時に発症するバグの起因を自動的に捉え、それらをソースプログラムの構文の位置で示し、更にバグの波及する範囲を導出する。テストデータを用いてプログラムを起動させないことが本方法の特徴である。
【選択図】図２－３

【書類名】図面

【図１－１】

TOX	プログラム構文のソース	構文解析の要否	座標付きの構文と単元化された構文の要素	構文の要素のTOX	構文の要素の種別	構文またはその要素の調和度探					
						TOX	TCY	TCZ1	TCZ2	TCZ3	TCZ4
1	#include <stdio.h>	否		1	翻訳文	1	2	黒	黒	黒	黒
2	//	否		2	注釈文	2	3	黒	黒	黒	黒
3	int main(int argc, char* argv[])	要	int main[3-1]([3-2]int argc[3-3],[3-4] char* argv[][3-5])[3-6]	3	領域定義文	3	4	黒	黒	黒	黒
			int main	3-1	翻訳文	黒	黒	黒	黒	黒	黒
			(	3-2	翻訳文	黒	黒	黒	黒	黒	黒
			int argc	3-3	名前	黒	黒	黒	黒	黒	黒
			,	3-4	翻訳文	黒	黒	黒	黒	黒	黒
			char* argv[]	3-5	名前	黒	黒	黒	黒	黒	黒
			)	3-6	翻訳文	黒	黒	黒	黒	黒	黒
4	{	否		4	翻訳文	4	5	黒	黒	黒	黒
5	printf_s("Start.¥n");	要	printf_s[5-1]([5-2]"Start.¥n"[5-3])[5-4];[5-5]	5	出力文	5	6	黒	黒	黒	黒
			printf_s	5-1	出力文	黒	黒	黒	黒	黒	黒
			(	5-2	翻訳文	黒	黒	黒	黒	黒	黒
			"Start.¥n"	5-3	定数	黒	黒	黒	黒	黒	黒
			)	5-4	翻訳文	黒	黒	黒	黒	黒	黒
			;	5-5	翻訳文	黒	黒	黒	黒	黒	黒
6	FILE* fp1 = 0;	要	FILE* fp1[6-1] = 0[6-2];[6-3]	6	領域定義文	6	7	黒	黒	黒	黒
			FILE* fp1	6-1	名前	黒	黒	黒	黒	黒	黒
			0	6-2	定価	黒	黒	黒	黒	黒	黒
			;	6-3	翻訳文	黒	黒	黒	黒	黒	黒
7	FILE* fp2 = 0;	要	FILE* fp2[7-1] = 0[7-2];[7-3]	7	領域定義文	7	8	黒	黒	黒	黒
			FILE* fp2	7-1	名前	黒	黒	黒	黒	黒	黒
			0	7-2	定価	黒	黒	黒	黒	黒	黒
			;	7-3	翻訳文	黒	黒	黒	黒	黒	黒
8	int num = 0;	要	int num[8-1] = 0[8-2];[8-3]	8	領域定義文	8	9	黒	黒	黒	黒
			int num	8-1	名前	黒	黒	黒	黒	黒	黒
			0	8-2	定価	黒	黒	黒	黒	黒	黒
			;	8-3	翻訳文	黒	黒	黒	黒	黒	黒
9	if (argc <= 2)	要	if[9-1] ([9-2]argc[3-3] <= 2[9-3])[9-4][9,,10,13,13,14]	9	条件文	9	10	黒	黒	黒	14
			if	9-1	条件文	黒	黒	黒	黒	黒	黒
			(	9-2	翻訳文	黒	黒	黒	黒	黒	黒
			argc	9,,3-3	名前	黒	黒	黒	黒	黒	黒
			<=	9-3	定価	黒	黒	黒	黒	黒	黒
			2	9-4	翻訳文	黒	黒	黒	黒	黒	黒
			)	10	翻訳文	10	11	黒	黒	黒	黒
10	{	否									
11	printf("Usage : SampleC infilename outfilename¥n");	要	printf[11-1]([11-2]"Usage : SampleC infilename outfilename¥n"[11-3])[11-4];[44-5]	11	出力文	11	12	黒	黒	黒	黒
			printf	11-1	出力文	黒	黒	黒	黒	黒	黒
			(	11-2	翻訳文	黒	黒	黒	黒	黒	黒
			"Usage : SampleC infilename outfilename¥n"	11-3	定数	黒	黒	黒	黒	黒	黒
			)	11-4	翻訳文	黒	黒	黒	黒	黒	黒
			;	11-5	翻訳文	黒	黒	黒	黒	黒	黒
12	return 1;	要	return[12-1] 1[12-2];[12-3]	12	制御文	12	36	黒	黒	黒	黒
			return	12-1	制御文	黒	黒	黒	黒	黒	黒
			1	12-2	定価	黒	黒	黒	黒	黒	黒
			;	12-3	翻訳文	黒	黒	黒	黒	黒	黒
13	}	否		13	翻訳文	13	14	黒	黒	黒	黒

