Attribute Grammars and Monadic Second Order Logic

Roderick Bloem

October 15, 1996

Abstract

It is shown that formulas in monadic second order logic (MSO) with one free variable can be mimicked by attribute grammars with a designated boolean attribute and vice versa.

We prove that MSO formulas with two free variables have the same power in defining binary relations on nodes of a tree as regular path languages have. For graphs in general, MSO formulas turn out to be stronger. We also compare path languages against the routing languages of Klarlund and Schwartzbach. We compute the complexity of evaluating MSO formulas with free variables, especially in the case where there is a dependency between free variables of the formula.

Last, it is proven that MSO tree transducers have the same strength as attributed tree transducers with the single use requirement and flags.

Introduction

The main purpose of this paper is to investigate the relation between tree transductions defined by attribute grammars on one hand and tree transductions defined by monadic second order logic on the other. Along the way, we will cover path languages and we will go into some complexity issues.

Attribute grammars have been introduced by Knuth [Knu68] as a way to assign a meaning to a string from a context-free language, in a syntax directed way. Attribute grammars have become popular as a tool for building compilers and as an object for study in formal language theory. One can take a slightly different view from Knuth's, and see an attribute grammar as assigning a meaning to a *derivation tree* of a context-free language, or, in our case, to a tree over an operator alphabet. If we limit the possible meanings to trees, we obtain attributed tree transducers [Eng81, Fül81]. Tree transducers define functions from trees over one alphabet to trees over another. Our main interest lies in attributed tree transducers that have attributes whose value can range over trees (tree attributes), and attributes with a finite semantic domain (flags). Furthermore, we will forbid a tree attribute to be used more than once, in order to limit the power of attributed tree transducers. This is Giegerich's single use requirement [Gie88].

Our second important formalism, monadic second order logic, is used to define properties of graphs. It is popular because it combines great ease and strength of expression with desirable decidability properties. Closed monadic second order formulas define sets of graphs, while open monadic order formulas define relations on nodes of graphs. We also use monadic second order logic to define tree transductions, as in [Eng91, Cou92]. The monadic second order tree transducers that we consider are deterministic ones that can copy nodes. A closed MSO formula defines the domain of the transducer, while formulas with one free variable define the nodes in the output and formulas with two free variables define the edges.

Regular path languages are regular languages over a set of directives that tell you how to move through a graph, while checking properties of the nodes by MSO formulas with one free variable. Path languages define binary relations on nodes of a graph: a pair of nodes is in the relation if you can walk from the first to the second, following the directions in a string from the language. *Routing languages*, a concept akin to path languages are used by Klarlund and Schwartzbach [KS93] to extend the concept of recursive data structures. They allow the definition of data structures such as circular linked lists in an elegant

way, avoiding explicit use of pointers. It will turn out that there are structures that cannot be defined by a routing language, while they can be defined by a path language. Path languages can be evaluated efficiently, like the routing languages of Klarlund and Schwartzbach.

In Chapter 1, we will introduce the necessary formalisms and notation.

An operational description of closed MSO formulas exists in the form of tree automata. In Chapters 2 and 3 we will explore operational characterizations of MSO formulas with one and two free variables, respectively.

An attribute grammar with a designated boolean attribute recognizes exactly those nodes of a tree for which the boolean attribute has decoration 'true'. In Chapter 2, we will prove that in this way, monadic second order formulas with one free variable are equivalent to attribute grammars with a designated boolean attribute.

From this we derive the complexity of computing whether a fixed formula with free variables holds for a given tree and nodes. This can be done in linear time. We also infer the complexity of computing *all* node sequences for which a fixed formula holds, given a tree. For an MSO formula with k free variables, we can do this in $O(n^k)$ time.

In the third chapter, we show that regular path languages and monadic second order formulas with two free variables define the same binary relations on nodes in trees. On graphs in general, monadic second order formulas will turn out to be stronger than regular path languages. We will show that path languages that define a partial function can be evaluated in linear time. From this result, we conclude that if there is a dependency in the relation defined by a monadic second order formula with k free variables, we can compute the relation defined by the formula in $O(n^{k-1})$ time. This is a factor n more efficient than was shown in Chapter 2.

