(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5073767号
(P5073767)
(45)発行日　平成24年11月14日(2012.11.14)　(24)登録日　平成24年8月31日(2012.8.31)

(51)Int.Cl.　F I
G06F 11/36 (2006.01)　G06F 9/06 620M

請求項の数 3　(全 10 頁)

(21)出願番号　特願2010-27222(P2010-27222)
(22)出願日　平成22年2月10日(2010.2.10)
(65)公開番号　特開2011-164954(P2011-164954A)
(43)公開日　平成23年8月25日(2011.8.25)
審査請求日　平成22年6月21日(2010.6.21)

(73)特許権者　591057256
株式会社エクサ
神奈川県川崎市幸区堀川町５８０番地
(74)代理人　100091096
弁理士　平木　祐輔
(74)代理人　100105463
弁理士　関谷　三男
(74)代理人　100102576
弁理士　渡辺　敏章
(74)代理人　100153903
弁理士　吉川　明
(72)発明者　武市　正人
神奈川県川崎市幸区堀川町５８０番地

審査官　金子　秀彦

最終頁に続く

(54)【発明の名称】ＣＯＢＯＬソースコードチェックプログラム、ＣＯＢＯＬソースコードチェックシステム

(57)【特許請求の範囲】
【請求項１】
ＣＯＢＯＬソースコードのエラーをチェックする処理をコンピュータに実行させるプログラムであって、前記コンピュータに、
ＣＯＢＯＬソースコードを読み込むステップと、
前記ＣＯＢＯＬソースコードの構文を解析し、前記ＣＯＢＯＬソースコードの構文要素を構成要素として保持する構文木を生成するステップと、
前記構文木の構成要素が保持している前記ＣＯＢＯＬソースコードの構文要素をチェックするチェックステップと、
を実行させ、
前記チェックステップでは、前記コンピュータに、
ＣＯＢＯＬ言語のテーブルを参照し、またはデータを書き込むテーブルアクセス処理が前記構文木内に存在するか否かをチェックするステップと、
前記テーブルアクセス処理が存在する場合には、前記テーブルのサイズを超えて前記テーブルにアクセスするステートメントが存在するか否かをチェックするステップと、
前記テーブルのサイズを超えて前記テーブルにアクセスするステートメントが存在する場合は前記ＣＯＢＯＬソースコードにエラーがあると判定するステップと、
を実行させることを特徴とするＣＯＢＯＬソースコードチェックプログラム。
【請求項２】
前記チェックステップでは、前記コンピュータに、

ＰＥＲＦＯＲＭステートメント内にある括弧が付与されたデータ項目を探索するステップと、
前記構文木内から前記データ項目に対応するＯＣＣＵＲＳステートメントとそのＯＣＣＵＲＳステートメントが宣言する前記テーブルのサイズを探索するステップと、
前記構文木内の前記データ項目から遡って、前記テーブルに前記データ項目をセットするステートメントの繰返し条件を探索するステップと、
前記繰返し条件が前記テーブルのサイズを超過しているか否かをチェックするステップと、
前記テーブルのサイズを超過している場合は前記ＣＯＢＯＬソースコードにエラーがあると判定するステップと、
を実行させることを特徴とする請求項１記載のＣＯＢＯＬソースコードチェックプログラム。

【請求項３】
請求項１または請求項２記載のＣＯＢＯＬソースコードチェックプログラムを実行するコンピュータと、
前記ＣＯＢＯＬソースコードが参照している他のＣＯＢＯＬソースコードを、前記ＣＯＢＯＬソースコードまたは前記他のＣＯＢＯＬソースコードが配置されているホストから定期的に取得するサーバと、
を有することを特徴とするＣＯＢＯＬソースコードチェックシステム。

【発明の詳細な説明】

【技術分野】

【０００１】
本発明は、ＣＯＢＯＬ言語で記述されたプログラムソースコードをチェックする技術に関するものである。

【背景技術】

【０００２】
ソフトウェア開発では常に高品質な成果物が求められており、品質向上のために様々な点検活動が行われている。開発の期間は限られているため、品質点検を素早く高精度に実施する必要がある。

