Towards Higher-Order Syntax of C Programming Language

発表者: kinaba
Towards Higher-Order Syntax of C Programming Language

Kazuhiro Inaba
kMonos.net
kiki@kmonos.net

Kazuhiro Inaba
Google Inc.
kinaba@google.com

Kazuhiro Inaba
Free Researcher
binhzkr@gmail.com

ABSTRACT
The grammar of the C programming language [1] is widely known to go beyond the expressive power of context free grammar. The main, and in fact, the sole cause of the context sensitivity is that the indistinguishability of type names and variable names.

The natural question arises here is, then what kind of grammar formalism can capture such kind of name declarations? As a candidate, we seek a way to describe the C syntax using higher-order grammars, which have much more power than context free grammar, yet still have clearly understandable formal definition. Unfortunately, the paper reports the failure of this attempt, but at the same time, explain what addition is needed for higher-order grammar to achieve the goal: expressing C syntax.

Categories and Subject Descriptors
D.3.3 [Programming Languages]: Language Constructs and Features - abstract data types, polymorphism, control structures.

General Terms
Algorithms, Languages, Theory, Verification.

Keywords
Higher-Order Grammar, Parsing

1. INTRODUCTION
The grammar of the C programming language [1] is widely known to go beyond the expressive power of context free grammar. The main, and in fact, the sole cause of the context sensitivity is that the indistinguishability of type names and variable names.

The natural question arises here is, then what kind of grammar formalism can capture such kind of name declarations? As a candidate, we seek a way to describe the C syntax using higher-order grammars, which have much more power than context free grammar, yet still have clearly understandable formal definition. Unfortunately, the paper reports the failure of this attempt, but at the same time, explain what addition is needed for higher-order grammar to achieve the goal: expressing C syntax.

The proceedings are the records of the conference. ACM hopes to give these conference by-products a single, high-quality appearance. To do this, we ask that authors follow some simple guidelines. In essence, we ask you to make your paper look exactly like this document. The easiest way to do this is simply to download a template from [2], and replace the content with your own material.

2. MOTIVATING EXAMPLE
All material on each page should fit within a rectangle of 18 × 23.5 cm (7 × 9.25”), centered on the page, beginning 1.9 cm (0.75”) from the top of the page and ending with 2.54 cm (1”) from the bottom. The right and left margins should be 1.9 cm (0.75”). The text should be in two 8.45 cm (3.33”) columns with a .83 cm (.33”) gutter.

3. REFERENCES


Towards Higher-Order Syntax of C Programming Language

Kazuhiro Inaba
kmonos.net
kiki@kmonos.net

Kazuhiro Inaba
Google Inc.
kinaba@google.com

Kazuhiro Inaba
Free Researcher
binh2kr@gmail.com

ABSTRACT
The grammar of the C programming language [1] is widely known to go beyond the expressive power of context-free grammar. The main, and in fact, the sole cause of the context sensitivity is that the indistinguishability of type names and variable names.

The natural question arises here is, then what kind of grammar formalism can capture such kind of name declarations? As a candidate, we seek a way to describe the C syntax using higher-order grammars, which have much more power than context-free grammars, yet still have clearly understandable formal definition.

Unfortunately, the paper reports the failure of this attempt, but at the same time, explain what addition is needed for higher-order grammar to achieve the goal: expressing C syntax.

Categories and Subject Descriptors
D.3.3 [Programming Languages]: Language Constructs and Features – abstract data types, polymorphism, control structures.

General Terms
Algorithms, Languages, Theory, Verification.

Keywords
Higher-Order Grammar, Parsing

1. INTRODUCTION
The grammar of the C programming language [1] is widely known to go beyond the expressive power of context-free grammar. The main, and in fact, the sole cause of the context sensitivity is that the indistinguishability of type names and variable names.

The natural question arises here is, then what kind of grammar formalism can capture such kind of name declarations? As a candidate, we seek a way to describe the C syntax using higher-order grammars, which have much more power than context-free grammar, yet still have clearly understandable formal definition.

Unfortunately, the paper reports the failure of this attempt, but at the same time, explain what addition is needed for higher-order grammar to achieve the goal: expressing C syntax.

The proceedings are the records of the conference. ACM hopes to give these conference by-products a single, high-quality appearance. To do this, we ask that authors follow some simple guidelines. In essence, we ask you to make your paper look exactly like this document. The easiest way to do this is simply to download a template from [2], and replace the content with your own material.