【図１－２】

TCX	プログラムソースのソース	構文解析の要否	プログラムソースの構文解析情報 座標付きの構文と単元化された構文の要素	構文の要素のTCX	構文の要素の種別	TCX	TCY	構文または構文要素の開始座標 TCZ1	TCZ2	TCZ3	TCZ4
14	errno_t status1 = fopen_s(&fp1, (const char*)argv[1], "rt");	要	errno_t status1[14-1] = fopen_s[14-2](([14-9]&fp1[6-1],[14-4] (const char*)argv[1][3-5],[14-5] "rt"[14-6])[14-7];[14-8]	14	領域定義文	14	15	無し	無し	無し	無し
			errno_t status1	14-1	名詞	無し	無し	無し	無し	無し	無し
			fopen_s	14-2	翻訳文	無し	無し	無し	無し	無し	無し
			(	14-3	翻訳文	無し	無し	無し	無し	無し	無し
			fp1	14, 8-1	名詞	無し	無し	無し	無し	無し	無し
			,	14-4	翻訳文	無し	無し	無し	無し	無し	無し
			argv[1]	14, 3-5	名詞	無し	無し	無し	無し	無し	無し
			, "rt"	14-5	翻訳文	無し	無し	無し	無し	無し	無し
			)	14-6	定詞	無し	無し	無し	無し	無し	無し
			;	14-7	翻訳文	無し	無し	無し	無し	無し	無し
			:	14-8	翻訳文	無し	無し	無し	無し	無し	無し
15	if (status1 == 0)	要	if[15-1] ([15-2]status1[14-1] == 0[15-3])[15-4](15, .16,33,33,34)	15	条件文	15	無し	16	33	33	34
			if	15-1	条件文	無し	無し	無し	無し	無し	無し
			(	15-2	翻訳文	無し	無し	無し	無し	無し	無し
			status1	15, 14-1	名詞	14	無し	無し	無し	無し	無し
			0	15-3	定詞	15-3	無し	無し	無し	無し	無し
			)	15-4	翻訳文	無し	無し	無し	無し	無し	無し
16	{	否		16	翻訳文	16	17	無し	無し	無し	無し
17	errno_t status2 = fopen_s(&fp2, (const char*)argv[2], "wt");	要	errno_t status2[17-1] = fopen_s[17-2](([17-9]&fp2[17, 7-1],[17-4] (const char*)argv[2][3-5],[17-5] "wt"[17-6])[17-7];[17-8]	17	領域定義文	17	18	無し	無し	無し	無し
			errno_t status2	17-1	名詞	無し	無し	無し	無し	無し	無し
			fopen_s	17-2	翻訳文	無し	無し	無し	無し	無し	無し
			(	17-3	翻訳文	無し	無し	無し	無し	無し	無し
			fp2	17, 7-1	名詞	無し	無し	無し	無し	無し	無し
			,	17-4	翻訳文	無し	無し	無し	無し	無し	無し
			argv[2]	17, 3-5	名詞	無し	無し	無し	無し	無し	無し
			, "wt"	17-5	翻訳文	無し	無し	無し	無し	無し	無し
			)	17-6	定詞	無し	無し	無し	無し	無し	無し
			;	17-7	翻訳文	無し	無し	無し	無し	無し	無し
			:	17-8	翻訳文	無し	無し	無し	無し	無し	無し
18	if (status2 == 0)	要	if[18-1] ([18-2]status2[17-1] == 0[18-3])[18-4](18, .18,31,31,32)	18	条件文	18	無し	19	31	31	32
			if	18-1	条件文	無し	無し	無し	無し	無し	無し
			(	18-2	翻訳文	無し	無し	無し	無し	無し	無し
			status2	18, 17-1	名詞	無し	無し	無し	無し	無し	無し
			0	18-3	定詞	無し	無し	無し	無し	無し	無し
			)	18-4	翻訳文	無し	無し	無し	無し	無し	無し
19	{	否		19	翻訳文	19	20	無し	無し	無し	無し
20	while (true)	要	while[20-1] ([20-2]true[20-3])[20-4]	20	条件文	20	21	無し	無し	無し	無し
			while	20-1	条件文	無し	無し	無し	無し	無し	無し
			(	20-2	翻訳文	無し	無し	無し	無し	無し	無し
			true	20-3	定詞	無し	無し	無し	無し	無し	無し
			)	20-4	翻訳文	無し	無し	無し	無し	無し	無し
21	{	否		21	翻訳文	21	22	無し	無し	無し	無し
22	char buf[80];	要	char buf[80][. 22-1];[22-2]	22	領域定義文	22	23	無し	無し	無し	無し
			char buf[80]	22-1	名詞	無し	無し	無し	無し	無し	無し
			;	22-2	翻訳文	無し	無し	無し	無し	無し	無し