【０００３】
開発下流工程の成果物であるソースコードの品質点検に目を向けると、大きくテストと目視点検の２つの方法がある。このうちテストではソフトウェアに要求される仕様を確認し、目視点検では仕様確認に加えてテストで発見しにくい誤作動、非効率な記述、保守性などの品質も確認する。このように多くの内容を確認する目視点検は品質確保の活動として重要であるが、時間がかかる割に見落としが発生しやすく、さらに属人的で精度がばらつくという問題を含んでいる。

【０００４】
近年Ｊａｖａ（登録商標）やＣなどのオープン系言語では目視点検をサポートするツールとしてＦｉｎｄＢｕｇｓ（非特許文献１）やＳｐｌｉｎｔ（非特許文献２）のようなツールが登場して、様々なプロジェクトで利用されている。しかしホスト系での開発案件が多いＣＯＢＯＬについてはこういったツールが活用されておらず、未だ目視点検に係る上記課題を抱えている。

【０００５】
下記特許文献１では、コーディング規約に反するキーワードとソースコードを比較することにより、ソースコードがコーディング規約に沿っているか否かをチェックする技術が記載されている。

【０００６】
下記特許文献２では、実行頻度の高低によってソースコード内の各部をランク付けし、実質的にエラーとしてカウントすべきものとそうでないものを識別し易くする技術が記載されている。

【先行技術文献】
【特許文献】
【０００７】
【特許文献１】特開平１１－７３３２８号公報
【特許文献２】特開２００７－１７９４８８号公報
【非特許文献】
【０００８】
【非特許文献１】ＵＲＬ：http://findbugs.sourceforge.net/
【非特許文献２】ＵＲＬ：http://www.splint.org/
【発明の概要】
【発明が解決しようとする課題】
【０００９】
上記特許文献１に記載の技術では、キーワードに合致した部分がエラーとして検出されるため、プログラムの動作上は必ずしもエラーではない部分がエラーとして大量に検出される可能性を否定できない。
【００１０】
上記特許文献２に記載の技術では、実行頻度を基準としてエラー可能性を識別しているため、実行頻度は低いが明らかにエラーである部分が、チェック対象から外れてしまう可能性がある。
【００１１】
本発明は、上記のような課題を解決するためになされたものであり、ＣＯＢＯＬ言語の特性に適した、エラー検出精度の高いソースコードチェック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】
【００１２】
本発明に係るＣＯＢＯＬソースコードチェックプログラムは、ＣＯＢＯＬ言語のテーブルサイズを超えてそのテーブルにアクセスする処理がソースコード内に存在するか否かをチェックする。
【発明の効果】
【００１３】
本発明に係るＣＯＢＯＬソースコードチェックプログラムによれば、テーブルに対するサイズ制限に違反する不正アクセス処理を精度良く検出することができる。
【図面の簡単な説明】
【００１４】
【図１】実施の形態１に係るＣＯＢＯＬソースコードチェックプログラム１１０を実行するコンピュータ１００および関連するデータの入出力を示す図である。
【図２】テーブルを使用するＣＯＢＯＬソースコード２００の１例を示す。
【図３】図２に示すＣＯＢＯＬソースコード２００の構文木を示す図である。
【図４】複数項目を有するデータ変数を探索する様子を示す図である。
【図５】条件文を探索する様子を示す図である。
【図６】テーブル定義に合致する条件文を探索する様子を示す図である。
【図７】実施の形態２に係るＣＯＢＯＬソースコードチェックシステム１０００の構成図である。
【発明を実施するための形態】
【００１５】
＜実施の形態１＞
図１は、本実施の形態１に係るＣＯＢＯＬソースコードチェックプログラム１１０を実行するコンピュータ１００および関連するデータの入出力を示す図である。コンピュータ１００は、ＣＯＢＯＬプログラムのチェック作業を行うために用いるコンピュータである。

