

# Temporal Attribute Grammars and Incremental Evaluation

Senhua Tao  
Department of Computer Science  
University of Victoria  
Victoria, B.C. V8W 2Y2  
e-mail: stao@uvicctr.UVic.CA

(Extended Abstract)

April 11, 1990

## 1 Introduction

Recently, some research has been done on circular attribute grammars [Fa86] [JS86] [Ka89] [WJ88]. But most existing circular attribute grammars have restrictions. *Temporal Attribute Grammars* [TW90] are a new kind of attribute grammars that can specify circular attribute grammars based on iterative algorithms as non-circular ones without restrictions. In temporal attribute grammars, a circularity is viewed as an iteration, and attributes are treated as functions of time. In other words, a value of an attribute is a sequence indexed by the set of time points. If an attribute instance in a derivation tree is inside a circular definition, it may have different values at different time points; otherwise it has a constant value sequence. In a derivation tree, the value of a circular attribute instance at a time point may depend on its value at previous time points. From the temporal point of view, circular attribute instances in temporal attribute grammars do depend on themselves directly or indirectly, but not at the same time points.

Since circular attribute definitions are reduced to non-circular definitions, all attributes can be treated equally (non-circularly) in temporal attribute grammars, whether or not their instances are inside circular definitions. This allows us to evaluate circular attributes incrementally, like non-circular ones.

## 2 Expressing Circular Definitions Temporally

To express circular attribute definitions temporally, *Lucid* [WA85] is adopted as a specification language for attribute definitions in temporal attribute grammars. In a circular

definition there are normally some key attributes, like the *gate* attributes in [WJ88]. These attributes depend on the values from outside of the circularities at the initial time, then they depend on values from inside of the circularities at later time. Since these attributes may depend on their previous values directly or indirectly, circularities are involved in these attributes' definitions. Using Lucid, these kinds of circularities can be defined temporally. The following is an example showing how the Lucid operators define a circular attribute definition non-circularly.

```

stmt ::= "while" expr "do" stmts
      stmt.finalstate = if expr.value then stmts.finalstate asa not next expr.value
                        else stmt.initstate
      stmts.initstate = stmt.initstate fby stmts.finalstate
      expr.initstate = stmt.initstate fby stmts.finalstate

```

The above is a fragment of a temporal attribute grammar that specifies the denotational semantics of a *while*-statement in an imperative language. In the grammar, nonterminal symbols *stmt* and *stmts* have two attributes: *initstate* and *finalstate* that represent the states before and after a statement is executed, respectively. Since the temporal operator **fb**y shifts time points backward, *stmts.initstate* here depends on the value of *stmts.finalstate* at previous time, or  $stmts.initstate_{t+1}$  depends on  $stmts.finalstate_t$ . When the value of *expr.value* becomes false at time  $t + 1$ , the *stmt.finalstate* is assigned to the value of *stmts.finalstate* at time  $t$ . In this definition, the value of *stmts.initstate* depends on its own value at previous time, so the circular definition is reduced to a non-circular one.

### 3 Incremental Evaluation

Recently, many incremental evaluation schemas [Al88] [HK88] [Ka89] [RT83] [WJ88] have been proposed to avoid re-evaluating entire attribute values after modifications of a derivation tree. Most existing incremental systems adopt the *demand-driven* evaluation model, but they may need to construct attribute dependency graphs initially at the incremental phase.

Since our interpreter uses the demand-driven model as well, we can ask the interpreter to do a little extra work [DW90] to *remember* attribute dependencies among attributes in a derivation tree when the attribute values are first evaluated. In the derivation tree, each attribute instance has a *successor set* and a *marking counter*. The successor set of an



attribute instance remembers all of its direct successors and the marking counter counts the number of its predecessors that may have inconsistent values because of the modifications. When an attribute instance  $a$  is evaluated, the interpreter searches the values that  $a$  depends on. If  $a$  depends on the value of an attribute instance  $b$ , then  $a$  is a direct successor of  $b$  and  $b$  is a predecessor of  $a$ . The interpreter inserts  $a$  into the *successor set* of  $b$ . When all the attributes are evaluated, a dependency graph of the derivation tree is built according to the successor sets of all the attributes for further incremental evaluation.

