Vol.2017-SE-196 No.21
2017/7/20

C ソースコード静的解析のための問い合わせ言語 CXmlPyQuery とその応用

岩間恵梨沙[†1]　福原和哉[†2]　猪股俊光[†1]　杉野栄二[†1]　今井信太郎[†1]
新井義和[†1]　成田匡輝[†1]

概要：本研究では C 言語のソースコードを XML 形式で表現したファイルに対して，簡単かつ柔軟に要素検索を行うための問合わせ言語 CXmlPyQuery を設計し，実装した．この言語は，Python を基にした問い合わせ言語であり，検索した要素集合に対して各種集合演算を行うことにより，ソースコード中の特定のパターンを検出することができる．応用として，コーディング規約の検査への適用例を示す．

キーワード：静的解析，XML，問い合わせ言語

Query language CXmlPyQuery for static analysis of C source code and its application

ERISA IWAMA[†1]　KAZUYA FUKUHARA[†2]　YOSHIMITSU INOMATA[†1]
EIJI SUGINO[†1]　SHINTAROU IMAI[†1]　YOSHIKAZU ARAI[†1]
MASAKI NARITA[†1]

Abstract: This study was designed and implemented an inquiry language ”CXmlPyQuery” for easy and flexible element search for file expressing C language source code in XML language. XML language is Python-based inquiry language, it can detect a specific pattern in source code by performing various set operations on the retrieved element set. As an application, this study shows application example of bug pattern detection and coding regulation inspection.

Keywords: Static Analysis, XML, Query Language

1. はじめに

ソフトウェア開発では，製品出荷前のソフトウェア検査が品質向上の観点から非常に重要である．製品出荷前の検査としては，ソースコードがあらかじめ決められたコーディング規約に従っているかどうか，既知の不具合と同様なコードが含まれているかどうか，不具合の発生を防ぐための対策が施されているかどうかなどを，人間の目視，あるいは自動検査ツールの利用によって行うコードレビューが広く行われている．人間の目視による場合には，検査精度は実施者の経験や技術力に依存する．自動検査ツールを利用する場合には，検査実施者の能力差の影響をなくすることはできるものの，検査できる項目が使用ツールに依存することになる．検査項目の中には開発会社ごとに品質担保のための非公開な項目も存在することから，検査ツール開発を外部委託することも難しい．

このような問題に対して開発者が検査ツールを実現できるようにすることを目指した研究[1][2]が行われているが，開発者が製品開発ではほとんど利用していない言語（例えば関数型言語）が利用されていたり[1]，中間表現で用いられている特別なモデル[2]の理解が必要であったりと，製品開発者や検査担当者にとって利用上の課題が多い．

筆者らは，これまでに，ANSI-C で記述されたソフトウェアを対象としたソースコードの静的検査やコードレビューのためのツール開発ためのフレームワークを提案している．このフレームワークは，ソースコードを XML 形式で表し，XML で記述されたソースコードに対して各種解析手法を適用するアイディアに基づいており，そのために必要とされる中間表現モデル CXml と，ソースコードから CXml を生成するツール群を開発している[3][4].

本論文では，この CXml に対して以下の特徴を持った問い合わせを行う言語“CXmlPyQuery”を設計し，この問い合わせ言語を実行するためのツール“CXmlPyQuery Tool”を実装した．

- 制御構造やデータコンテナ型を利用できる既存のプログラミング言語（Python）で記述できる．
- 可読性が高く，習得が容易である．
- インタプリタでの実行が可能である．

†1 岩手県立大学大学院ソフトウェア情報学研究科
Graduate School of Software and Information Science, Iwate Prefectual University
†2 岩手県立大学共同研究員

2. ソースコード静的解析のための表現形式

2.1 解析の対象とする項目

本研究では，C 言語で記述されたソースコード（構文エラーを含んでいないもの）を対象とし，ソースコードの作成途中または作成後に行われるコードレビューにおいて，以下の項目を解析（検査）項目として想定した．