【００１６】
コンピュータ１００は、チェック対象であるＣＯＢＯＬソースコード２００と、チェック規則を定義するチェック規則データ３００を入力として受け取り、ＣＯＢＯＬソースコード２００のチェック結果を出力する。これらのデータは、必要に応じてコンピュータ１００が備える記憶装置に格納される。
【００１７】
コンピュータ１００は、ＣＯＢＯＬソースコードチェックプログラム１１０を格納するＨＤＤ（Ｈａｒｄ　Ｄｉｓｋ　Ｄｒｉｖｅ）などの記憶装置と、ＣＯＢＯＬソースコードチェックプログラム１１０を実行するＣＰＵ（Ｃｅｎｔｒａｌ　Ｐｒｏｃｅｓｓｉｎｇ　Ｕｎｉｔ）などの演算装置を備える。また、各データを入出力するインタフェース、メモリなどを適宜備える。以下では説明の便宜上、ＣＯＢＯＬソースコードチェックプログラム１１０を動作主体として説明する場合があるが、実際にＣＯＢＯＬソースコードチェックプログラム１１０を実行するのはＣＰＵなどの演算装置であることを付言しておく。
【００１８】
チェック規則データ３００は、ＣＯＢＯＬソースコードチェックプログラム１１０がＣＯＢＯＬソースコード２００をチェックする際の基準となるチェック規則を１以上定義する。例えば以下のようなチェック規則が例として挙げられる。
【００１９】
（チェック規則その１：テーブルの件数超過を避ける）
ＣＯＢＯＬ言語では、複数のデータを配列状に格納する変数の一種として、テーブルと呼ばれるものが用いられる。このテーブルは、はじめに最大サイズを指定しておき、その最大サイズ内でデータを保持する変数である。最大サイズを超えてテーブルにアクセスすることは、不正な処理として禁止される。
【００２０】
（チェック規則その２：ＥＮＤ－ＩＦを必ず記述する）
ＣＯＢＯＬ言語では、ＩＦステートメントに対応するＥＮＤ－ＩＦステートメントを記述しなくても、ピリオドによりＩＦステートメントの終端を記述することができる。しかし、ピリオドは見落としやすいため、ソースコードのメンテナンス担当者がソースコードを誤読したり、誤修正したりする要因となる。そこで本発明に係るＣＯＢＯＬソースコードチェックプログラム１１０は、ＩＦステートメントに対応するＥＮＤ－ＩＦステートメントが存在しているか否かをチェックすることとした。
【００２１】
ＣＯＢＯＬソースコードチェックプログラム１１０は、以下の手順でＣＯＢＯＬソースコード２００をチェックする。
【００２２】
（ＣＯＢＯＬソースコードチェック手順：ステップ１）
ＣＯＢＯＬソースコードチェックプログラム１１０は、ＣＯＢＯＬソースコード２００の構文を解析し、木構造のデータ（構文木）に変換して、メモリまたはＨＤＤなどの記憶装置上に記憶する。１つのＣＯＢＯＬソースコード２００に対して１つの構文木が生成される。構文木の各構成要素には、ＣＯＢＯＬソースコード２００を構成する構文要素が格納される。
【００２３】
（ＣＯＢＯＬソースコードチェック手順：ステップ２）
ＣＯＢＯＬソースコードチェックプログラム１１０は、ステップ１で生成した構文木の構成要素が保持しているＣＯＢＯＬソースコード２００の構文要素を走査する。ＣＯＢＯＬソースコードチェックプログラム１１０は、チェック規則データ３００が定義しているチェック規則とＣＯＢＯＬソースコード２００の構文要素を照らし合わせながら、ＣＯＢＯＬソースコード２００の記述エラーをチェックする。
【００２４】
（ＣＯＢＯＬソースコードチェック手順：ステップ２：補足）

本ステップでは、構文木の個々の構成要素に対してエラーチェックを行うのみならず、複数の構成要素の組み合わせが適切であるか否かなど、複数の構成要素に対してエラーチェックを行う場合もある。