Finally, in Chapter 4, with help from the results of Chapters 2 and 3, we prove that monadic second order tree transducers have the same strength as attributed tree transducers with flags and the single use requirement. We also show that monadic second order tree transductions can be evaluated in linear time.

Chapter 1

Definitions and Notation

In this chapter we will define the terminology and notations we use throughout this paper. First we give some common mathematical definitions, then we define regular languages, terms and algebras, graphs and trees, tree automata, attribute grammars, and last of all monadic second order logic.

1.1 Preliminaries

We will discuss the basic mathematics here: sets, relations, functions and strings.

Sets

The set of boolean values is

$$\mathbb{B} = \{\text{false}, \text{true}\}.$$

For the sake of brevity, we sometimes use 0 instead of false and 1 instead of true. The set of natural numbers is

$$\mathbb{N} = \{0, 1, 2, \dots\}.$$

The set of positive natural numbers is $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. We use the following notation for sets of integers ($a, b \in \mathbb{N}$):

$$[a, b] = \{x \in \mathbb{N} \mid a \leq x \leq b\}.$$

Sequences are denoted $(a_1, a_2, \dots, a_n)$ or $\langle a_1, a_2, \dots, a_n \rangle$. By $(a_1, \dots, \underset{i}{b}, \dots, a_n)$, we mean the sequence $(a_1, \dots, a_{i-1}, b, a_{i+1}, \dots, a_n)$. Also,

$$(a_1, \dots, \underset{i}{b}, \dots, \underset{j}{b'}, \dots, a_n) = \begin{cases} (a_1, \dots, a_{i-1}, b, a_{i+1}, \dots, a_{j-1}, b', a_{j+1}, \dots, a_n) & \text{if } i < j \\ (a_1, \dots, a_{j-1}, b', a_{j+1}, \dots, a_{i-1}, b, a_{i+1}, \dots, a_n) & \text{if } i > j \end{cases}$$

For a set A, $\mathcal{P}(A)$ is the *power set* of A, i.e., the set of all subsets of A. The cardinality of A is denoted $\#A$. A set $\{A_1, \dots, A_n\}$ of non-empty subsets of A is a *partition* of A if $\bigcup_{i \in [1,n]} A_i = A$ and $A_i \cap A_j = \emptyset$ for $i \neq j$. If $\{A_1, \dots, A_n\}$ is a partition of A, the sequence $(A_1, \dots, A_n)$ is an *ordered partition* of A.

Relations and Functions

For sets $A_1, \ldots, A_n$, a *n-ary relation* is a subset $R \subseteq A_1 \times \cdots \times A_n$. Its *restriction* to $A_1' \subseteq A_1$ is $R \restriction A_1' = R \cap (A_1' \times A_2 \times \cdots \times A_n)$.

In particular, for sets A and B, a (binary) relation is a subset $R \subseteq A \times B$. The *image* of $A' \subseteq A$ under R is $R(A') = \{b \mid \exists a \in A' : (a, b) \in R\}$. If $A' = \{a\}$, we also write $R(a)$, instead of $R(\{a\})$, and whenever $R(A') = \{b\}$, we can write $R(A') = b$.

The *identity* on a set A is the relation $\mathrm{id}_A = \{(a, a) \mid a \in A\}$.

The *composition* of two relations $R \subseteq B \times C$ and $S \subseteq A \times B$ is $R \circ S = \{(a, c) \in A \times C \mid \exists b : (b, c) \in R \text{ and } (a, b) \in S\}$. We also use $S ; R = R \circ S$. If $R \subseteq A \times A$, we define $R^0 = \mathrm{id}_A$, and $R^i = R^{i-1} ; R$ for $i \geq 1$. The *transitive, reflexive closure* of R is $R^* = \bigcup_{i \in \mathbb{N}} R^i$.