- 一般的なコーディング規約に準拠しているかどうか（例えば，MISRA-C:2004[5]）．
- 開発現場独自のコーディング規約に準拠しているかどうか（例えば，自作ライブラリの利用方法や独自の命名規則など）．
- 過去の不具合の要因となったコードが含まれているかどうか．

2.2 CXml モデル

本研究では，CXmlPyQuery の問い合わせ対象として CXml モデル（以下，CXml）を利用する．

CXml は CXml DTD で定義される XML 文書であり，ソースコードに対して以下に示す処理を施して C ソースコードの曖昧さを排除したうえで, XML 文書とすることとした．

- 演算の順序を明確化
- 識別子の参照と宣言を対応付け
- 識別子の型の同定
- 複合宣言の分割
- 匿名型の匿名解除
- 暗黙の宣言を除去
- 省略可能要素の省略を明確化

CXml は，XML 文書の特性を引き継いでおり，可読性，可搬性に優れている．また，各種 XML ライブラリでの読み取りが可能である．

図 1 のソースコードを CXml に変換したものが，図 2 である．図 2 の CXml のタグは全部で 62 個であり，要素数は 34 個である．

```
1  int sample(int num)
2  {
3      num = num * 10;
4      return num;
5  }
```

図 1　C 言語のサンプルコード
Figure 1　Sample code of C language

2.3 CXmlQuery

CXmlQuery は利用者による拡張が容易なコーディング規約検査ツールへの応用を主な目標としている言語および検査器[4]である．

CXmlQuery では, 検査マクロと呼ばれる問い合わせ言語を使用して，特定のコード断片を取り出すことができる．逐次的に問い合わせを実行して結果を確認しながら記述できる対話型と，問い合わせをすべて記述したファイルを入力して一括実行できるバッチ型の 2 種類の入力方式がある．

2.3.1 検査マクロの記述形式

検査マクロは，CXml で表現されたコードの中から検出したい対象を抽出するための演算命令列であり，集合演算と XPath 式を組み合わせて記述する．表 1 に検査マクロの記述形式を BNF 記法で表現したものを示す．

表 1　検査マクロの記述形式
Table 1　Description format of inspection macro

構　文	説　明
<問い合わせ式> ::= 'LET' <変数名> '=' <二項演算式> | <変数名> | 'EACH' <変数名> <実行式>	<変数名>で<式>の結果を束縛 | <変数名>で示される値を簡易的に表示 | <変数名>で示される値（集合）の各要素に<実行式>を適用
<二項演算式> ::= <基本要素式> | <基本要素式> '&' <二項演算式> | <基本要素式> '|' <二項演算式> | <基本要素式> '-' <二項演算式> | <基本要素式> '^' <二項演算式>	'&'は集合の積 | '|'は集合の和 | '-'は集合の差 | '^'は集合の対称差
<基本要素式> ::= ('NODE' | 'ATTR') <CXPath 式> <FROM 部>? | <変数名> | '(' <二項演算式> ')'	<FROM 部>で示される集合に <CXPath 式> を適用して結果（NODE であれば要素，ATTR であれば属性）を返す（<FROM 部>省略時は FROM ROOT と等価）
<FROM 部> ::= FROM <変数名> | FROM ROOT	変数名で示される集合 | CXml 文書木のルートノード
<実行式> ::= PRINT <書式文字列> | DO <問い合わせ式> END	書式文字列に従って要素の出力を行う | DO~END 中の問い合わせ式を行う
<書式文字列>	文字列だが{}で括られた箇所はその内容に応じて書き換えられる

CXml で表現されたコード中の要素等を検索するときには LET 文を用いる．LET 以降に“$”で始まる文字列を変数として，NODE（要素検索指定）もしくは ATTR（属性検

索指定）以降に XPath を記述することで要素集合もしくは属性集合が取得される．このとき，FROM <変数名> があれば，その変数で指定されている集合に対して XPath での検査が行われる．一方，FROM ROOT の場合は CXml の最初の要素から検索される．この FROM 以降は省略可能であり，省略した場合には FROM ROOT とみなされる．