【００２５】

（ＣＯＢＯＬソースコードチェック手順：ステップ３）

ＣＯＢＯＬソースコードチェックプログラム１１０は、ステップ２の結果を、例えばＣＳＶ（Ｃｏｍｍａ　Ｓｅｐａｒａｔｅｄ　Ｖａｌｕｅ）などの形式で出力する。

【００２６】

以下では、「チェック規則その１：テーブルの件数超過を避ける」を例として、ＣＯＢＯＬソースコードチェックプログラム１１０の具体的な動作を説明する。

【００２７】

図２は、テーブルを使用するＣＯＢＯＬソースコード２００の１例を示す。図２の４行目において、テーブル「ＴＡＢＬＥ０１」が、「ＯＣＣＵＲＳ」ステートメントで最大サイズ「１００」のテーブルとして定義されている。一方、８～９行目において、変数「ＣＯＵＮＴＥＲ」の値が２００になるまでテーブルにアクセスしており、テーブルの最大サイズを超過したアクセスが発生している。以下、ＣＯＢＯＬソースコードチェックプログラム１１０が図２のソースコードをチェックする過程を説明する。

【００２８】

（ステップ１：構文木を生成する）

図３は、図２に示すＣＯＢＯＬソースコード２００の構文木を示す図である。ここでは記載の都合上、「ＰＲＯＣＥＤＵＲＥ」ＤＩＶＩＳＩＯＮの構文木のみを示した。ＣＯＢＯＬソースコードチェックプログラム１１０は、ＣＯＢＯＬソースコード２００の構文要素と階層構造を解析し、図３に示すような構文木を生成する。構文木はメモリなどの一時記憶装置内にプログラム処理上のオブジェクトとして記憶してもよいし、構文木の内容を記述した構文木データをＨＤＤに格納してもよい。

【００２９】

（ステップ１：構文木を生成する：補足）

ＣＯＢＯＬソースコードチェックプログラム１１０は、ＣＯＢＯＬソースコード２００の構文要素をソースコードのまま構文木に格納するのではなく、必要に応じて構文要素をカテゴリ分けする。例えば、ＩＦステートメント、ＥＶＡＬＵＡＴＥステートメントなどの、条件文を持つステートメントに、「ＣＯＮＤＩＴＩＯＮ」という構成要素を付与し、その配下に実際の条件文を格納する。ここでいう条件文とは、例えば不等式や等式が成立するか否かを判定する際の条件式に相当する。

【００３０】

（ステップ２：複数項目を有するデータ変数を探索する）

図４は、複数項目を有するデータ変数を探索する様子を示す図である。ＣＯＢＯＬソースコードチェックプログラム１１０は、構文木を走査し、複数項目を有するデータ変数を探索する。具体的には、括弧（）を添えているデータ項目を探索する。また、（）内の添字もデータ項目とセットにして記録しておく。なお、データ項目を部分参照する構文要素は、本ステップの対象外とする。図４に示す例では、データ項目「ＤＡＴＡ０１」とその添字「ＣＯＵＮＴＥＲ」が本ステップの探索対称となる。

【００３１】

（ステップ２：複数項目を有するデータ変数を探索する：補足）

本ステップは、サイズ制限のあるデータ項目を探索し、サイズ超過アクセスの問題を引き起こす可能性のあるソースコードを探し出す起点とする意義がある。

【００３２】

（ステップ３：テーブル定義を探索する）

ＣＯＢＯＬソースコードチェックプログラム１１０は、ＣＯＢＯＬソースコード２００の「ＤＡＴＡ」ＤＩＶＩＳＩＯＮの構文木（図示せず）を走査し、ステップ２で探索したデータ項目（ここではＤＡＴＡ０１）の定義を探索する。