2. MOTIVATING EXAMPLE
All material on each page should fit within a rectangle of 18 × 23.5 cm (7 × 9.25″), centered on the page, beginning 1.9 cm (0.75″) from the top of the page and ending with 2.54 cm (1″) from the bottom. The right and left margins should be 1.9 cm (0.75″). The text should be in two 8.45 cm (3.33″) columns with a 83 cm (33″) gutter.

3. REFERENCES
1. Introduction

Towards
Higher-Order Syntax of
C Programming Language
なぜか。

Cの文法は文脈自由ではないと言われる
なぜか。
なぜか。
Cの文法は文脈自由ではないと言われる
なぜか。

```
FILE *fp;
width * height;
n = (size_t)*ptr;
f = (m) * a;
strdup(*argv)[9];
byte (*bufs)[9];
puts(str);
clock_t (t);
```
すごくどうでもいいんですけど、いろいろ面白いですね。

クイズ：\( a*b*c \) に括弧を入れて宣言文をつくることはできるか？
Cの文法は文脈自由ではないと言われる

根本原因は、型名と変数/関数名が文脈情報を持ってないと区別できないこと。

cf.

gotoのラベル名
structやenumの名前
は文法的に区別できる。
Cの文法は文脈自由ではないと言われる

文脈自由じゃないなら、BNFより強力な文法記述言語で書けばいいじゃない！
Cの文法は文脈自由ではないと言われる

文脈自由じゃないなら、BNFより強力な文法記述言語で書けばいいじゃない！

ISO/IEC 9899:1999 自然言語で書いてある

gcc, clang, … : C や C++ で書いてある

CompCert (Coqで証明されたCコンパイラ) : パーサだけ OCaml で書いてあった

Elsa/Elkhound C++ Parser : GLR構文解析で頑張るが結局最後はC++。
Cの文法は文脈自由ではないと言われる

文脈自由じゃないなら、BNFより強力な文法記述言語で書けばいいじゃない！

ISO/IEC 9899:1999 自然言語で書いてある

gcc, clang, ... : C や C++ で書いてある

CompCert (Coqで証明されたCコンパイラ) : パーサだけ OCaml で書いてあった

Elsa/Elkhound C++ Parser : GLR構文解析で頑張るが結局最後はC++。

なぜ揃いも揃ってチューリング完全な言語を使うのか。根性が足りない。
チューリング完全脳の恐怖

∞: チューリングマシン

2: 文脈自由文法

1: 正規表現
現実を見よう

1: 正規表現
2: 文脈自由文法
添字文法
多重CFG
木結合文法
CbVマクロ文法
範囲結合文法
And文法
Boolean文法
帰納的木変形文法
拡張表0Lシステム
文脈依存文法
再帰文法
停止証明付き計算
∞: チューリングマシン
PEG

他色々

現実を見よう
拡張表

システム

他色々

1: 正規表現

2: 文脈自由文法

高階文法

Quoteマクロ文法

添字文法

多重CFG

木結合文法

範囲結合文法

帰納的木変形文法

多重

文脈依存文法

帰納的木変形文法

拡張表0Lシステム

Boolean文法

And文法

現実を見よう

この辺の中間的な表現力を持つ言語を使ってC言語の文法を記述してみたい
この辺なら書けるはずだけど
人類には難しい
この辺では無理そうな気がなんとなくする

- 添字文法
- CbVマクロ文法
- 多重CFG
- 範囲結合文法
- 木結合文法

2: 文脈自由文法

PEG

0Lシステム

1: 正規表現
この辺で頑張ってみる

高階文法

Quoteマクロ文法

帰納的木変形文法
Q. その前に、なぜ強力すぎる文法記述言語ではいけないのか

A. ツールで扱えない。

BNFで書いてあればそのままほぼコピーで yacc (など) に持って行ける。
IDE, エディタ, 静的解析器, lint, 構造化diff, 色分けマークアップツール等のバーサに即座に転用できる。

A. 他言語に自動的に持って行くのが難しい。

Eclipse上の開発環境作りたいのにC++で書いたコンパイラが仕様書しかない等々。

A. 抽象性・非決定性がないことが多い

どこまでが仕様でどこまでが実装の都合なのか境界が曖昧。

A. 安全でない

コードジェネレータの吐くコードが意図した通りにparseされることの証明をどう書けば良いのか？どう検証すればよいのか？

A. 文法の複雑さをなんとなくの感覚でしか議論できない。

現状「Regular」「LALR」「CF」「その他」の4分類しか無いに等しいが、実際はほとんどが「その他」なので、現実の言語はほとんど何も分類されていない。
そういうわけで、この辺で頑張ってみる