変数どうしは次のような集合演算が可能である．

- 和集合　　$A | $B
- 積集合　　$A & $B
- 差集合　　$A - $B
- 対称差集合　$A ^ $B

変数に格納されている要素集合を表示するときは LET 文以降に変数名を入力すると，変数名の示す要素集合が CXml 形式で表示される．

変数に格納されている要素集合の属性を表示するときは EACH 文を用いて PRINT 以降に書式を定義し，属性値は{@属性名}で指定する．

ある変数に対して，さらに問い合わせを行いたい場合には EACH 文を用いて DO～END 中に問い合わせを記述する．

2.3.2 検査マクロの記述例

検査マクロの記述例を図 3 に示す．図 3 は図 1 のサンプルコードに対して，「変数と定数の掛け算」を行っている箇所（図 1 の 3 行目）を検索するための記述例である（行頭の“>”はプロンプトであり，対話型で記述されている）．

図 3 の 1 行目で“掛け算の算術二項演算式の左辺もしくは右辺に変数式が使用されている式文”を検索し，変数 $variable_exp に格納している．2 行目の文は変数 $variable_exp の内容を参照する文であり，結果として 3 行目の文が得られる．{}の中には@マーク以降に属性を指定することができ，この例では{@line}によって行番号が表示される．このように対話型では変数の内容を確認しながら進めていくことができる．

次に 4 行目で“掛け算の算術二項演算式の左辺もしくは右辺に定数式が使用されている式文”を検索し，変数 $constant_exp に格納している．5 行目の文は変数 $constant_exp の内容を参照する文であり，結果として 6 行目の文が得られる．

7 行目では 1 行目と 4 行目で検索した箇所の積集合によって，“掛け算の算術二項演算式の左辺もしくは右辺それぞれに変数式と定数式が使用されている式文”に絞る．そして，これらの結果は変数$result に格納される．8 行目は $result の内容を参照する文である．図 3 の 9 行目は，問い合わせの結果であり，図 1 の 3 行目が検索結果として表示されている．

2.4　検査マクロの課題

検査マクロは集合演算や構文要素の選択などを容易に記述するために設計したために，制御構文や配列型，リスト型などのデータ型は持たない．そのため，複雑な検査をすることができない．そこで複雑な検査を行うための直接的な検査方式として C#を用いた検査スクリプトでの記述を可能とした．しかし，検査スクリプトを用いた記述を行う際には LINQ[6]の知識が必要不可欠である．LINQ とは Language INtegrated Query の略であり，統合言語クエリとも呼ばれる．データベースや XML などのさまざまなデータ集合に対して，標準化された方法で問い合わせを行うことができる．LINQ を利用するには，ライブラリメソッドやラムダ式の理解が必須であるため，ある程度のトレーニングが必要である．また，C#はインタプリタ型の言語ではないため，対話型の入力はできない．

3. 問い合わせ言語 CXmlPyQuery

3.1 CXmlPyQuery の記述形式

2.4 節で述べた検査マクロの課題点を踏まえ，本研究では制御構造やデータコンテナ型を有するプログラミング言語 Python を拡張した問い合わせ言語 CXmlPyQuery を設計した．Python を基にしたのは，可読性が高い，習得に必要な知識も少ない，インタプリタでの記述が可能，対話型での入力が可能などの特徴を有するからである．

問い合わせ言語 CXmlPyQuery では，関数を連続して実行することによって要素を絞る．表 2 に CXmlPyQuery の関数の一部を列挙する．

表 2　CXmlPyQuery の関数の一部
Table 2　Part of the function of CXmlPyQuery

関　数	機　能
node_def_variable(“*ID*”, “*変数名*”, “*型*”)	要素の“*ID*”,検索したい“*変数名*”,検索したい変数の“*型*”をそれぞれ指定して変数定義を検索する
node_def_function(“*ID*”, “*関数名*”, “*型*”)	要素の“*ID*”,検索したい“*関数名*”,検索したい関数の“*型*”をそれぞれ指定して関数定義を検索する
node_if(“*ID*”, “*対象子要素*”)	要素の“*ID*”を指定して If 文を検索する“*対象子要素*”が指定されている場合はその子要素をさらに検索する
node_while(“*ID*”, “*対象子要素*”)	要素の“*ID*”を指定して while 文を検索する“*対象子要素*”が指定されている場合はその子要素をさらに検索する
node_binary_exp(“*ID*”, “*演算子*”, “*対象子要素*”)	要素の“*ID*”と算術二項演算式の“*演算子*”を指定して算術二項演算式を検索す