【００３３】
（ステップ４：テーブルサイズを取得する）
ＣＯＢＯＬソースコードチェックプログラム１１０は、ステップ３で探索したデータ項目自身の定義、またはデータ項目が属する集団項目に、「ＯＣＣＵＲＳ」ステートメントが宣言されているか否かをチェックする。「ＯＣＣＵＲＳ」ステートメントが宣言されている場合は、その数値とデータ項目名を記録する。図２に示したＣＯＢＯＬソースコード２００の例では、「ＤＡＴＡ０１」が所属する集団項目「ＴＡＢＬＥ０１」に「ＯＣＣＵＲＳ」ステートメントが宣言されているので、その数値「１００」と「ＤＡＴＡ０１」をセットにして記憶する。
【００３４】
（ステップ５：条件文を探索する）
図５は、条件文を探索する様子を示す図である。ＣＯＢＯＬソースコードチェックプログラム１１０は、ＣＯＢＯＬソースコード２００の「ＰＲＯＣＥＤＵＲＥ」ＤＩＶＩＳＩＯＮの構文木を、ステップ２で探索したデータ項目（ここではＤＡＴＡ０１）から遡って走査し、条件文を探索する。ステップ１で説明した通り、条件文を持つ構成要素は必ず「ＣＯＮＤＩＴＩＯＮ」構成要素を持つので、「ＣＯＮＤＩＴＩＯＮ」を親ノードから順に探索すればよい。
【００３５】
（ステップ５：条件文を探索する：補足）
本ステップは、データ項目の最大サイズを超えてそのデータ項目にアクセスする処理を引き起こす可能性のある条件文を洗い出す意義がある。
【００３６】
（ステップ６：テーブル定義に合致する条件文を探索する）
図６は、テーブル定義に合致する条件文を探索する様子を示す図である。ＣＯＢＯＬソースコードチェックプログラム１１０は、ステップ５で探索した条件文の構文木のうち、ステップ２で記録した（）内の添字に合致するものを探索する。図６に示した例では、ステップ２で記録した添字「ＣＯＵＮＴＥＲ」が、「ＣＯＮＤＩＴＩＯＮ」配下の「ＣＯＵＮＴＥＲ＞２００」として見つかる。
【００３７】
（ステップ７：テーブル定義と条件文を比較する）
ＣＯＢＯＬソースコードチェックプログラム１１０は、ステップ４で取得したテーブルサイズと、ステップ５で取得した条件文とを比較し、テーブルサイズを超過するアクセスが生じるか否かを検証する。ここでは、ステップ４で取得した「ＯＣＣＵＲＳ」ステートメントが指定するテーブルサイズが「１００」であるのに対し、ステップ５で取得した条件文は「２００」を指定している。したがって、ＣＯＢＯＬソースコードチェックプログラム１１０は、「ＣＯＵＮＴＥＲ＞２００」が記述エラーであるものとみなす。
【００３８】
以上、ＣＯＢＯＬソースコードチェックプログラム１１０が図２のソースコードをチェックする過程を説明した。
【００３９】
以上のように、本実施の形態１によれば、ＣＯＢＯＬソースコードチェックプログラム１１０は、テーブルにアクセスする処理をＣＯＢＯＬソースコード２００の構文木から探索し、テーブルサイズを超過するアクセス処理が存在するか否かを検証する。これにより、不正なテーブルアクセス処理を事前に検出することができる。
【００４０】
具体的には、ＣＯＢＯＬソースコードチェックプログラム１１０は、複数項目を有するデータ項目（上記例ではＤＡＴＡ０１）を起点として構文木を探索し、そのデータ項目の条件文（上記例ではＣＯＵＮＴＥＲ＞２００）とテーブルサイズ定義（上記例ではＯＣＣＵＲＳ　１００）を比較する。データ項目を起点として構文木を探索することにより、データ項目のサイズを基準として不正アクセスを検出することができるので、検出漏れを確

実に防ぐことができる。

【００４１】

＜実施の形態２＞

図７は、本発明の実施の形態２に係るＣＯＢＯＬソースコードチェックシステム１０００の構成図である。ＣＯＢＯＬソースコードチェックシステム１０００は、実施の形態１で説明したコンピュータ１００、ホスト４００、共有サーバ５００を有する。