高階文法

帰納的木変形文法

Quoteマクロ文法
Towards Higher-Order Syntax of C Programming Language
Macro Grammar [Fischer 68] とは

パラメタ付き文脈自由文法

(↓はインデントの深さでブロックを表現する文法簡易バージョン)

```
BLOCK(x)  ::=  STATEMENT(x) | STATEMENT(x) BLOCK(x)
STATEMENT(x) ::=  S_IF(x) | S_WHILE(x) | S_EXPR(x)
S_IF(x)    ::=  x if  EXPR  ≪n BLOCK(x ≪t)
S_WHILE(x) ::=  x while EXPR  ≪n BLOCK(x ≪t)
S_EXPR(x)  ::=  x E
E          ::=  E + E | E * E | ...  
PROGRAM    ::=  BLOCK()
```
Macro Grammar [Fischer 68] とは

Fischer さんが提案したのは 3 種類

OI Macro Grammar : パラメタを call-by-name で遅延評価する
IO Macro Grammar : パラメタを call-by-value で正格評価する

Quoted Macro Grammar : quote と unquote でどちらの評価モードか明示できる

\[
\begin{align*}
\text{AORB} & \quad ::= \ a \mid b \\
\text{PAIR}(x) & \quad ::= \ < \ x \ , \ x \ > \\
S & \quad ::= \ \text{PAIR}(\ \text{AORB} )
\end{align*}
\]
Higher-Order Grammar

Context Free Grammar

各規則の型は :: string

Macro Grammar

各規則は、文字列を返す式を引数としてとって、文字列を返す :: string⇒string

Higher-Order Grammar

((string ⇒ string) ⇒ string) ⇒ string ⇒ string みたいな高階規則を考えよう

```
TYPE1(h, b) ::= h { b }
TYPE2(h, b) ::= h begin b end
SS(t) ::= | S(t) SS(t)
S(t) ::= IF(t) | WHILE(t) | UBE
IF(t) ::= t(if E, SS(t))
WHILE(t) ::= t(while E, SS(t))
UBE ::= beginend S(TYPE2)
PROGRAM ::= SS(TYPE1)
```
3. Formalizing C Syntax
Towards Higher-Order Syntax of C Programming Language
期待

パラメタ渡しを使って、「今宣言された変数の名前」とか「型の名前」とか持ち回っていけば文脈情報が表現できそうな気がする！

それだけで難しくても、なんかハイヤーオーダーパワーでどうにかなりそうな気がする！
やってみる
（優先順位とか複雑な型とか無視して簡略化）
「今使える型名と変数名」をパラメタに持たせる

EXPR ::= ( TYPE ) EXPR |
      * EXPR |
      EXPR * EXPR |
      VAR |
      ( EXPR )

EXPR(t, v) ::= ( t ) EXPR(t, v) |
              * EXPR(t, v) |
              EXPR * EXPR(t, v) |
              v |
              ( EXPR(t, v) )
やってみる

わりとよい

EXPR( size_t, ptr | m | a ) は曖昧性無く以下を区別する。
引数は call-by-name で解釈

```c
n = (size_t)*ptr;
f = (m) * a;
```

キャスト式 かけ算
宣言の扱い
（いろいろ無視して簡略化）

BlockItems ::= (S | D)
   | (S | D) BlockItems
D ::= typedef TYPE ID ;
   | TYPE VAR ;
S ::= EXPR ;

BlockItems(t, v) ::= (S | D)(t, v, Empty)
   | (S | D)(t, v, BlockItems)
D(t, v, k) ::= TDecl(ID, t, v, k)
   | VDecl(ID, t, v, k)
S(t, v, k) ::= EXPR(t, v) ; k(t, v)
where Empty(t, v) ::= 
   TDecl(tp, t, v, k) ::= typedef t tp ; k(t|tp, v)
   VDecl(x, t, v, k) ::= t x ; k(t, v|x)
宣言の扱い

変数宣言は、

\[
VDecl(x, t, v, k) ::= t x ; k(t, v|x)
\]

宣言される変数名（cbv）と、今既にある型名（cbn）、変数名（cbn）を受け取って変数宣言 “t x;” を生成して、更新された名前環境を継続 k に渡す。

型宣言も同様。

\[
TDecl(tp, t, v, k) ::= typedef t tp ; k(t|tp, v)
\]