	る “*対象子要素*”が指定されている場合はその子要素をさらに検索する
node_variable_exp(“*ID*”, “*対象子要素*”)	要素の“*ID*”を指定して変数式を検索する “*対象子要素*”が指定されている場合はその子要素をさらに検索する
node_constant_exp(“ID”, “*定数型*”, “*定数値*”)	要素の“*ID*”を定数の“*定数型*”，定数の“*定数値*”を指定して定数式を検索する
print_node(*変数*)	変数で指定した集合の要素を表示する
attr(“*属性名*”)	“*属性名*”で指定した属性を取得する
child(“*要素名*”)	指定した“*要素名*”の子要素を取得する
sibling(“*要素名*”, “*前後*”)	指定した“*要素名*”の兄弟要素を“*前後*”の指定によって前もしくは後から取得する

CXmlPyQuery の関数の種類は大きく分けて 5 つある．

(1) 定義や宣言を検索する関数

node_def_variable 関数や node_def_function 関数などは，要素の ID のほか，変数や関数の名前，型を引数として指定することによって条件を絞り込む．

(2) 制御構文を検索する関数

node_while 関数や node_if 関数などは，要素の ID のほか，要素の子要素（while 文であれば条件式，if 文であれば then 文，else 文，条件式）を指定して，各子要素の中身を検索，取得することも可能となっている関数もある．

(3) 式文を検索する関数

node_constant_exp 関数では定数の型や定数値を引数として指定することで条件を絞り込める．

(4) 要素の属性を取得する関数

attr 関数は要素検索を行う関数の後に続けて呼び出すことで要素の属性の取得ができる．

(5) 子要素や兄弟要素などを取得する関数

child 関数や sibling 関数は，要素検索を行う関数の後に続けて呼び出すことで，直前までで検索された要素からみた子要素や兄弟要素などを取得する関数である．要素名を指定することで，子要素や兄弟要素の集合から要素名が一致するものだけを取り出せる．

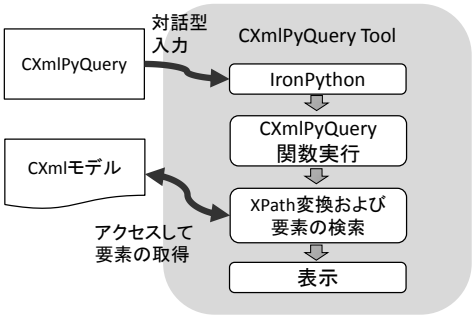

図 5　CXmlPyQuery Tool の構成
Figure 5　Structure of CXmlPyQuery Tool

CXmlPyQuery の関数により求められた要素集合は，CXmlQuery と同様に変数に格納することができ，集合演算を行うこともできる．

3.2 CXmlPyQuery の記述例

CXmlPyQuery の記述例を図 4 に示す．2.3.2 の検査マクロの記述例（図 3）と同様に，図 1 のサンプルコードにおいて「変数と定数の掛け算」を行っている箇所（図 1 の 3 行目）を検索するための記述例である．図 4 の 1 行目で“算術二項演算式の左辺式および右辺式で変数式を使用している式文”を検索し，変数 variable に格納する．このとき左辺式および右辺式は算術二項演算式の子要素なので child 関数を用いて指定している．2 行目では“算術二項演算式の左辺式および右辺式で定数式を使用している式文”を検索し，変数 constant に格納する．1 行目と同じく左辺式および右辺式は child 関数で指定している．3 行目では変数 variable と変数 constant の積集合を取ることで，“算術二項演算式の左辺式および右辺式どちらかに変数式と定数式が使用されている式文”を絞り込み，変数 result に格納する．4 行目の print_node 関数によって，5 行目の変数 result の要素を表示される．