Let A be a set. A set $A' \subseteq A$ is *closed* under a relation $R \subseteq A \times A$ if for all $x \in A'$: if $(x, y) \in R$, then $y \in A'$.

A relation $R \subseteq A \times B$ is a *(total) function*, denoted $R : A \to B$, if $\#R(a) = 1$ for every $a \in A$. So, for any function $f : A \to B$ and $A' \subseteq A$, $f(A') = \{f(a) \in B \mid a \in A'\}$. A function $f : \{a_1, \ldots, a_n\} \to \{b_1, \ldots, b_n\}$, defined as $f = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ is also written as an *enumerated function*: $[a_1 \mapsto b_1, \ldots, a_n \mapsto b_n]$.

A relation $R \subseteq A \times B$ is a *partial function*, denoted $R : A \rightharpoonup B$, if $\#R(a) \leq 1$ for every $a \in A$. The *domain* of R is the subset $\mathrm{dom}(R) = \{a \in A \mid \#R(a) = 1\}$. We use the letters f and g for total and partial functions.

Strings and String Languages

Let A be a finite set, or *alphabet*. A *string* over A is a sequence $(a_1, \ldots, a_n)$, with $n \geq 0$, and $a_i \in A$, for all $i \in [1, n]$. The string $(a_1, \ldots, a_n)$ has *length* n. The *empty string* is the string with length 0, denoted ε. The set of all strings over A is denoted A^*, and $A^+ = A^* \setminus \{\varepsilon\}$.

If $a = (a_1, \ldots, a_n)$, and $b = (b_1, \ldots, b_m)$ are strings over A, then the *concatenation* of a and b is $a \cdot b = (a_1, \ldots, a_n, b_1, \ldots, b_m)$. The empty string is the unity with respect to concatenation.

Usually, we leave out the brackets and commas, and write $a_1 a_2 \cdots a_n$ for $(a_1, a_2, \ldots, a_n)$.

Let A be an alphabet. A *(string) language* is a set of strings $L \subseteq A^*$. We have the following operations on string languages

- Concatenation: for languages $K, L \subseteq A^*$,

$$K \cdot L = \{a \cdot b \in A^* \mid a \in K \text{ and } b \in L\}.$$

- Powers: For $L \subseteq A^*$,

$$L^0 = \{\varepsilon\}, \text{ and } L^{n+1} = L^n \cdot L, \text{ for } n \in \mathbb{N}.$$

- Kleene star and Kleene plus: for $L \subseteq A^*$,

$$L^* = \bigcup_{n \in \mathbb{N}} L^n, \quad \text{and} \quad L^+ = \bigcup_{n \in \mathbb{N}_+} L^n.$$

- Naturally, the usual set operators can be used on string languages, as in $K \cup L$, $K \cap L$, and $K \setminus L$.

1.2 Regular Languages

In this section we discuss three well-known ways of defining regular string languages. See, for example [Gin75] for more details. Later on, we give an example of one language defined in these three different ways.

Finite State Automata

Definition 1.1. *A* finite state (string) automaton *(*fsa*, for short) over* Σ *is a quintuple* $A = (Q, \Sigma, \delta, q_0, F)$*, with*

- Q *is a finite set of* states,
- Σ *is the* input alphabet,
- $\delta \subseteq Q \times \Sigma \times Q$ *is the* transition relation,
- $q_0 \in Q$ *is the* start state,
- $F \subseteq Q$ *is the set of* final states.

A finite state automaton $A = (Q, \Sigma, \delta, q_0, F)$ induces a step relation $\vdash_A \subseteq (\Sigma^* \times Q) \times (\Sigma^* \times Q)$, as follows. For every $\sigma \in \Sigma$, $x \in \Sigma^*$, and $q \in Q$, if $(q, \sigma, q') \in \delta$, then $(\sigma x, q) \vdash_A (x, q')$.

Definition 1.2. *An fsa* A *over* Σ recognizes *the string language* $\|A\| \subseteq \Sigma^*$*:*

$$\|A\| = \{x \in \Sigma^* \mid \exists q_f \in F : (x, q_0) \vdash_A^* (\varepsilon, q_f)\}$$

In most of the literature, the language $\|A\|$ is denoted $L(A)$. A language is called *recognizable* if there is an fsa that recognizes it.