【図１－３】

TCX	プログラム構文のソース	構文解析の要否	プログラムソースの構文解析情報 座標付きの構文と置換化された構文の要素	構文の要素のTCX	構文の要素の属別	TCX	構文または構文要素の調和座標 TCY	TCZ1	TCZ2	TCZ3	TCZ4
23	if (fgets(buf, 80, fp1) == NULL)	要	if[23-1]([23-2]fgets[23-3]([23-4]buf[22-1],[23-5] 80[23-6],[23-7]fp1[6-1])[23-8] == NULL[23-9])[23-10]{23, .24,26,27}	23	条件文	23	黒	24	26	26	27
			if	23-1	条件文	黒	黒	黒	黒	黒	黒
			(	23-2	翻訳文	黒	黒	黒	黒	黒	黒
			fgets	23-3	入力文	黒	黒	黒	黒	黒	黒
			(	23-4	翻訳文	黒	黒	黒	黒	黒	黒
			buf	23, 22-1	名詞	黒	黒	黒	黒	黒	黒
			,	23-5	翻訳文	黒	黒	黒	黒	黒	黒
			80	23-6	定修	黒	黒	黒	黒	黒	黒
			,	23-7	翻訳文	黒	黒	黒	黒	黒	黒
			fp1	23, 6-1	名詞	黒	黒	黒	黒	黒	黒
			)	23-8	翻訳文	黒	黒	黒	黒	黒	黒
			NULL	23-9	定修	黒	黒	黒	黒	黒	黒
			)	23-10	翻訳文	黒	黒	黒	黒	黒	黒
24	{	否		24	翻訳文	24	黒	25	黒	黒	黒
25	break;	要	break[25-1];[25-2]{25, .30}	25	制御文	25	黒	30	黒	黒	黒
			break	25-1	制御文	黒	黒	黒	黒	黒	黒
			;	25-2	翻訳文	黒	黒	黒	黒	黒	黒
26	}	否		26	翻訳文	26	黒	27	黒	黒	黒
27	fputs(buf, fp2);	要	fputs[27-1]([27-2]buf[22-1],[27-3]fp2[7-1])[27-4];[27-5]	27	出力文	27	黒	28	黒	黒	黒
			fputs	27-1	出力文	黒	黒	黒	黒	黒	黒
			(	27-2	翻訳文	黒	黒	黒	黒	黒	黒
			buf	27, 22-1	名詞	黒	黒	黒	黒	黒	黒
			,	27-3	翻訳文	黒	黒	黒	黒	黒	黒
			fp2	27, 7-1	名詞	黒	黒	黒	黒	黒	黒
			)	27-4	翻訳文	黒	黒	黒	黒	黒	黒
			;	27-5	翻訳文	黒	黒	黒	黒	黒	黒
28	num++;	要	num[28, 8-1]++;[28-1]	28	代入文	28	黒	29	黒	黒	黒
			num	28, 8-1	名詞	黒	黒	黒	黒	黒	黒
			num=num+1	28	算術文	黒	黒	黒	黒	黒	黒
			1	28	定修	黒	黒	黒	黒	黒	黒
			;	28-1	翻訳文	黒	黒	黒	黒	黒	黒
29	}	否	{29, .20}	29	制御文	29	黒	20	黒	黒	黒
30	fclose(fp2);	要	fclose[30-1]([30-2]fp2[7-1])[30-3];[30-4]	30	翻訳文	30	31	黒	黒	黒	黒
			fclose	30-1	翻訳文	黒	黒	黒	黒	黒	黒
			(	30-2	翻訳文	黒	黒	黒	黒	黒	黒
			fp2	30, 7-1	名詞	黒	黒	黒	黒	黒	黒
			)	30-3	翻訳文	黒	黒	黒	黒	黒	黒
			;	30-4	翻訳文	黒	黒	黒	黒	黒	黒
31	}	否		31	翻訳文	31	32	黒	黒	黒	黒
32	fclose(fp1);	要	fclose[32-1]([32-2]fp1[6-1])[32-3];[32-4]	32	翻訳文	32	33	黒	黒	黒	黒
			fclose	32-1	翻訳文	黒	黒	黒	黒	黒	黒
			(	32-2	翻訳文	黒	黒	黒	黒	黒	黒
			fp1	32, 6-1	名詞	黒	黒	黒	黒	黒	黒
			)	32-3	翻訳文	黒	黒	黒	黒	黒	黒
			;	32-4	翻訳文	黒	黒	黒	黒	黒	黒
33	}	否		33	翻訳文	33	34	黒	黒	黒	黒
34	printf_s("%d lines.¥nEnd.¥n", num);	要	printf_s[34-1]([34-2]"%d lines.¥nEnd.¥n"[34-3],[34-4] num[, 8-1])[34-5];[34-6]	34	出力文	34	35	黒	黒	黒	黒
			printf_s	34-1	出力文	黒	黒	黒	黒	黒	黒
			(	34-2	翻訳文	黒	黒	黒	黒	黒	黒
			"%d lines.¥nEnd.¥n"	34-3	定数	黒	黒	黒	黒	黒	黒
			,	34-4	翻訳文	黒	黒	黒	黒	黒	黒
			num	34, 8-1	名詞	黒	黒	黒	黒	黒	黒
			)	34-5	翻訳文	黒	黒	黒	黒	黒	黒
			;	34-6	翻訳文	黒	黒	黒	黒	黒	黒

