Closure Properties of General Grammars Formally Verified

Martin Dvorak, Jasmin Blanchette

2023-08-03

Symbols

```
inductive symbol (T : Type) (N : Type)
| terminal    : T → symbol
| nonterminal : N → symbol
```

```
inductive symbol (T : Type) (N : Type)
| terminal    : T → symbol
| nonterminal : N → symbol
```

Rules

```
inductive symbol (T : Type) (N : Type)
| terminal    : T → symbol
| nonterminal : N → symbol

structure grule (T : Type) (N : Type) :=
(input_L : list (symbol T N))
(input_N : N)
(input_R : list (symbol T N))
(output_string : list (symbol T N))
```

```
structure grule (T : Type) (N : Type) :=
(input_L : list (symbol T N))
(input_N : N)
(input_R : list (symbol T N))
(output_string : list (symbol T N))
```

Grammars

```
structure grule (T : Type) (N : Type) :=
(input_L : list (symbol T N))
(input_N : N)
(input_R : list (symbol T N))
(output_string : list (symbol T N))

structure grammar (T : Type) :=
(nt : Type)
(initial : nt)
(rules : list (grule T nt))
```

```
structure grammar (T : Type) :=
(nt : Type)
(initial : nt)
(rules : list (grule T nt))
```

Grammar transformations

```
structure grammar (T : Type) :=
(nt : Type)
(initial : nt)
(rules : list (grule T nt))

def grammar_transforms (g : grammar T)
  (w₁ w₂ : list (symbol T g.nt)) :
  Prop :=
∃ r : grule T g.nt,
  r ∈ g.rules                                  ∧
  ∃ u v : list (symbol T g.nt),
    w₁ = u ++ r.input_L
         ++ [symbol.nonterminal r.input_N]
         ++ r.input_R ++ v                                ∧
    w₂ = u ++ r.output_string ++ v
```

```
def grammar_transforms (g : grammar T)
  (w₁ w₂ : list (symbol T g.nt)) :
  Prop :=
∃ r : grule T g.nt,
  r ∈ g.rules                                  ∧
  ∃ u v : list (symbol T g.nt),
    w₁ = u ++ r.input_L
         ++ [symbol.nonterminal r.input_N]
         ++ r.input_R ++ v                                ∧
    w₂ = u ++ r.output_string ++ v
```

Grammar derivations

```
def grammar_transforms (g : grammar T)
  (w₁ w₂ : list (symbol T g.nt)) :
  Prop :=
∃ r : grule T g.nt,
  r ∈ g.rules                                  ∧
  ∃ u v : list (symbol T g.nt),
    w₁ = u ++ r.input_L
         ++ [symbol.nonterminal r.input_N]
         ++ r.input_R ++ v                                ∧
    w₂ = u ++ r.output_string ++ v

def grammar_derives (g : grammar T) :
  list (symbol T g.nt) → list (symbol T g.nt)
  → Prop :=
relation.refl_trans_gen (grammar_transforms g)
```

```
def grammar_derives (g : grammar T) :
  list (symbol T g.nt) → list (symbol T g.nt)
  → Prop :=
relation.refl_trans_gen (grammar_transforms g)
```

Words generated by a grammar

```
def grammar_derives (g : grammar T) :
  list (symbol T g.nt) → list (symbol T g.nt)
  → Prop :=
relation.refl_trans_gen (grammar_transforms g)

def grammar_generates (g : grammar T)
  (w : list T) : Prop :=
grammar_derives g
  [symbol.nonterminal g.initial]
  (list.map symbol.terminal w)
```

```
def grammar_generates (g : grammar T)
  (w : list T) : Prop :=
grammar_derives g
  [symbol.nonterminal g.initial]
  (list.map symbol.terminal w)

def language (T : Type) : Type :=
set (list T)
```

Language of a grammar

```
def grammar_generates (g : grammar T)
  (w : list T) : Prop :=
grammar_derives g
  [symbol.nonterminal g.initial]
  (list.map symbol.terminal w)

def language (T : Type) : Type :=
set (list T)

def grammar_language (g : grammar T) :
  language T :=
set_of (grammar_generates g)
```

```
def language (T : Type) : Type :=
set (list T)

def grammar_language (g : grammar T) :
  language T :=
set_of (grammar_generates g)
```

Type-0 languages

```
def language (T : Type) : Type :=
set (list T)

def grammar_language (g : grammar T) :
  language T :=
set_of (grammar_generates g)

def is_T0 (L : language T) : Prop :=
∃ g : grammar T, grammar_language g = L
```