Certified Normalization of Context-Free Grammars

Denis Firsov and Tarmo Uustalu

Institute of Cybernetics at TUT, Tallinn, Estonia, {denis, tarmo}@cs.ioc.ee

Abstract

Every context-free grammar can be transformed into an equivalent one in Chomsky normal form by a sequence of four transformations. In this work, we prove in the Agda programming language that each of these transformations is correct in the sense of making progress toward normality and preserving the language of the given grammar. Also, we show that the right sequence of these transformations leads to a grammar in Chomsky normal form (since each next transformation preserves the normality property established by the previous one) that accepts the same language as the given grammar. Since we work in a constructive setting, soundness and completeness proofs are functions converting between parse trees in the normalized and original grammars.

1 Introduction

In our previous work [2] we reported about a certified implementation of Cocke–Younger–Kasami (CYK) parsing algorithm in the Agda dependently typed programming language [1]. The CYK algorithm works only with grammars in Chomsky normal form. Now we extend the reach of this work by a certified implementation of the standard normalization transformation of general context-free grammars. This transformation is the composition of the following transformations:

1. eliminating all *ε-rules*;
2. eliminating all *unit rules*;
3. replacing all rules $N \to s_1 s_2 \dots s_k$ where $k \geq 3$ with rules $N \to s_1 N_1$, $N_1 \to s_2 N_2$, $N_{k-2} \to s_{k-1} s_k$ where N_i are new nonterminals;
4. for each terminal x adding a new rule $N \to x$ where N is a new nonterminal and replacing x in the right hand sides of all rules with N.

In this extended abstract, we present only the main definitions and describe the elimination of unit rules.

The full Agda code can be found at `http://cs.ioc.ee/~denis/cert-norm/`.

2 Setup

We assume that `NT` and `Tm` are some predefined types for nonterminals and terminals respectively and define a datatype for rules:

```
String = List Tm

data Symbol : Set where  % nonterminals and terminals
  nt : NT → Symbol
  tm : Tm → Symbol
```

```
RHS = List Symbol        % right-hand sides

data Rule : Set where
  _⟶_ : NT → RHS → Rule
```

For our purposes, it is sufficient to define a grammar as a list of rules:

```
Grammar = List Rule
```

The datatype of parse trees for a grammar is defined inductively as follows:

```
mutual
  data Tree (G : Grammar) : NT → String → Set where
    node : ∀ {L R xs} → (L ⟶ R) ∈ G
                      →  Forest G R xs → Tree G L xs

  data Forest (G : Grammar) : RHS → String → Set where
    empty  : Forest G [] []
    _::t_  : ∀ {R xs} → (x : T) →  Forest G R xs
                      → Forest G (tm x :: R) (x :: xs)
    _::n_  : ∀ {R xs N ys} → Tree G N ys → Forest G R xs
                      → Forest G (nt N :: R) (ys ++ xs)
```

The type `Tree G L xs` collects all parse trees for a string `xs` in the grammar `G` starting from a nonterminal `L`. The auxiliary type `Forest G R xs` collects all parse forests for a string `xs` whose constituent individual parse trees start with the symbols `R`.

3 Unit rules elimination and its correctness

We describe in list comprehension notation how unit rules with a particular right hand nonterminal are eliminated:

```
nur : Grammar → NT → Grammar
nur G N = [ rule' | rule ← G, rule' ← nur-f G N rule ]
  where
    nur-f : Grammar → NT → Rule → Grammar
    nur-f G N (L ⟶ R) =
       if R == [ nt N ] then
          [ L ⟶ R' | (L' ⟶ R') ← G, L' == N, R' != [ nt N ] ]
       else [ L ⟶ R ]
```

The function `nur G N` replaces rules of the form `L → [ nt N ]` with rules `L ⟶ R'`, where `R'` stands for right hand sides such that `(N → R') ∈ G`. Now full unit rules elimination is defined as

```
nur-full : Grammar → Grammar
nur-full G = foldl nur G NTs
```

where `NTs` is a list of all nonterminals in the grammar.

Progress First, we show that `nur` gains some progress towards normality:

```
nur-progress : ∀ {G L N} → (L ⟶ [ nt N ]) ∉ nur G N
```

This lemma states that there is no rule with right hand side `[ nt N ]` in the grammar `nur G N`. The similar progress lemma for `nur-full` is a consequence.

Soundness Soundness states that the language of the transformed grammar is a subset of the language of the original grammar.

We start by proving a lemma about possible shapes of rules in the original grammar:

```
nur-sound-main : ∀ {G N L R} → (L ⟶ R) ∈ nur G N
      → (L ⟶ R) ∈ G ∨ (L ⟶ [ nt N ]) ∈ G × (N ⟶ R) ∈ G
```

This lemma shows that, if a rule `L ⟶ R` belongs to grammar `nur G N`, then either the rule `L ⟶ R` belongs to `G` or the rules `L ⟶ [ nt N ]` and `N ⟶ R` do.

Now, soundness can be proved by applying lemma `nur-sound-main` to each level of a given parse tree.

```
nur-soundness : ∀ {G N S xs} → Tree (nur G N) S xs → Tree G S xs
```

Completeness Completeness states that the language of the original grammar is a subset of the language of the transformed grammar.

Again we start by proving a special property:

```
nur-complete-main : ∀ {G N L R} → (L ⟶ [ nt N ]) ∈ G → (N ⟶ R) ∈ G
      → R ≠ [ nt N ] → (L ⟶ R) ∈ nur G N
```

This lemma states that, if rules `L ⟶ [ nt N ]` and `N ⟶ R` belong to the grammar `G`, then the rule `A ⟶ xs` belongs to the transformed grammar `nur G N`.

Using this property, completeness is proved by induction on a given parse tree and analyzing rules at two consecutive levels and applying lemma `nur-complete-main`.

```
nur-completeness : ∀ {G N S xs} → Tree G S xs → Tree (nur G N) S xs
```

4 Conclusion

We have done this work in the constructive setting of the Agda programming language. Hence the soundness and completeness theorems are functions for conversion of parse trees between the normalized and original grammars. We have therefore attained our initial goal of extending certified CYK parsing to grammars in general form.

Acknowledgement This research was supported by the ERDF funded Estonian ICT national programme project "Coinduction", the Estonian Science Foundation grant No. 9475 and the Estonian Ministry of Education and Research target-financed research theme No. 0140007s12.

References

[1] A. Bove, P. Dybjer, U. Norell. A brief overview of Agda, a functional language with dependent types. In *Proc. of TPHOLs 2009*, v. 5674 of *LNCS*, pp. 73–78. Springer, 2009.

[2] D. Firsov, T. Uustalu. Certified CYK parsing of context-free languages. In *Abstracts of NWPT 2012*, v. 403 of *Reports in Informatics*, 3 pp. U. of Bergen, 2012.