Realizing Bidirectional Transformations in Attribute Grammars

João Saraiva[1] and Eric Van Wyk[2]

[1] University of Minho, Braga, Portugal
[2] University of Minnesota, Minneapolis, Minnesota, USA

Abstract. This position paper considers the possibility of implementing bidirectional transformations in modern attribute grammar systems. Such systems support features such as higher-order attributes, reference attributes, and forwarding of attribute queries. In the 1980's Yellin considered bidirectional transformations in attribute grammars lacking these features. But these features may open the door to automatically generating more expressive and powerful bidirectional transformations; this paper discusses the prospects of doing so.

1 Introduction

Interest in bidirectional transformations has increased in the past few years and has been studied in a number of computing disciplines [2]. In the bidirectional transformation literature the function that maps the source to the target, also called the view, is called the forward or "*get*" transformation. The function mapping the target back to the source is called the backward or "*put*" transformation. Of special interest is computing the *put* transformation after some modification of the target; this is illustrated in Figure 1.

The most interesting bidirectional transformations are those in which the source language is the richer of the two languages. Consider, for example, transformations between the concrete (source) and abstract (target) context free grammars of arithmetic expressions. The source language is the concrete syntax; the additional richness of the source (realized as more nonterminals and productions than in the target) is used to express the precedence and association of the infix binary operators. In many cases, a related collection of concrete nonterminals map to a single nonterminal in the abstract syntax. It is also often the case that several distinct source language constructs map to the same construct in the target. In the arithmetic expression example we may treat the unary negation of an expression e in the source as syntactic sugar for $0 - e$ in the target. Thus the source concrete subtraction and unary negation both map to subtraction in the target abstract syntax.

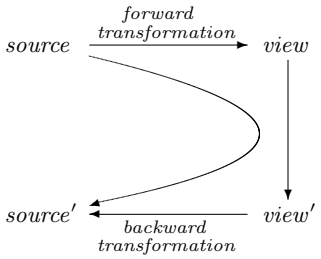

Fig. 1. Bidirectional Transformations

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 213–216

As another example consider transformations between HTML tables and an ASCII representation of them. In this case we may like to translate tables to text to allow easier editing by users not familiar with HTML and then translate back to HTML. Additional richness in the source language HTML should not be lost after the translation to text and back to HTML. Thus, it is often necessary to include the original source program as input to the *put* transformation. The idea being that a source phrase is mapped to the target, modified in some way, and then mapped back to the source. By including the original source phrase as input to the *put* transformation a more accurate or realistic source phrase can be computed from the modified target.

In the late 1980's Yellin [8] showed how attribute grammars can specify bidirectional transformations. His *reverse-inverse form* grammars were such that the *put* transformation could be automatically generated from the *get* transformation, specified as attribute definitions. Yellin's approach was proposed before higher order attribute grammars and other modern attribute grammar features were introduced. Thus his *get* and *put* attributes compute phrases as strings of symbols instead of well-typed syntax trees and assumes that only the terminal symbols of the target language are known. Furthermore, the generated *put* transformation does not take as input the original source phrase.

This raises some interesting questions. What if higher-order attributes [7] could be used? What if the nonterminals and productions of the target language are known and used in the definition of the *get* transformation? Furthermore, what if in computing the *put* transformation some information about the source construct is known? For example, is it beneficial to know the source language non-terminal type a target phrase needs to translate back to, or what production in the source language should be used to construct the translation back in the source? Also how can the actual source tree be taken into account by the generated *put* transformation to compute more appropriate translations back to the source? More questions can be raised for bidirectional transformations when additional modern attribute grammars are considered. For example, what benefit do remote [1] and reference [3] attributes provide? What about collection [1] attributes, forwarding [6], and generics [4]? With such features, how can the source be effectively used in computing the *put* transformation?

2 Bi-directional Transformations in Attribute Grammars

To get some idea how answers to these questions might play out consider Figure 2. On the left is the concrete syntax tree for the expression $-2 + 3 * 4$ and on the right is the corresponding abstract syntax tree, generated by the *get* transformation. This transformation is implemented by a collection of higher-order attributes [7] that decorate the source (concrete) nonterminals. These trees correspond to the grammars in Figure 3; nonterminals and productions in the concrete grammar are sub-scripted by "c" to distinguish them from their counterparts in the abstract grammar. Interior nodes of the trees are labelled by the productions that define them. The dashed edges from the abstract to the con-

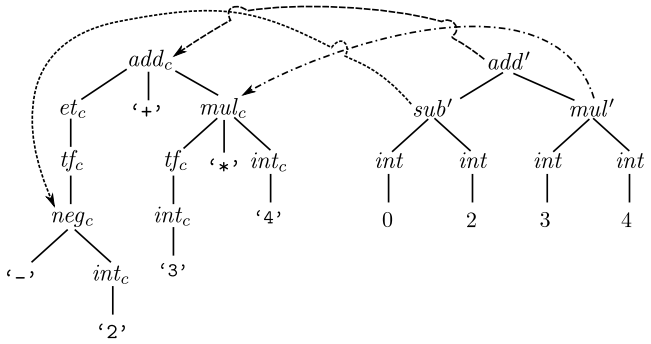

Fig. 2. Concrete and abstract syntax trees, with links back to the source, for $-2+3*4$.