図 3 の検査マクロによる記述と比べて，CXmlPyQuery による記述の方が簡潔に書くことができている．

3.3 CXmlPyQuery Tool の構成

問い合わせ言語 CXmlPyQuery を実行するための GUI アプリケーションとして CXmlPyQuery Tool を実装した．

CXmlPyQuery Tool の構成を図 5 に示す．CXmlPyQuery Tool は対話型で入力された問い合わせを IronPython[7]で実行する．IronPython は .NET Framework 上で動作する Python であり，Python 2 系と互換性がある．

IronPython は C#で実装されており，CXmlPyQuery で用意した各関数（表 2 参照）も同様に C#で実装した．各関数（print_node 関数と attr 関数を除く）は内部で XPath に変換され，C#の XML ライブラリで要素検索が行われる．そのため，XPath への変換以降の検索手法は CXmlQuery の検索手法と同じである．

CXmlPyQuery Tool の GUI 画面を図 6 に示す．図 6 の 1 番上のテキストボックスには検索対象の CXml ファイルの

パスを指定する．パス指定の際には右横の「ファイルを開いて指定」ボタンを押下することでオープンファイルダイアログからパスを指定することができる．中央の黒いテキストボックスはコンソール出力用であり，ここに検索結果やエラー文などが表示される．最下のテキストボックスは問い合わせ入力用であり，ここに問い合わせを記述する．図 4 や図 8 の問い合わせを記述する場合には各図の“>”を外し，1 行ずつ，または数行まとめて入力することができる．基本的に `for` 文や `if` 文などの制御構造文はまとめて入力する．問い合わせを行うときは，右隣の「Run」ボタンを押すことで実行される．

図 6　CXmlPyQuery Tool の GUI 画面
Figure 6　GUI screen of CXmlPyQuery Tool

4. C コード解析の適用例

4.1 MISRA-C の記述

CXmlPyQuery の応用例として MISRA-C 2004 の中のコーディング規約のうちルール 14.7 の「関数では，関数の最後に唯一の出口が無くてはならない.」を適用した．ルールに違反する箇所を含むサンプルコードを図 7 に示す．

```
1  int func(void)
2  {
3     int x = 0;
4      /* 省略 */
5     return x;
6  }
7  int func2(void)
8  {
9      int x = 0, y = 0;
10     /* 省略 */
11     if(x+y > 10){
12         return x; /* 違反 */
13     }
14     return y;
15 }
```

図 7　MISRAC-2004 ルール 14.7 のサンプルコード
Figure 7　GUI screen of CXmlPyQuery Tool

図 7 の 12 行目と 14 行目に `return` 文があり，ルール 14.7 に違反している．このことを検査するための問い合わせ記述例を図 8 に示す．

1 行目で関数定義をすべて取得し，変数 `funcs` に格納している．2 行目以降は変数 `funcs` の要素を 1 つずつ取り出し，変数 `func` に格納し，1 つの関数の中から検索を行う．

3 行目で関数の直下に存在する要素のうち 1 番最後の要素を検索し，変数 `func_last_child` に格納する．もし，この要素が `return` 文でなかった場合，`return` 文は関数の最後として記述されていないので，違反として出力する（4〜8 行目）．7 行目では関数内にあるすべての `return` 文を取得しており，関数直下の最後に `return` 文がない場合には，それ以外の場所にある `return` 文をすべて違反として出力する．

9 行目では，ある `return` 文よりも前の箇所に存在する他の `return` 文を検索し、変数 `ret_befor` に格納する．もし，要素が見つかった場合，関数中に `return` 文は唯一ではないため，違反として出力する（10〜13 行目）．

CXmlPyQuery Tool の出力結果が，図 8 の 14 行目から 15 行目にかけてであり，ルール 14.7 に違反している `return` 文が 12 行目に存在していることが正しく検査されている．

5. おわりに

本論文では、CXml モデルに対する問い合わせ言語 CXmlPyQuery の設計と実装について述べた．CXmlPyQuery は，既存の問い合わせ言語 CXmlQuery における検査マクロと検査スクリプトの問題を解消する目的で開発している．