For reasons of notational convenience, we extend δ to a relation over $Q \times \Sigma^* \times Q$. We inductively extend the definition of δ: for all q, $(q, \varepsilon, q) \in \delta$, and for $x \in \Sigma^+$, $\sigma \in \Sigma$: $(q, x \cdot \sigma, q') \in \delta$ if $\exists q'' : (q, x, q'') \in \delta$ and $(q'', \sigma, q') \in \delta$. This is equivalent to $(q, x, q') \in \delta$ iff $(x, q) \vdash_A^* (\varepsilon, q')$.

A *deterministic* finite state automaton is a finite state automaton for which the transition relation δ is a function $Q \times \Sigma \to Q$. Accordingly, we also write $\delta(q, x) = q'$ for $(q, x, q') \in \delta$. Deterministic automata recognize exactly the same languages as nondeterministic ones do.

Right-Linear Grammars

Right-linear grammars are very much akin to finite state automata. They define a way to derive, rather than recognize a regular string language.

Definition 1.3. *A* right-linear grammar *is a quadruple* $G = (N, \Sigma, P, S)$*, where*

- N *is a finite set of* nonterminals*,*
- Σ *is the* input alphabet*,*
- $P \subseteq (N \times \Sigma \times N) \cup (N \times \{\varepsilon\})$*, is the set of* productions*, and*
- S *is the* start symbol*.*

If $(n, \sigma, n') \in P$, we write $n \to \sigma n'$, for $(n, \varepsilon) \in P$ we write $n \to \varepsilon$.

We define the language of a right-linear grammar by transforming it into an equivalent fsa. Let $G = (\Sigma, N, P, S)$ be a right-linear grammar. The corresponding fsa A_G is $(N, \Sigma, \delta, S, \{n \mid (n, \varepsilon) \in P\})$, where

$$\delta = (P \cap (N \times \Sigma \times N))$$

The language defined by G is simply $\|A_G\|$.

We use two ways of abbreviating a group of productions: if $\sigma_1, \ldots, \sigma_n \in \Sigma$, $n \geq 1$,

1. $A \to \sigma_1 \cdots \sigma_n B$ is an abbreviation for $A \to \sigma_1 A_1$, $A_1 \to \sigma_2 A_2$, $\ldots$, $A_{n-1} \to \sigma_n B$ (where $A_1, \ldots, A_{n-1}$ are new non-terminals), and
2. $A \to \sigma_1 \cdots \sigma_n$ is an abbreviation for $A \to \sigma_1 A_1$, $A_1 \to \sigma_2 A_2$, $\ldots$, $A_{n-1} \to \sigma_n A_n$, $A_n \to \varepsilon$ (where $A_1, \ldots, A_n$ are new non-terminals).

Regular Expressions

Regular expressions are the third way to define a regular string language. First we define its syntax.

Definition 1.4. *Let* Σ *be an alphabet. A* regular expression *over* Σ *is recursively defined as follows.*

1. $\emptyset$*,* ε *and* σ *(for every* $\sigma \in \Sigma$*) are regular expressions,*
2. *if* r *and* r' *are regular expressions over* Σ*, so are* $r + r'$*,* $r \cdot r'$ *and* r^**.*

A regular expression r over Σ defines a language $\|r\| \subseteq \Sigma^*$, defined as follows.

Definition 1.5. *Let* Σ *be an alphabet.*

1. $\|\emptyset\| = \emptyset$, $\|\varepsilon\| = \{\varepsilon\}$, *and* $\|\sigma\| = \{\sigma\}$ *for all* $\sigma \in \Sigma$,

2. *if* r *and* r' *are regular expressions over* Σ, $\|r + r'\| = \|r\| \cup \|r'\|$, $\|r \cdot r'\| = \|r\| \cdot \|r'\|$, *and* $\|r^*\| = \|r\|^*$.