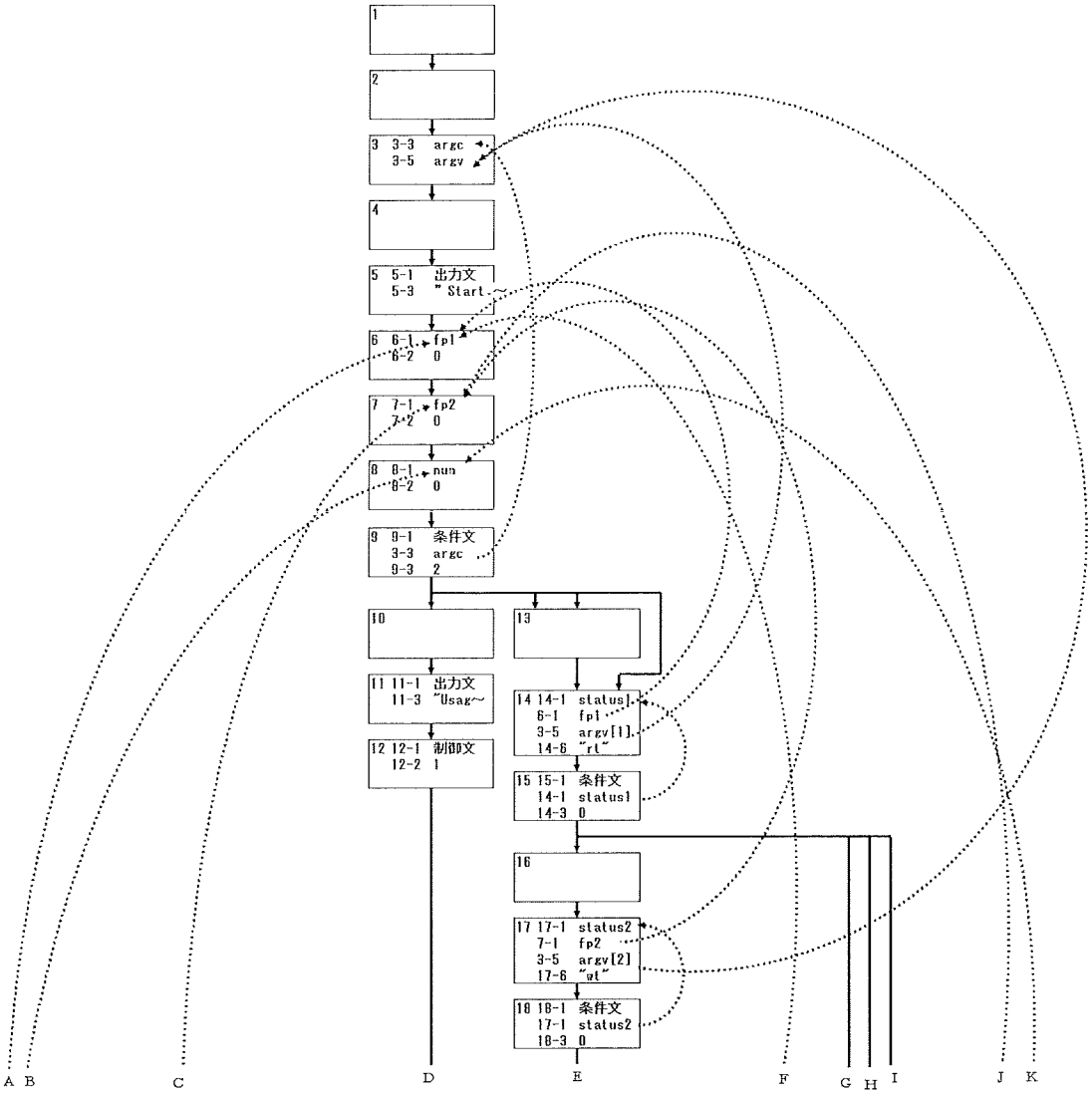
【図１－４】

TOX	プログラム構文のソース	プログラムソースの構文解析情報											
		構文解析の要否	座標付きの構文と展開された構文の要素	構文の要素のTOX	構文の要素の種別	TOX	TOY	TOZ1	TOZ2	TOZ3	TOZ4
35	return 0;	要	return[35-1] 0[35-2];[35-3]	35	制御文	35	36	無し	無し	無し	無し
			return	35-1	制御文	無し	無し	無し	無し	無し	無し
			0	35-2	定 値	無し	無し	無し	無し	無し	無し
			;	35-3	新規文	無し	無し	無し	無し	無し	無し
				36	翻訳文	36	不明	無し	無し	無し	無し
36	}	否									

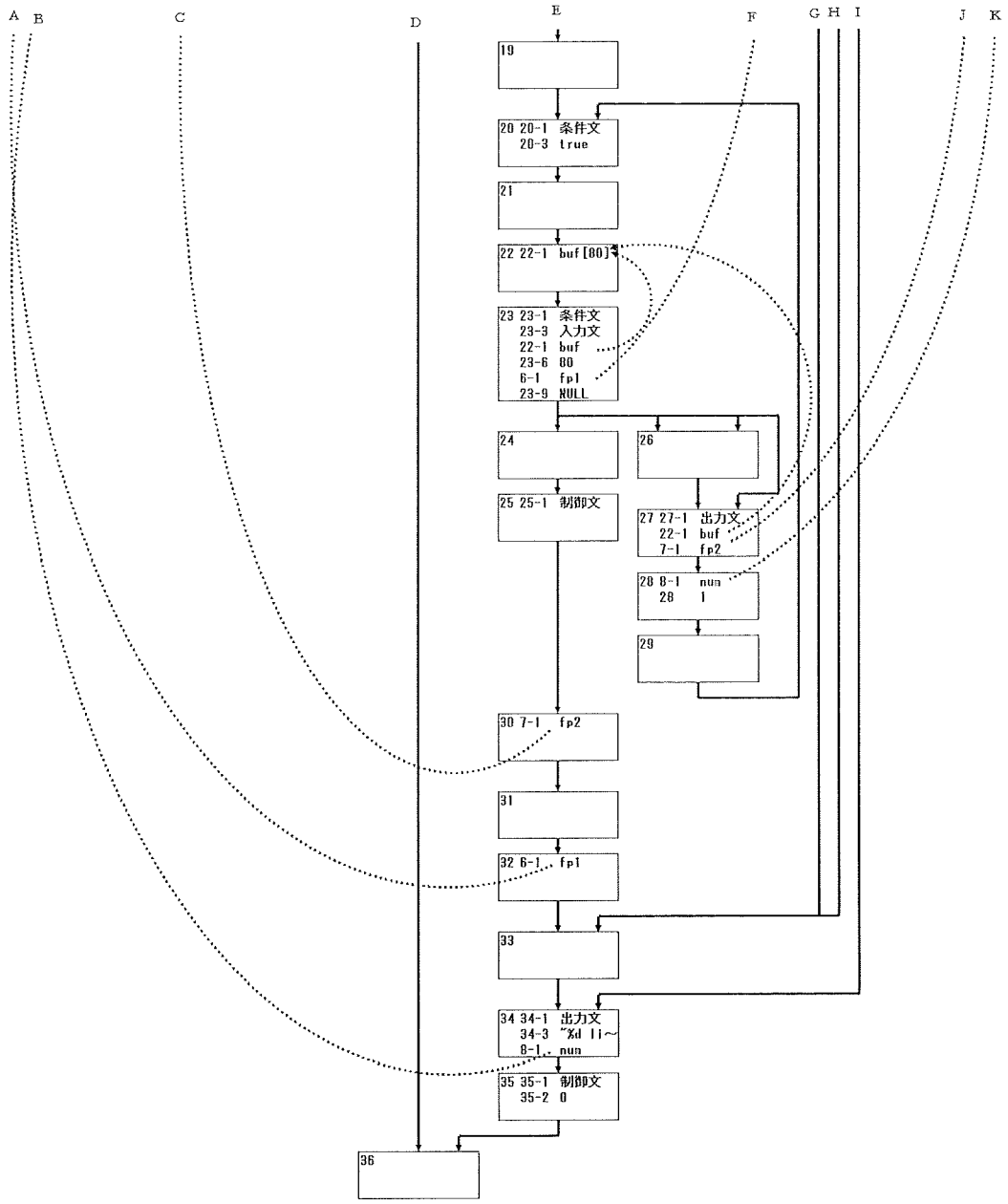
【図１－５】

```
1   #include <stdio.h>
2
3    int main(int argc, char* argv[])
4    {
5               printf_s("Start.¥n");
6               FILE* fp1 = 0;
7               FILE* fp2 = 0;
8               int num = 0;
9               if (argc <= 2)
10              {
11                      printf("Usage : SampleC infilename outfilename¥n");
12                      return 1;
13              }
14              errno_t status1 = fopen_s(&fp1, (const char*)argv[1], "rt");
15              if (status1 == 0)
16              {
17                      errno_t status2 = fopen_s(&fp2, (const char*)argv[2], "wt");
18                        if (status2 == 0)
19                        {
20                                while (true)
21                                {
22                                        char buf[80];
23                                        if (fgets(buf, 80, fp1) == NULL)
24                                         {
25                                                break;
26                                         }
27                                         fputs(buf, fp2);
28                                         num++;
29                                 }
30                                 fclose(fp2);
31                        }
32                        fclose(fp1);
33              }
34              printf_s("%d lines.¥nEnd.¥n", num);
35              return 0;
36     }
```