MISRA-C 2004 のルール 14.7 を例に，CXmlPyQuery により検査できることを示した．

今後は，可読性や記述の容易さに重点を置き，より少ない記述で検査が行えるような関数や仕組みを考案する．また，CXmlPyQuery の記述能力の評価のために，MISRA-C 2004 のルールのうち，記述・検査が可能な項目を調査する．

参考文献

[1] ”Frama-c”. https://frama-c.com/, (参照 2017-6-11).
[2] ” Sapid Home Page (in Japanese) - CX-Checker”. http://www.sapid.org/index-ja.html, (参照 2017-6-11).
[3] 高橋耶真人，福原和哉，猪股俊光，新井義和，今井信太郎. パターン照合を用いた対話型静的検査ツールの開発．電子情報通信学会ソサイエティ大会, A-9-1 (2012).
[4] 福原和哉，高橋耶真人，猪股俊光，新井義和，今井信太郎. 組込みソフトウェア向けコーディング規約チェッカのためのカスタマイズの一方式. FIT2012 第 11 回情報科学技術フォーラム, c-024 (2012).
[5] MISRA-C 研究会．組込み開発者におくる MISRA-C:2004-C 言語利用の高信頼化ガイド-．日本規格協会, 2006.
[6] “統合言語クエリ (LINQ) | Microsoft Docs”. https://docs.microsoft.com/ja-jp/dotnet/csharp/linq/, (参照 2017-06-11).
[7] “IronPython.net”. http://ironpython.net/, (参照 2017-06-11).