The incremental phase in our approach is similar to the technique used in [HK88]. For simplicity, let us consider only subtree exchanges in a derivation tree. Suppose a new subtree is rooted by  $U'$  and the subtree being substituted is rooted by  $U$ , where  $U$  and  $U'$  are labeled by the same non-terminal grammar symbol. Since some synthesised attributes of  $U'$  may have values different from those they have in  $U$ , the successors of these attributes of  $U$  may be affected. To propagate the changes, marks must be sent to all the directly and indirectly affected successors according to  $U$ 's successor sets. Meanwhile, when an attribute instance receives a mark, its counter is increased by 1 to indicate that the attribute instance has one more inconsistent predecessor.

Re-evaluation for the inconsistent attribute values in a modified derivation tree is performed by the user's request. A marked attribute is re-evaluated only if it has a marked predecessor. Suppose an attribute instance  $a$  has a marked predecessor  $b$ . To re-evaluate  $a$ , the evaluator sends a demand to  $b$ . If the new value of  $b$  is the same as the old one, the marking counter of  $a$  is decreased by 1, since  $b$  has a consistent value. If the evaluator finds all predecessors of  $a$  have consistent values, then counter will be decreased to 0, and there will be no re-evaluation performed for  $a$ .

There are two advantages of my approach over others. First, since the dependency graph is build dynamicly during the evaluation phase, the required static local dependency graphs for finding an attribute dependency graph are not needed. Secondly, if a modification is made in a subtree that has not yet been evaluated, no re-evaluation will be performed because the successor sets of the root of the new subtree are empty.

The above approach also applies to the circular attribute dependencies defined by temporal attribute grammars, since temporal attribute grammars turn circular attribute dependencies into non-circular ones.

## 4 Conclusions

Temporal attribute grammars are a new kind of attribute grammars. Temporal attribute grammars can specify many circular attribute grammars as non-circular attribute grammars without restrictions. The formation of temporal attribute grammars is simple and easy to understand. Temporal attribute grammars support circular attribute grammars based on iterative algorithms and/or fixed point solutions. They also support nested circularities. Therefore, temporal attribute grammars are expected to have more expressive power than conventional attribute grammars. Incremental evaluation for temporal attribute grammars adopts the demand-driven evaluation model. In the demand-driven model, only the attributes affected by modifications are re-evaluated.

## References

- [Al88] Alblas, Henk: *Attributed Tree Transformations with Delayed and Smart Re-Evaluation*, Memorandum INF-88-45, University of Twente, The Netherlands (1988)
- [DW90] Du, Weichang. and William W. Wadge: *The Eductive Implementation of a 3-D Spreadsheet*, To appear in *Software - Practice & Experience*
- [Fa86] Farrow, R.: *Automatic Generation of Fixed-point-finding Evaluators for Circular, but Well-defined, Attribute Grammars*, SIGPLAN Notices 21, 7, pp. 85-98 (1986)
- [HK88] Hudson, Scott. and Roger King *Semantic Feedback in the Higgs UIMS*, IEEE Transaction on Software Engineering, Vol. 14, No. 8, pp. 1188-1206 (1988)
- [JS86] Jones, Larry G. and Janos Simon: *Hierarchical VLSI Design Systems Based on Attribute Grammars*, Proc. of the Thirteenth ACM Symposium on Principles of Programming Languages, pp. 58-69 (January 1986)
- [Ka89] Kaiser, Gail E.: *Incremental Dynamic Semantics for Language-Based Programming Environments*, ACM Transactions on Programming Languages and Systems 11(2), pp. 169-193 (April 1989)
- [RT83] Reps, T., T. Teitelbaum, and A. Demers: *Incremental Context-Dependent Analysis for Language-Based Editors*, ACM Transactions on Programming Languages and Systems 5(3), pp. 449-477 (1983)
- [TW90] Tao, S. and W. W. Wadge: *Temporal Attribute Grammars*, Tech. Rep. DCS-126-IR, Dept. of Comp. Sc., University of Victoria, (1990)
- [WA85] Wadge, W. W., E. A. Ashcroft: *Lucid, the Dataflow Programming Language*, Academic Press, London, England (1985)
- [WJ88] Walz, J. A. and G. F. Johnson: *Incremental Evaluation for a General Class of Circular Attribute Grammars*, Proceedings of the SIGPLAN'88 Conference on Programming Language Design and Implementation, pp. 209-221 (June 1988)