【図２－１】

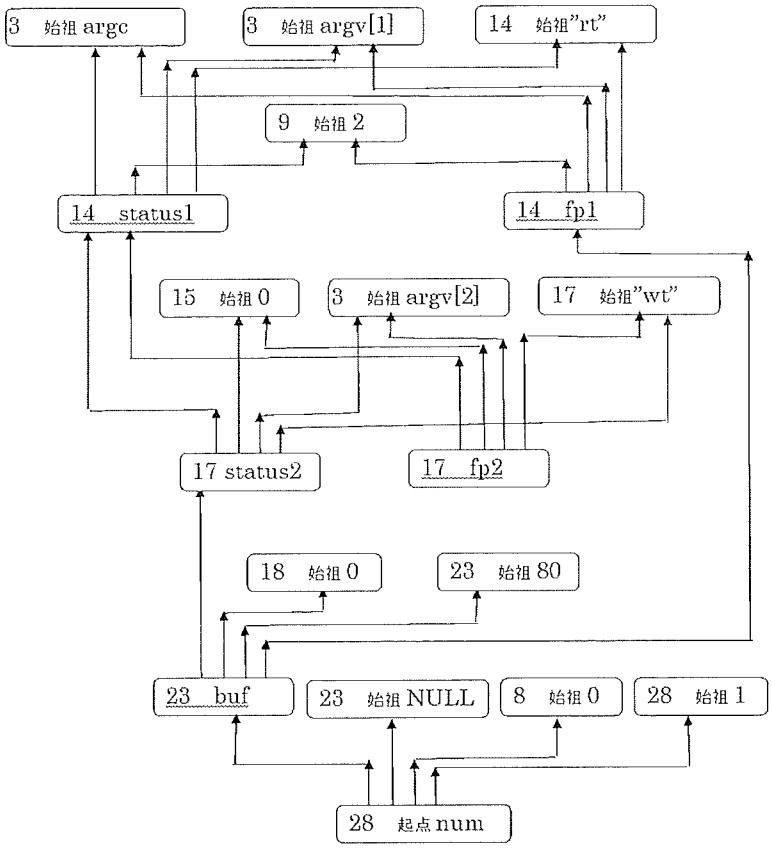

1
2
3 3-3 argc
3-5 argv
4
5 5-1 出力文
5-3 " Start
6 6-1 fp1
6-2 0
7 7-1 fp2
7-2 0
8 8-1 num
8-2 0
9 9-1 条件文
9-3 argc
9-3 2
10
11 11-1 出力文
11-3 "Usag~
12 12-1 制御文
12-2 1
13
14 14-1 status1
6-1 fp1
3-5 argv[1]
14-6 "rt"
15 15-1 条件文
14-1 status1
14-3 0
16
17 17-1 status2
7-1 fp2
3-5 argv[2]
17-6 "wt"
18 18-1 条件文