A language L is called *regular* if there is a regular expression r with $\|r\| = L$. The set of all regular languages is denoted REG.

It is well known that REG is equal to the set of all recognizable languages, defined by a finite state automaton or a right-linear grammar.

Example 1.6. We specify one language in the three notations. The alphabet is $\Sigma = \{a, b, c\}$.

- As a finite state automaton: $A = (\{S, T\}, \Sigma, \delta, S, \{S\})$, where

$$\delta = \lambda q, \sigma. \begin{cases} S & \text{if either } q = S \text{ and } \sigma = c, \text{ or } q = T \text{ and } \sigma = b, \\ T & \text{if } q = S \text{ and } \sigma = a. \end{cases}$$

  See Figure 1.1 for the 'transition graph'.

- As a right-linear grammar: $G = (\{S, T\}, \Sigma, P, S)$, where

$$\begin{aligned} P = \{&S \to aT, \\ &S \to cS, \\ &S \to \varepsilon, \\ &T \to bS\} \end{aligned}$$

- As a regular expression: $r = (ab + c)^*$.

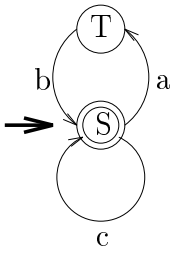

Figure 1.1: Transition Graph

□

1.3 Terms and Algebras

We here define terms, which are strings over an operator alphabet of a certain form, and algebras, a way to assign meaning to a term by giving a meaning to each element of the alphabet.

Definition 1.7. *An* operator alphabet *is a pair* $(\Sigma, \mathrm{rk}_\Sigma)$*, with* Σ *a finite set and a* rank function $\mathrm{rk}_\Sigma : \Sigma \to \mathbb{N}$*. For all* $k \in \mathbb{N}$*,* $\Sigma_k = \{\sigma \in \Sigma \mid \mathrm{rk}_\Sigma(\sigma) = k\}$*.*

We will often keep the rank function implicit and refer to Σ as the operator alphabet. The subscript to rk is omitted whenever the set is clear from the context. The subset $\Sigma_k \subseteq \Sigma$ is the set of operators of rank k. The elements of Σ_0 are called *constants*. The rank of an operator alphabet is the maximum of the rank of its elements, $\mathrm{rk}(\Sigma) = \max\{\mathrm{rk}(\sigma) \mid \sigma \in \Sigma\}$. An operator alphabet has a *rank interval*, $\mathrm{rks}(\Sigma) = [1, \mathrm{rk}(\Sigma)]$.

The set T_Σ of *terms over* Σ is a subset of Σ^+, inductively defined as follows.

Definition 1.8. *Let* Σ *be an operator alphabet.* T_Σ *is the smallest set satisfying the following condition. If* $k \geq 0$*,* $\sigma \in \Sigma_k$*, and* $t_1, t_2, \ldots, t_k \in T_\Sigma$*, then* $\sigma t_1 t_2 \cdots t_k \in T_\Sigma$*.*

Please note that the base in this induction is for the operators with rank 0. Every term has a unique decomposition in an operator and sub-terms.

We also need terms with variables. These variables are treated as constants. For variables we use the letter ξ.

Definition 1.9. *Let* Σ *be an operator alphabet and* N *a finite set (of variables), such that* $\Sigma \cap N = \emptyset$*.* $\Sigma(N)$ *is the operator alphabet with* $\Sigma(N)_0 = \Sigma_0 \cup N$ *and* $\Sigma(N)_k = \Sigma_k$ *for* $k \geq 1$*.* $T_\Sigma(N) = T_{\Sigma(N)}$ *is the set of terms over* Σ *with variables in* N*.*

A term $t \in T_\Sigma(N)$ is *linear* if every variable appears no more than once in t. It is *non-deleting* if every variable appears at least once.

These terms are merely syntactical. In order to be able to interpret them, we use an algebra.

