Acyclic attribute evaluation in a dependently typed setting

Denis Firsov and Tarmo Uustalu

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

1 Introduction

Lately there has been some interest in certified parsing of context-free grammars. Attribute grammars extend context-free grammar with a semantics of parse trees in a declarative way. Let $G = \langle N, \Sigma, P, F\rangle$ be a context-free grammar. An attribute grammar extends G by specifying two disjoint sets for each nonterminal $X \in N$, namely $I(X)$ is a set of inherited and $S(X)$ a set of synthesized attributes. Let $A(X) := I(X) \cup S(X)$; then, for each production $X \to Y_1 \ldots Y_n$, every synthesized attribute in the set $S(X)$ has its value defined in terms of $A(Y_1) \cup \ldots \cup A(Y_n) \cup I(X)$ by so-called semantic equations. Similarly, each inherited attribute $I(Y_i)$ has its value defined in terms of $A(X) \cup S(Y_1) \cup \ldots \cup S(Y_n)$.

For example, we could specify the semantics of binary strings by the attribute grammar in Figure 1. Each nonterminal in this grammar has a synthesized attribute v and an inherited attribute p. The value of the attribute v for nonterminal X represents the semantical value of the tree starting from X. The value of the attribute p represents the position of the lowest bit of the subtree in the global tree.

$F \to L$	$v(F) = v(L), s(L) = s(F),$
	$s(F) = 0;$
$L_1 \to L_2B$	$v(L_1) = v(L_2) + v(B),$
	$s(L_2) = s(L_1) + 1,$
	$s(B) = s(L_1);$
$L \to B$	$v(L) = v(B), s(B) = s(L);$
$B \to 0$	$v(B) = 0;$
$B \to 1$	$v(B) = 2^{s(L)};$

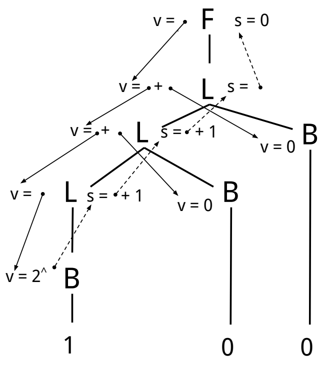

Figure 1: The attribute grammar specifies a parse tree traversal.

Figure 1 shows the parse tree of a string “100” and the evaluation of its semantic value. The semantics of that string is given by the value of attribute v of the start nonterminal F and equals 5. The value is computed by traversing the tree and applying the given semantic equations.

An attribute grammar is specification for attribute evaluation. However, it is easy to define grammars such that the dependency between attributes will be cyclic, causing lazy demand-driven attribute evaluation to diverge. An attribute grammar is called *cyclic*, if it is possible to construct a parse tree where an attribute of a particular node depends on itself.

Knuth [1] gave an algorithm for deciding whether an attribute grammar is cyclic or not. In this work we implement this algorithm in the Agda dependently typed programming language together with proofs of correctness (soundness and completeness). This certified implementation of acyclicity checker is used to define a terminating evaluator for acyclic attribute grammars.

In this abstract, we focus our attention on designing the data structures and proving the principles that allow us to talk formally about cycles between attribute occurrences on parse trees.

2 Positions and paths in trees

In this section we will give some basic inductive definitions for trees, positions in trees and paths between these positions. In these definitions, we stay as general as possible and abstract away from specifics of attribute grammars such as production rules, attributes, attribute dependencies etc.

Our work is formalized in the constructive and dependently typed setting of the Agda language. But to avoid the notational clutter in definitions and theorems, we adopt an informal language, still relying on the Curry-Howard isomorphism and dependent type theory (propositions as types, proofs as programs).

Definition 2.1 (Tree)**.** *A finitely branching tree is given by a list of finitely branching subtrees (this definition is to be read inductively; the base case occurs when the list is empty).*

Next we define positions inside a tree. Each subtree of a tree occupies a certain position in the tree taken as a whole. The position consists of the subtree, along with the directions of how to navigate from the root to that location.