17-1 status2
18-3 0
A B C D E F G H I J K

【図２－２】

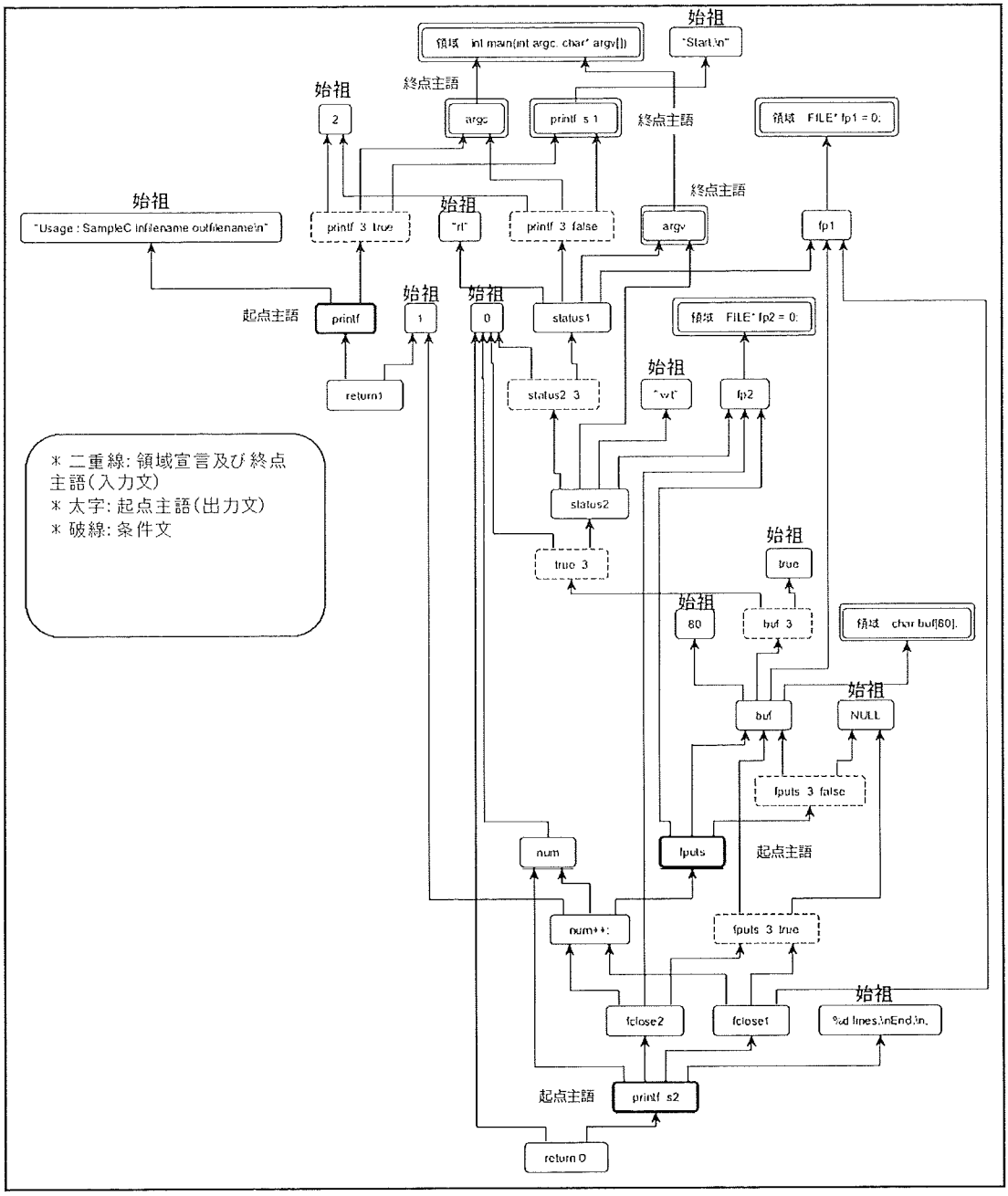

A
B
C
D
E
F
G
H
I
J
K
19
20 20-1 条件文
20-3 true
21
22 22-1 buf[80]
23 23-1 条件文
23-3 入力文
22-1 buf
23-6 80
6-1 fp1
23-9 NULL
24
25 25-1 制御文
26
27 27-1 出力文
22-1 buf
7-1 fp2
28 8-1 nun
28 1
29
30 7-1 fp2
31
32 6-1 fp1
33
34 34-1 出力文
34-3 "%d !!～
8-1 nun
35 35-1 制御文
35-2 0
36