Definition 1.10. *Let* Σ *be an operator alphabet.* A (deterministic) Σ-algebra *is a set* A*, together with, for every* $\sigma \in \Sigma_k$ $(k \geq 0)$*, a (total) function* $\sigma_A : A^k \to A$*. The algebra is* finite *if* A *is finite.*

We usually keep these functions implicit and refer to the algebra as A. We identify the nullary functions σ_A for $\sigma \in \Sigma_0$ with the corresponding elements in A.

It is not strictly necessary to associate functions with the elements of Σ. We can also use relations, and obtain a 'non-deterministic algebra'.

Definition 1.11. *Let* Σ *be an operator alphabet.* A non-deterministic algebra *is a set* A *with, for every* $\sigma \in \Sigma_k$ $(k \geq 0)$*, a relation* $\sigma_A \subseteq A^k \times A$*.*

Definition 1.12. *Let A be a non-deterministic Σ-algebra. The* subset algebra $\mathcal{P}(A)$ *is the (deterministic) algebra consisting of the set $\mathcal{P}(A)$ of subsets of A, with for every $\sigma \in \Sigma_k$, and $A_1, \ldots, A_k \subseteq A$*

$$\sigma_{\mathcal{P}(A)}(A_1, \ldots, A_k) = \sigma_A(A_1, \ldots, A_k) \\ (= \{\sigma_A(a_1, \ldots, a_k) \mid a_i \in A_i \text{ for } 1 \leq i \leq k\}).$$

Every algebra is accompanied by an interpreting function, the valuation function 'val'. First we give the valuation for terms without variables.

Definition 1.13. *Given an operator alphabet Σ and a Σ-algebra A, the function* $\mathrm{val}_A : T_\Sigma \to A$ *is defined as follows. For every $\sigma \in \Sigma_k$ $(k \geq 0)$ and $t_1, t_2, \ldots, t_k \in T_\Sigma$,*

$$\mathrm{val}_A(\sigma t_1 t_2 \cdots t_k) = \sigma_A(\mathrm{val}_A(t_1), \mathrm{val}_A(t_2), \ldots, \mathrm{val}_A(t_k))$$

The valuation of terms with variables depends on the value given to the variables. So, a term with variables needs a variable assignment, i.e., a function $N \to A$. A term over $T_\Sigma(N)$ defines a function $(N \to A) \to A$.

Definition 1.14. *Let Σ be an operator alphabet, N a set of variables, $t \in T_\Sigma(N)$, and $\alpha : N \to A$. Then,*

$$\mathrm{val}_A(t)(\alpha) = \begin{cases} \sigma_A(\mathrm{val}_A(t_1)(\alpha), \ldots, \mathrm{val}_A(t_k)(\alpha)) & \text{if } t = \sigma t_1 \cdots t_k \text{ for some } \sigma \in \Sigma_k \ (k \geq 0), \\ \alpha(t) & \text{otherwise } (t \in N) \end{cases}$$

Note that if $t \in T_\Sigma(N)$, then $t \in T_\Sigma(N')$ for any $N' \supseteq N$. Also, if $t \in T_\Sigma(N)$ and $\alpha : N \to A$ then $\mathrm{val}_A(t)(\alpha) = \mathrm{val}_A(t)(\alpha')$ for any $\alpha' : N' \to A$ with $\alpha' \restriction N = \alpha$.

Example 1.15. Consider the operator alphabet $(\Sigma, \mathrm{rk}_\Sigma)$, where $\Sigma = \Sigma_0 \cup \Sigma_2$, with $\Sigma_0 = \{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$, and $\Sigma_2 = \{\mathtt{+}\}$.

Some terms over Σ are `0`, `+01`, and `+2+11`. Mind the prefix notation.