【００４２】

ＣＯＢＯＬソースコードチェックプログラム１１０は、開発作業用コンピュータであるコンピュータ１００上で実行される一方、チェック対象であるＣＯＢＯＬソースコード２００は、ホスト（汎用機）４００上に置かれていることが多い。この場合、ソースコードをチェックするためには、ＣＯＢＯＬソースコード２００をホスト４００からコンピュータ１００にダウンロードする必要がある。

【００４３】

ところが、チェックを完全にするためには、ＣＯＢＯＬソースコード２００から参照している外部ソースコード（図７の参照先ソースコード２１０）を併せてダウンロードする必要がある。この参照先ソースコード２１０は、膨大な量になることが多く、ダウンロード時間もその分だけ多くかかる。したがって、チェック作業を開始するまでの準備作業に時間がかかってしまう。

【００４４】

そこで本実施の形態２では、ホスト４００とコンピュータ１００の間に共有サーバ５００を設置し、共有サーバ５００がホスト４００から参照先ソースコード２１０を定期的にダウンロードすることとした。

【００４５】

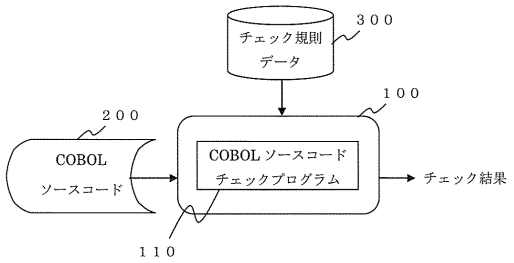
ホスト４００が開発作業を行う場所から離れた遠隔に設置されているような場合には、ホスト４００と共有サーバ５００の間のダウンロード処理は時間がかかるものの、共有サーバ５００とコンピュータ１００の間のダウンロードは、例えば共有サーバ５００を開発作業場所に設置しておけば、ごく短時間で済む。共有サーバ５００は、時間のかかるダウンロード処理を例えば夜中にあらかじめ行っておく。これにより、ＣＯＢＯＬソースコードチェックプログラム１１０が参照先ソースコード２１０を必要とする場合は、即座に必要な参照先ソースコード２１０を取得することができ、チェック作業にかかる時間を大幅に低減することができる。

【符号の説明】

【００４６】

１００：コンピュータ、１１０：ＣＯＢＯＬソースコードチェックプログラム、２００：ＣＯＢＯＬソースコード、２１０：参照先ソースコード、３００：チェック規則データ、４００：ホスト、５００：共有サーバ、１０００：ＣＯＢＯＬソースコードチェックシステム。