【図２－３】

ソースプログラム例の主語系譜



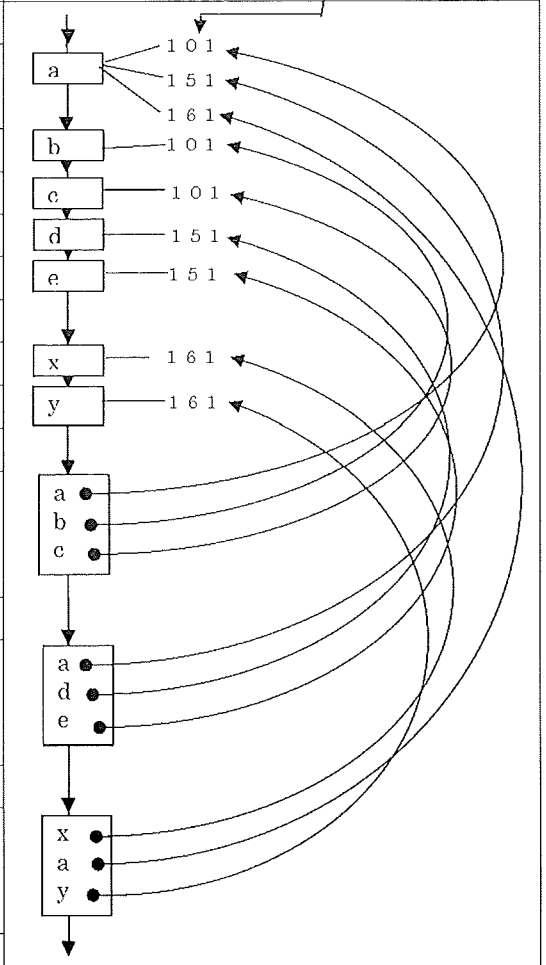
【図２－４】


領域 int main(int argc, char* argv[])
"Start.\n"
始祖
終点主語
2
argc
printf_s 1
終点主語
領域 FILE* fp1 = 0;
始祖
"Usage : SampleC inFilename outFilename\n"
printf_3_true
始祖
"rt"
printf_3_false
argv
終点主語
fp1
起点主語
printf
始祖
1
始祖
0
status1
領域 FILE* fp2 = 0;
return1
status2_3
始祖
"wt"
fp2
＊二重線: 領域宣言及び終点主語(入力文)
＊太字: 起点主語(出力文)
＊破線: 条件文
status2
始祖
true
true_3
始祖
80
buf_3
領域 char buf[80];
始祖
buf
NULL
fputs_3_false
num
fputs
起点主語
num++;
fputs_3_true
始祖
fclose2
fclose1
%d lines.\nEnd.\n.
起点主語
printf_s2
return 0

【図３】

バグの起因を含むがエラーとならないプログラムの主語系譜図

行番号（TCX）	ソース	主語系譜　矢印線源 TCX
０１０	領域定義文ａ	
０１１	領域定義文ｂ	
０１２	領域定義文ｃ	
０１３	領域定義文ｄ	
０１４	領域定義文ｅ	
０２１	領域定義文ｘ	
０２２	領域定義文ｙ	
１０１	a=b+c	
１５１	a=d+e	
１６１	x=a+y	



【図４】

バグの起因を含みエラーとなるプログラムの主語系譜図

行番号（TCX）	ソース	主語系譜　矢印線源 TCX
０１１	領域定義文ａ	
０１２	領域定義文ｂ	
０１３	領域定義文ｃ	
０１４	領域定義文ｄ	
０１５	領域定義文ｅ	
０２１	領域定義文ｘ	
０２２	領域定義文ｙ	
０２３	領域定義文ｚ	
１０１	a=b+c	
１０２	x=a+b	
１５１	a=d+e	
２０１	z=x+y (x>y でも良い。ｘが変数主語なら良い)