Definition 2.2 (Position)**.** *A position is a proof of a proposition* $\mathsf{TreePos}\ \mathsf{t}_1\ \mathsf{t}_2$ *that states that* t_2 *is a subtree of* t_1*. The inductive definition has two constructors:*

- *If* t *is a finitely branching tree, then* $\mathsf{top}\ \mathsf{t}$ *is a position of type* $\mathsf{TreePos}\ \mathsf{t}\ \mathsf{t}$.
- *If* p *is a position of type* $\mathsf{TreePos}\ \mathsf{t}\ \mathsf{t}_1$ *and* c *is a proof that a tree* t_2 *is an immediate subtree of* t_1*, then* $\mathsf{ins}\ \mathsf{p}\ \mathsf{c}$ *is a position of type* $\mathsf{TreePos}\ \mathsf{t}\ \mathsf{t}_2$.

Definition 2.3 (Step)**.** *A single step from a position* p_1 *to a position* p_2 *is a proof of a proposition* $[\ \mathsf{p}_1\ ]\ \mathsf{d}\ [\ \mathsf{p}_2\ ]$ *where* $\mathsf{d} \in \{\uparrow, \downarrow\}$.

- $[\ \mathsf{p}_1\ ]\downarrow[\ \mathsf{p}_2\ ]$ *iff* $\mathsf{p}_2 = \mathsf{ins}\ \mathsf{p}_1\ \mathsf{c}$ *for some* c.
- $[\ \mathsf{p}_1\ ]\uparrow[\ \mathsf{p}_2\ ]$ *iff* $\mathsf{p}_1 = \mathsf{ins}\ \mathsf{p}_2\ \mathsf{c}$ *for some* c.

Next, we would like to concatenate single steps from one position to another by taking the reflexive-transitive closure of single steps. However, we also want to state that a path is bounded by some position in the tree (it is confined to the corresponding subtree) and have a flag telling whether the path is empty (zero steps) or not. Having the bound parameter will allow us to have a unique decomposition of a cycle into smaller cycles.

Definition 2.4 (Path)**.** *A path from a position* p_1 *to a position* p_2 *bounded by a position* b *is a proof of a proposition* $\mathsf{b}|[\ \mathsf{p}_1\ ]\ \mathsf{f}\ [\ \mathsf{p}_2\ ]$ *where* $\mathsf{f} \in \{\rightsquigarrow, \circlearrowleft\}$.

- *An empty cycle* $\mathsf{p}_1|[\ \mathsf{p}_1\ ]\circlearrowleft[\ \mathsf{p}_1\ ]$ *is constructed by* $\mathsf{empty}\ \mathsf{p}_1$.
- *If* s *is a single step of type* $[\ \mathsf{p}_1\ ]\ \mathsf{d}\ [\ \mathsf{p}_2\ ]$*,* p_2 *and* p_1 *are bounded by some* b*, then* $\mathsf{sngl}\ \mathsf{s}$ *is a path of type* $\mathsf{b}|[\ \mathsf{p}_1\ ]\rightsquigarrow[\ \mathsf{p}_2\ ]$.
- *If* s *is a step* $[\ \mathsf{p}_1\ ]\ \mathsf{d}\ [\ \mathsf{p}_2\ ]$*, and* p *is a path of type* $\mathsf{b}|[\ \mathsf{p}_2\ ]\rightsquigarrow[\ \mathsf{p}_3\ ]$ *for some* b*, and* p_1 *is bounded by* b*, then* $\mathsf{step}\ \mathsf{s}\ \mathsf{p}$ *is a path of type* $\mathsf{b}|[\ \mathsf{p}_1\ ]\rightsquigarrow[\ \mathsf{p}_3\ ]$.

It is clear that any path on a tree can be represented by a value of type $\mathsf{b}|[\ \mathsf{p}_1\ ]\ \mathsf{f}\ [\ \mathsf{p}_2\ ]$ for appropriately chosen b, p_1, p_2 and f. Cycles correspond to special cases when $\mathsf{p}_1 = \mathsf{p}_2$.

3 Decomposition and induction principle for cycles

Inductive types come with eliminators. We know that any natural number is either zero or successor of a smaller natural number and this deconstruction process cannot go on infinitely. The fact that a natural number can be deconstructed like this is the basis for proofs by induction. What about cycles on trees? To prove some property by induction for all cycles $\mathsf{p}|[\ \mathsf{p}\ ]\ \mathsf{f}\ [\ \mathsf{p}\ ]$, we need to argue from how cycles decompose into smaller cycles.

