Introduction

Consider a denotational semantic specification for an imperative programming language. Essentially, such a definition can be viewed as a translation, which describes the control-structure of the programming language in terms of certain elementary operations (as e.g. updating the store, reserving storage, composing two continuations) using the concepts of the meta language used in the semantic-specification. Clearly the "complexity" of the meta-language depends on the expressive power inherent in the control-structures of the programming language. As an example, to describe the procedure concept of ALGOL 68, the full λ-calculus is needed as meta language, while the restriction to finite modes can be handled by a typed λ-calculus with fixedpoint operators [Da 2, DF 1, DF 2]. Hence, by viewing the elementary operations in the meta language as uninterpreted operation symbols, a programming language canonically induces a class of program schemes. In the deterministic case, it is well known how to associate with a scheme an infinite tree over the operation symbols which completely describes its semantics (over all possible interpretations of the operation symbols; see [Sco], [Niv], [Da 1], [Fe] for the case of flowchart-, recursive-, level-n-, and λ-schemes, respectively). In particular, when starting with some program P, the infinite tree of the associated scheme $S_p$ completely specifies the semantics of P for all possible inputs. Hence the process of compiling a higher level language into some target language PL can be viewed as a semantic preserving translation of the infinite trees of the higher level language to the class of infinite trees associated with PL. This paper should be viewed as a starting point in investigating the power of transducers needed to describe such transformations on infinite trees. We investigate the extension of the simplest kind of tree transducers, the deterministic (finite-state) top-down tree transducers, to infinite trees. It is shown, that the class $n\mathcal{R}$ of level-n trees (corresponding to ALGOL 68 programs with finite modes), is closed under deterministic top-down translations. The proof is based on the operational characterization of level-n trees as join of the schematic language generated by the corresponding scheme using outermost-innermost-derivations given in [Da 2]. As an application,
we give a full proof of the result stated in [ES], that the \( \mathcal{O} \mathcal{I} \)-hierarchy of string languages (c.f. [Da 2, ES]) starts with the regular, context-free and \( \mathcal{O} \mathcal{I} \)-macro languages. In fact, this paper was motivated by the wish to give a precise meaning to the ideas suggested in the proof of this result in [ES].

1 BASIC NOTIONS

Let \( I \) be a set of base types. \( \omega \in I^* \) of length \( k \) is viewed as a mapping \( w : [k] \to I \), where \( [k] := \{1, \ldots, k\} \), hence \( w(j) \) is the \( j \)-th letter of \( w \). We denote the empty string by \( e \).

An \( I \)-set \( A \) is a family of sets \( (A_i | i \in I) \). Define \( A^\omega \) inductively by \( A^e := \{(\)\} \), \( A^{wi} := A^w \times A^i \). For \( I \)-sets \( A, B \) we write \( A \subseteq B \) iff \( A_i \subseteq B_i \) for all \( i \in I \) and \( A \cup B \) to denote the \( I \)-set \( (A_i \cup B_i | i \in I) \).

The set of derived types over \( I \), \( D^*(I) \), is defined by

\[
D^0(I) := I, \quad D^{n+1}(I) := D^n(I)^* \times D^n(I), \quad D^*(I) := \bigcup_n D^n(I).
\]

For \( \alpha \in D^*(I)^* \) of length 1, we define \( Y_\alpha := \{ y_{1,\alpha(1)}, \ldots, y_{1,\alpha(1)} \} \), and

\[
y_\alpha := (y_{1,\alpha(1)}, \ldots, y_{1,\alpha(1)}) - \text{in particular } Y_e = \emptyset \text{ and } y_e = (\).
\]

We extend this notation to \( \tau = (a_{m}, \ldots, (a_{o}, i)) \in D^*(I) \) by setting \( \tau^T := \bigcup_{y \in \tau^T} y \).

The basic objects of this paper are typed applicative terms over sets \( Y \) of parameters, \( X \) of procedure names and \( \Sigma \) of operations symbols. Formally, let \( Y \subseteq \{ y_{j,\tau} | j \in \omega, \tau \in D^*(I) \} \), \( X \subseteq D^*(I) \), \( \Sigma \subseteq D(I) \), then the \( D^*(I) \)-set \( T_{\Sigma, X, Y} \) is the smallest \( D^*(I) \)-set \( T \) satisfying

\[
\begin{align*}
(i) & \quad \Sigma \subseteq T, \quad X \subseteq T, \quad y_{j,\tau} \in T^\tau \\
(ii) & \quad t \in T^{(\alpha, \tau)}, \quad \tau \in \tau^T.
\end{align*}
\]

Intuitively, the parameters denote positions, where terms of suitable type should be substituted (as specified by the rewriting rules of the scheme), based on the elementary operation of substituting (in parallel) \( t \in T_{\Sigma, X, Y}^\alpha \) for \( y_\alpha \) in \( t \), \( t[y_\alpha/u] \).

Note, that \( T_{\Sigma}^i (= T_{\Sigma, /, /}^i) \) coincides with the set of \( \Sigma \)-trees of sort \( i \). If \( \Sigma \) is monadic \( (\Sigma(e, i)) = \{e\}, \Sigma^T = \emptyset \) for \( \tau \neq (i, i) \), \( T_{\Sigma}^i \) is isomorphic to \((\Sigma(i, i))^* \).

If \( Y \) happens to be an \( I \)-set, we write \( T_{\Sigma}(Y) \) for \( T_{\Sigma, X, Y} \), the set of \( \Sigma \)-trees generated by \( Y \).

We use the following auxiliary functions:

- \( \tau \in D^n(I) \) has level \( n \).