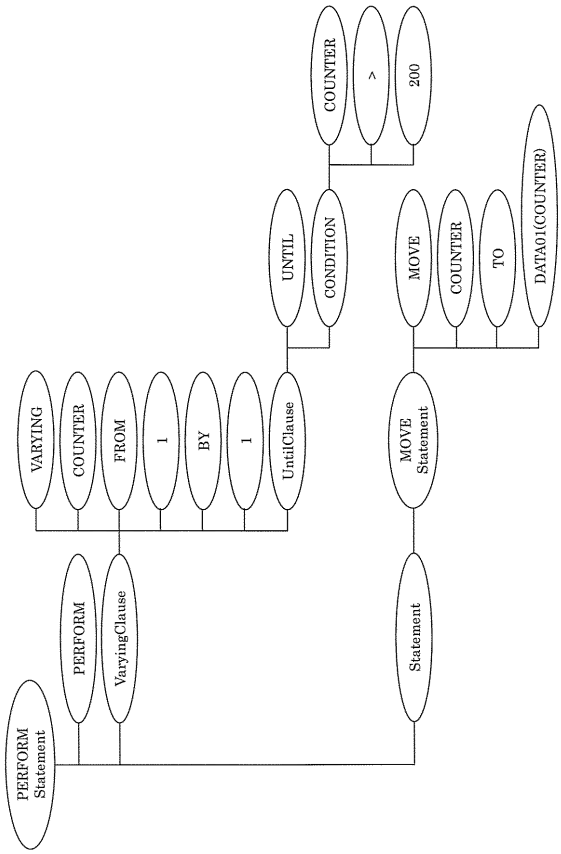
【図１】



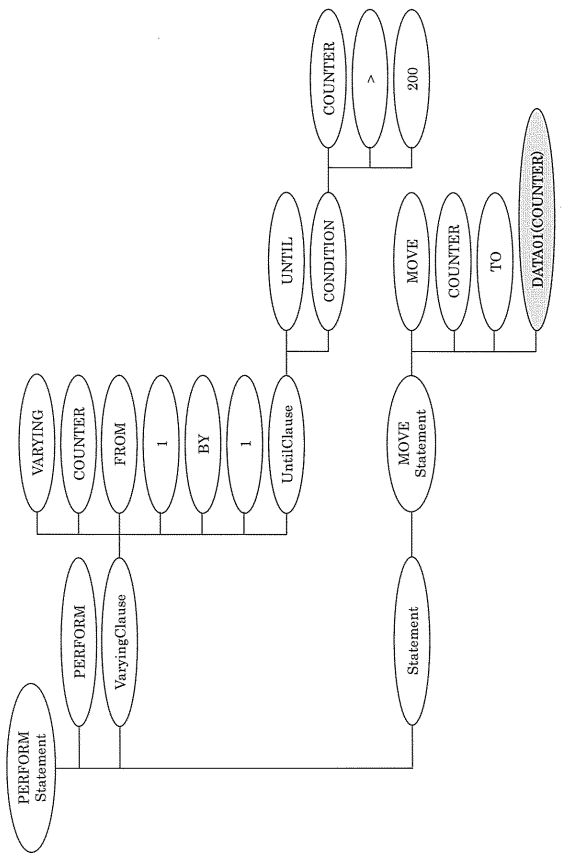
【図２】

```
 1: DATA                                    DIVISION.
 2: WORKING-STRAGE                          SECTION.
 3: 01 TABLE-AREA.
 4:     02  TABLE01  OCCURS 100.
 5:         03   DATA01    PIC X(05).
 6: PROCEDURE                               DIVISION.
 7:     PERFORM   VARYING  COUNTER  FROM  1  BY  1
 8:         UNTIL  COUNTER > 200
 9:             MOVE   COUNTER   TO   DATA01(COUNTER)
10:     END-PERFORM.
```