Theorem 3.1 (Decomposition)**.** *If* c *is a cycle of type* $\mathsf{p}|[\ \mathsf{p}\ ]\ \mathsf{f}\ [\ \mathsf{p}\ ]$ *for some tree position* p*, then either* c *is empty (*$\mathsf{f} = \circlearrowleft$*) or* $\mathsf{c} \equiv \mathsf{sngl}\ \mathsf{sd} \mathbin{+\!\!+} \mathsf{c}_1 \mathbin{+\!\!+} \mathsf{sngl}\ \mathsf{su} \mathbin{+\!\!+} \mathsf{c}_2$ *(*$\_ \mathbin{+\!\!+} \_$ *concatenates paths), where:*

- sd *is a single step down of type* $[\ \mathsf{p}\ ] \downarrow [\ \mathsf{ins}\ \mathsf{p}\ \mathsf{c}\ ]$ *for some* c*;*
- c_1 *is a cycle of type* $(\mathsf{ins}\ \mathsf{p}\ \mathsf{c})|[\ \mathsf{ins}\ \mathsf{p}\ \mathsf{c}\ ]\ \mathsf{f}\ [\ \mathsf{ins}\ \mathsf{p}\ \mathsf{c}\ ]$*; note that it is bounded by a position one level lower than the original* c*;*
- su *is a single step up of type* $[\ \mathsf{ins}\ \mathsf{p}\ \mathsf{c}\ ] \uparrow [\ \mathsf{p}\ ]$*;*
- c_2 *is another cycle of type* $\mathsf{p}\ |[\ \mathsf{p}\ ]\ \mathsf{f}\ [\ \mathsf{p}\ ]$*.*

Moreover, this decomposition is unique.

Now we could prove an induction principle for cycles on trees.

Theorem 3.2 (Induction principle)**.** *Let* P *be a property of paths of type* $\mathsf{p}|[\ \mathsf{p}\ ]\ \mathsf{f}\ [\ \mathsf{p}\ ]$*, where* p *is any position in any tree. Then to conclude that* P *holds for all cycles on all trees, we need to establish the following:*

- P *holds for empty cycles on all trees.*
- *Any proofs that* P *holds for a cycle* c_1 *of type* $(\mathsf{ins}\ \mathsf{p}\ \mathsf{c})|[\ \mathsf{ins}\ \mathsf{p}\ \mathsf{c}\ ]\ \mathsf{f}\ [\ \mathsf{ins}\ \mathsf{p}\ \mathsf{c}\ ]$ *and* P *holds for a cycle* c_2 *of type* $\mathsf{p}\ |[\ \mathsf{p}\ ]\ \mathsf{f}\ [\ \mathsf{p}\ ]$ *can be converted into a proof that* P *holds for the cycle* $\mathsf{sngl}\ \mathsf{sd} \mathbin{+\!\!+} \mathsf{c}_1 \mathbin{+\!\!+} \mathsf{sngl}\ \mathsf{su} \mathbin{+\!\!+} \mathsf{c}_2$ *where* sd *and* su *are steps down from and up to* p*.*

4 Conclusion

As we already discussed, an attribute grammar specifies attribute evaluation on parse trees. We want to know if, for a given attribute grammar, it is possible to construct a parse tree with a cyclic attribute dependency. In this abstract, we described how cycles on trees can be decomposed into smaller cycles. Then we established that this decomposition implies an induction principle for cycles. The main components of our formalization are an acyclicity checker and an attribute evaluator. By using the decomposition theorem and induction principle, we prove that the acyclicity checker is correct (sound and complete). Our attribute evaluator works on attribute grammars coming with an acyclicity proof. Attribute evaluation is defined by a recursion that is wellfounded by acyclicity (the length of any acyclic path on a parse tree is bounded by the total number of attribute occurrences in this tree).

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 institutional research grant No. IUT33-13.

References

[1] D. E. Knuth. Semantics of context-free languages. In *Mathematical Systems Theory*, v. 2, pp. 127–148, 1968.