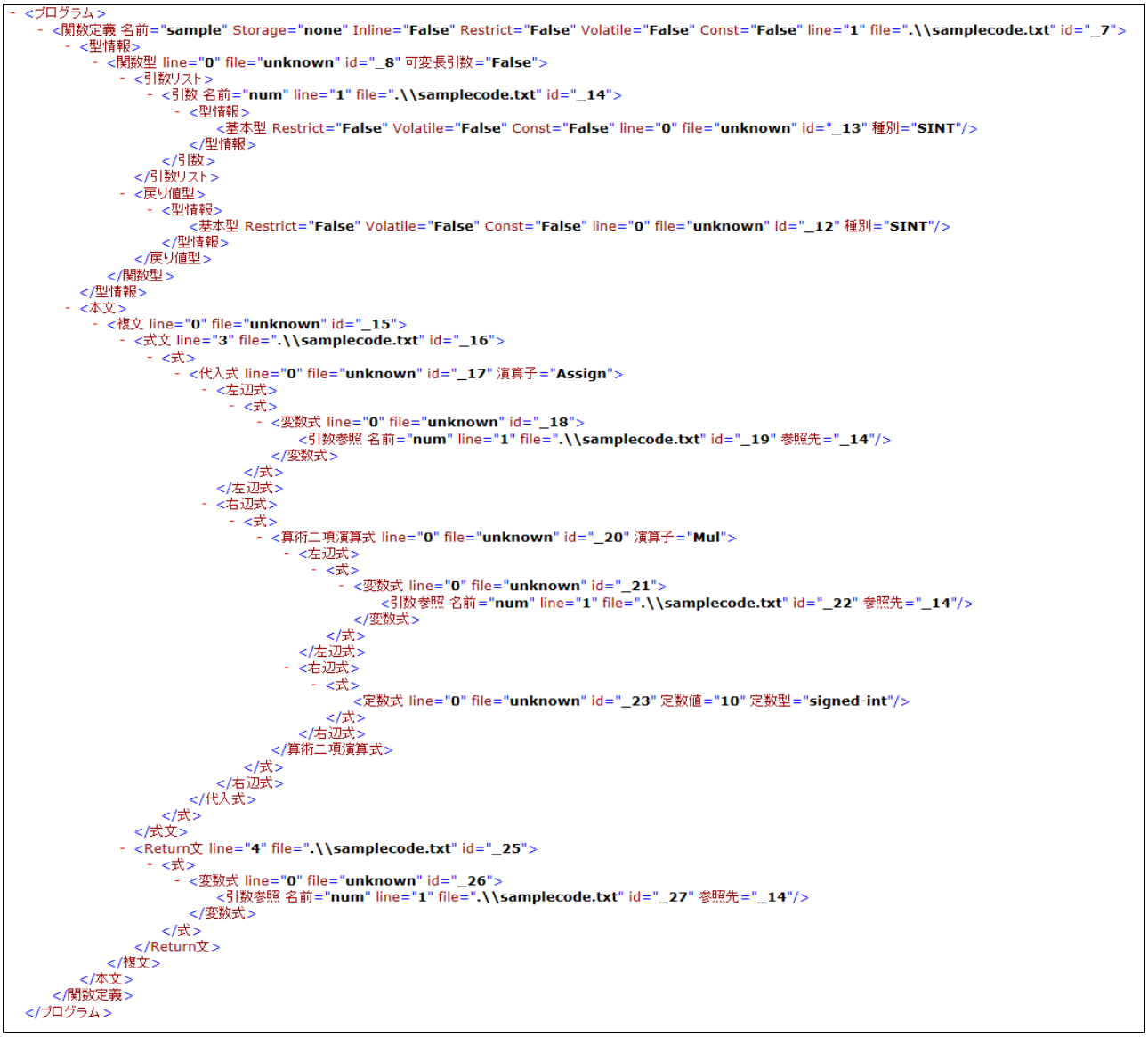

図 2　サンプルコードの CXml モデル

Figure 2　CXml model of sample code

```
1  > LET $variable_exp = NODE"//算術二項演算式[@演算式='Mul']/左辺式//変数式[first-ancestor('式
   文')]" | NODE"//算術二項演算式[@演算式='Mul']/右辺式//変数式[first-ancestor('式文')]"
2  > EACH $variable_exp PRINT"変数式を使った掛け算を行っている箇所は{@line}行目"
3  変数式を使った掛け算を行っている箇所は3行目
4  > LET $constant_exp = NODE"//算術二項演算式[@演算式='Mul']/左辺式//定数式[first-ancestor('式
   文')]" | NODE"//算術二項演算式[@演算式='Mul']/右辺式//定数式[first-ancestor('式文')]"
5  > EACH $constant_exp PRINT"定数式式を使った掛け算を行っている箇所は{@line}行目"
6  定数式を使った掛け算を行っている箇所は3行目
7  > LET $result = $variable_exp $ constant_exp
8  > EACH $result PRINT"変数式と定数式の掛け算を行っている箇所は{@line}行目"
9  変数式と定数式の掛け算を行っている箇所は3行目
```

図 3　図 1 のサンプルコードに対する検査マクロ記述例

Figure 3　Example of description of inspection macro for sample code of figure 3

```
1  > variable = node_binary_exp("", "Mul", "")
   .child("").node_variable_exp("","").first_ancestor("式文")
2  > constant = node_binary_exp("","Mul", "")
   .child("").node_constant_exp("",""). first_ancestor("式文")
3  > result = variable & constant
4  > print_node(result)
5  <式文 line="3" file=".¥¥samplecode.txt" id="_16">
```

図 4　図 3 のサンプルコードに対する CXmlPyQuery 記述例

Figure 4　Example of description of CXmlPyQuery for sample code of figure 3

```
1  > funcs = node_def_function("","","")
2  > for func in funcs:
3  >  func_last_child = func.node_under_func("","","").last()
4  >  if str(func_last_child).find('Return文') < 0:
5  >     print "MISRA-C 2004 ルール14.7 違反です"
6  >     print "関数の最後にReturn文がありません"
7  >     ret_all = func.node_return("")
8  >     print_node(ret_all)
9  >  ret_befor = node_return("").preceding("Return文")
10 >  if str(ret_befor) != "":
11 >     print "MISRA-C 2004 ルール14.7 違反です"
12 >     print "Return文が唯一でありません"
13 >     print_node(ret_befor)
14 MISRA-C 2004 ルール14.7 違反です
15 <Return文 id="_52" file=".¥¥samplecode4.txt" line="12">
```

図 8　図 7 のサンプルコードに対する CXmlPyQuery 記述例

Figure 8　Example of description of CXmlPyQuery for sample code of figure 7