注目ポイント：
cbv と cbn 引数、どっちも必要そう。
継続渡し形式なのは、趣味で高階関数使ったかっただけで、別に要らないと思います。
宣言の扱い

BlockItems(t, v) ::= (S | D)(t, v, Empty)
    | (S | D)(t, v, BlockItems)
D(t, v, k) ::= TDecl(ID, t, v, k)
    | VDecl(ID, t, v, k)
S(t, v, k) ::= EXPR(t, v) ; k(t, v)
where Empty(t, v) ::= 
    TDecl(tp, t, v, k) ::= typedef t tp ; k(t|tp, v)
    VDecl(x, t, v, k) ::= t x ; k(t, v|x)

BlockItems(LPCSTR, {}) が以下を一通りにparseすることが確かめられる

typedef LPCSTR x;
x y;
(x)*y;
Towards Higher-Order Syntax of C Programming Language

4. Conflicting Names
問題点

これはエラー

```c
int main() {
    typedef int x;
    int x;
}
```
問題点

・inner scope での定義と outer scope での定義は区別する必要がある
・inner scope での衝突はエラーにする必要がある
・outer scope との衝突は、外の scope を「隠す」必要がある
  ➔ “int x;” より後で x を型名として使うコードは構文エラーにする必要あり
高階文法パワーでどうにかなるか？
高階文法パワーでどうにかなるか？

・inner scope での定義と outer scope での定義は区別する必要がある
  ＞ どうにでもなる

・inner scope での衝突はエラーにする必要がある
  ＞ 無理

・outer scope との衝突は、外の scope を「隠す」必要がある
  ＞ 無理
5. Future Work

Towards
Higher-Order Syntax of
C Programming Language
結論

・inner scope での衝突はエラーにする必要がある
  > 無理

・outer scope との衝突は、外の scope を「隠す」必要がある
  > 無理

高階文法でもまだ無理なことがわかりました。
では、文法へのさらなる拡張として、何が必要か？
提案

nondeterminism

Foo にマッチするものか Bar にマッチするものか、どっちか

Foo | Bar

CFG やその上の高階文法は、非決定性の足し算しかできない

anti-nondeterminism

Foo にマッチするもののうち Bar にマッチしないもの。引き算が必要。

Foo - Bar
提案

・inner scope での衝突はエラーにする必要がある
  ＞ 引き算あればできる

・outer scope との衝突は、外の scope を「隠す」必要がある
  ＞ 引き算あればできる

\[
D(ot, ov, it, iv, k) ::= \begin{cases} 
  \text{TDecl}(ID-it-iv, ot, ov, it, iv, k) \\
  \text{VDecl}(ID-it-iv, ot, ov, it, iv, k)
\end{cases}
\]

\[
\text{TDecl}(tp, ot, ov, it, iv, k) ::= \text{typedef} (ot|it) tp \\
  k(ot-tp, ov-tp, it|tp, iv)
\]

\[
\text{VDecl}(x, ot, ov, it, iv, k) ::= (ot|it) x \\
  k(ot-x, ov-x, it, iv|x)
\]
提案

高階文法
Quoteマクロ文法
引き算的なものができる人たち↓
パラメタ渡し的なものができる人たち↑

Boolean文法
PEG
このようなものを考えるとよいのではないでしょうか！

最終結論

注意：引き算的なもの入りの文法の意味論を綺麗に定めるのは難しいので、うかつにやるとチューリング完全な言語で書くのと何もかわらなくなってしまいます。注意。
Towards Higher-Order Syntax of C++
C++の文法は文脈自由ではないと言われるなぜか。

```c
FILE *fp; width * height;
n = (size_t)*ptr; f = (m) * a;
strdup(*argv)[9]; byte (*bufs)[9];
puts(str); clock_t (t);
```
そんなものは大した問題ではない

「C++は、どれだけ文脈情報を持っていても、上から順にはparseできない」

```cpp
class Foo {
    Foo() {
        hello * world;
    }
    typedef int hello;
    int world;
};
```

曖昧性の問題はCと本質的にはほぼ同じ（優先度付き非決定性で書けそう）

templateによる複雑化は、まあどうにかなる。

ので、根本問題は上のパターンだと思われる。
Towards C++ Parsing by

Higher-Order-Delimited-Continuation PEG

お楽しみに（？）