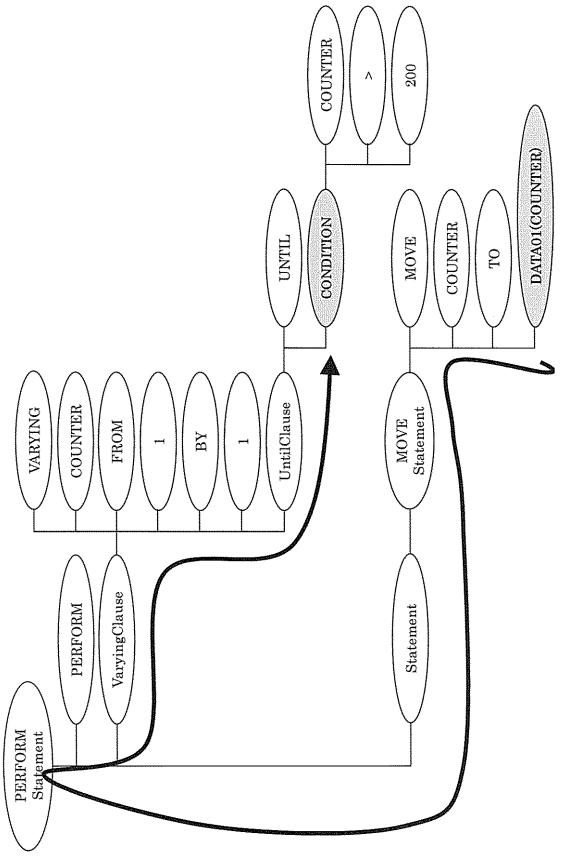
【図３】



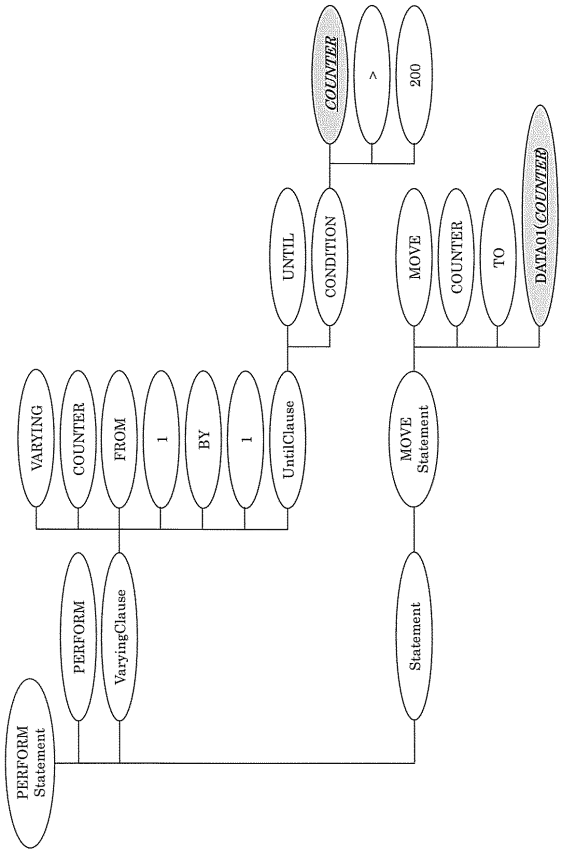
【図４】



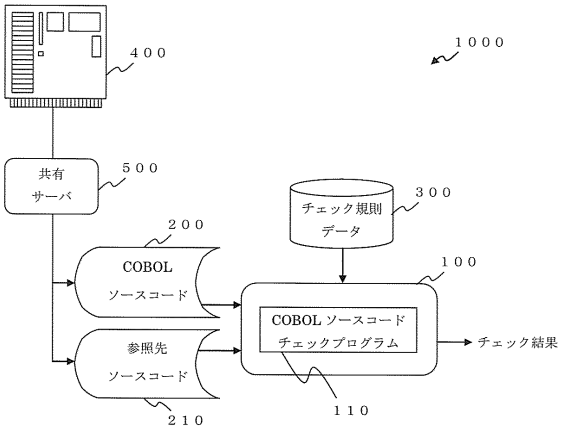
【図５】



【図６】


COUNTER
＜
200
UNTIL
CONDITION
MOVE
COUNTER
TO
DATA01(COUNTER)
VARYING
COUNTER
FROM
1
BY
1
UntilClause
MOVE Statement
PERFORM
VaryingClause
Statement
PERFORM Statement

【図７】


1000
400
共有サーバ
500
チェック規則データ
300
200
COBOL ソースコード
100
COBOL ソースコード チェックプログラム
チェック結果
参照先 ソースコード
210
110

フロントページの続き

(56)参考文献 特開２００５－１９０３３０（ＪＰ，Ａ）
特開平０７－１５２６０１（ＪＰ，Ａ）
特開２０００－３１１０８８（ＪＰ，Ａ）
特開２００７－０３４３５５（ＪＰ，Ａ）
特開平１１－０７３３２８（ＪＰ，Ａ）
特開２００７－１７９４８８（ＪＰ，Ａ）
武市　正人，構文解析を用いたCOBOLソースコード品質点検ツールの開発，exa review，日本，
株式会社　エクサ　exa review 編集室，２０１０年　４月　１日，No.10 2010.04，pp.15-22

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１１／３６
Ｇ０６Ｆ　９／４５