$add_c : E_c ::= E_c\ `+'\ T_c$	$tf_c : T_c ::= F_c$	$add : E ::= E\ E$
$sub_c : E_c ::= E_c\ `-'\ T_c$	$int_c : F_c ::= IntLit_t$	$sub : E ::= E\ E$
$et_c : E_c ::= T_c$	$nst_c : F_c ::= `('\ E_c\ `)'$	$mul : E ::= E\ E$
$mul_c : T_c ::= T_c\ `*'\ F_c$	$neg_c : F_c ::= `-'\ F_c$	$int : E ::= IntLit_t$

Fig. 3. Concrete grammar (first two columns) and abstract (third column) grammar.

crete tree are used to provide access to the original concrete (source) tree to be used by the abstract (target) tree to compute the *put* transformation. These can be realized in attribute grammars as reference [3] or remote [1] attributes. These are effectively pointers to remote nodes in the tree, or in this case, another tree.

We need some mechanism to ensure that the these links are not lost when subtrees are moved or otherwise modified by transformations or optimizations that are performed on the target tree. Such transformations are represented by the downward arrow in Figure 1. One way to accomplish this is to use forwarding [6] and create what amounts to new "wrapper" productions for each original abstract production; this new production takes an extra argument - the reference attribute pointing back to the original source tree. These are "primed" in Figure 2 to distinguish them from the original abstract productions. Attributes defining the *get* transformation can be automatically modified to use these new productions. To compute the *put* transformation of the abstract tree created by the sub' production, we examine the link to the source and determine if this sub' tree was created by neg_c or sub_c.

Consider the rather contrived transformation that switches the order of the two expressions under *add* - a semantically safe transformation given the commutativity of addition. Processes that create the modified tree would use the original *add* production since there would not be an appropriate link back to any source tree that created it. Thus, the new tree node created by the original *add* production would not have a link back to the source, but, for example, its second child (rooted at *sub*) would maintain its link back to the source.

The point here is that modern attribute grammars contain features that can quite naturally be used to specify bidirectional transformations.

We would like to automatically generate the *put* transformation from the specification of the *get* transformation. This has the disadvantage of restricting the specification of the *get* transformation to fall into the class for which a *put* transformation can be generated, but it has the advantage over hand-written *put* transformations in that we can generate *put* transformations that are more complex and sophisticated than one would want to write by hand. For example, in the expression example multiple concrete constructs (E_c, T_c, and F_c) all map to the same abstract construct (E). If we create a *put* attribute on E for each target type, say put_{E_c}, put_{T_c}, and put_{F_c} we can generate a more realistic mapping back to the target in which we do not need to unnecessarily wrap all expressions in parenthesis (the nst_c production) but can do so only when it is required.

3 Conclusion

Yellin's work [8] scratched the surface of what is possible in implementing bidirectional transformations in attribute grammars. With the development of modern attribute grammar features new opportunities are opening for easily implementing expressive transformations between source and target languages. Silver [5] is an extensible attribute grammar system supporting these modern features in which we intend to develop new notations for specifying bidirectional transformations. There are also additional areas of application, specifically in model driven engineering in which transformations between models and programs are common. New mechanisms for implementing bidirectional transformations and new opportunities for their application provide what we believe is a promising vein of research that we are beginning to explore.

References

1. J. T. Boyland. Remote attribute grammars. *J. ACM*, 52(4):627–687, 2005.
2. K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In *ICMT '09: Proc. of the 2nd Intl. Conf. on Theory and Practice of Model Transformations*, volume 5563 of *LNCS*, pages 260–283. Springer-Verlag, 2009.
3. G. Hedin. Reference attribute grammars. *Informatica*, 24(3):301–317, 2000.
4. J. Saraiva and D. Swierstra. Generic Attribute Grammars. In *2nd Workshop on Attribute Grammars and their Applications*, pages 185–204, 1999.
5. E. Van Wyk, D. Bodin, J. Gao, and L. Krishnan. Silver: an extensible attribute grammar system. *Science of Computer Programming*, 75(1–2):39–54, January 2010.
6. E. Van Wyk, O. de Moor, K. Backhouse, and P. Kwiatkowski. Forwarding in attribute grammars for modular language design. In *Proc. Intl. Conf. on Compiler Construction*, volume 2304 of *LNCS*, pages 128–142. Springer-Verlag, 2002.
7. H. Vogt, D. Swierstra, and M. Kuiper. Higher order attribute grammars. In *ACM SIGPLAN '89 Conf. on Programming Language Design and Implementation (PLDI)*, pages 131–145. ACM, July 1989.
8. D. M. Yellin. *Attribute Grammar Inversion and Source-to-source Translation*. Number 302 in LNCS. Springer-Verlag, 1988.