An example of a Σ-algebra is $\mathbb{N}$, with $\mathtt{0}_\mathbb{N} = 0$, $\mathtt{1}_\mathbb{N} = 1$, $\mathtt{2}_\mathbb{N} = 2$ and $\mathtt{+}_\mathbb{N} = \lambda x, y.x+y$, addition of natural numbers. Then, $\mathrm{val}_\mathbb{N}(\mathtt{0}) = \mathtt{0}_\mathbb{N} = 0$, and $\mathrm{val}_\mathbb{N}(\mathtt{+01}) = \mathtt{+}_\mathbb{N}(\mathrm{val}_\mathbb{N}(\mathtt{0}), \mathrm{val}_\mathbb{N}(\mathtt{1})) = \mathrm{val}_\mathbb{N}(\mathtt{0}) + \mathrm{val}_\mathbb{N}(\mathtt{1}) = \mathtt{0}_\mathbb{N} + \mathtt{1}_\mathbb{N} = 0 + 1 = 1$. Similarly, $\mathrm{val}_\mathbb{N}(\mathtt{+2+11}) = 4$.

Another Σ-algebra is $A^* = \{\alpha, \beta, \gamma\}^*$, with $\mathtt{0}_{A^*} = \alpha$, $\mathtt{1}_{A^*} = \beta$, $\mathtt{2}_{A^*} = \gamma$ and $\mathtt{+}_{A^*} = \lambda x, y.x \cdot y$, concatenation of strings. Now, $\mathrm{val}_{A^*}(\mathtt{0}) = \mathtt{0}_{A^*} = \alpha$, $\mathrm{val}_{A^*}(\mathtt{+01}) = \alpha \cdot \beta = \alpha\beta$, and $\mathrm{val}_\mathbb{N}(\mathtt{+2+11}) = \gamma \cdot (\beta \cdot \beta) = \gamma\beta\beta$.

Introducing variables, $\mathtt{+0+1}\xi$ is a term from $T_{\Sigma(\{\xi\})}$. If we assign a value from the algebra to ξ, we can evaluate the term. For example, using algebra $\mathbb{N}$, and assigning 5 to ξ, we obtain $\mathrm{val}_\mathbb{N}(\mathtt{+2+1}\xi)[\xi \mapsto 5] = \mathrm{val}_\mathbb{N}(\mathtt{2})[\xi \mapsto 5] + \mathrm{val}_\mathbb{N}(\mathtt{+1}\xi)[\xi \mapsto 5] = 2 + (val_\mathbb{N}(\mathtt{1})[\xi \mapsto 5] + val_\mathbb{N}(\xi)[\xi \mapsto 5]) = 2 + (1 + 5) = 8$. □

We can consider $T_\Sigma(N)$ to be an algebra itself, with very simple operations:

Definition 1.16. *Let Σ be an operator alphabet and N a finite set of variables, then $T_\Sigma(N)$ is the* term Σ-algebra generated by N, *in which for all $k \geq 0$, $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(N)$,*

$$\sigma_{T_\Sigma(N)}(t_1, \ldots, t_k) = \sigma t_1 \cdots t_k.$$

This may not seem very useful, but we can use it to define substitution of terms for variables.

Definition 1.17. *Let Σ be an operator alphabet, N a finite set of variables, $t \in T_\Sigma(N)$, and $\alpha : N \to T_\Sigma(N)$, a* substitution function. *Then,*

$$t\alpha = \mathrm{val}_{T_\Sigma(N)}(t)(\alpha)$$

is the result of applying α *to* t.

Applying α to t has the result of substituting $\alpha(\xi)$ for every occurrence of ξ in t, for every $\xi \in N$, leaving the rest of t unchanged. This is easily proven by induction on the structure of t.

We can now show that, in a way, val_A is distributive over substitution. In the next proposition, we omit the round brackets around an enumerated function. We will do this more often.

Proposition 1.18. *For any Σ-algebra A, variable set $N = \{\xi_1, \ldots, \xi_n\}$, variable assignment $\alpha : N \to A$, and terms $t, s_1, \ldots, s_n \in T_\Sigma(N)$,*

$$\mathrm{val}_A(t[\xi_1 \mapsto s_1, \ldots, \xi_n \mapsto s_n])(\alpha) = \mathrm{val}_A(t)[\xi_1 \mapsto \mathrm{val}_A(s_1)(\alpha), \ldots, \xi_n \mapsto \mathrm{val}_A(s_n)(\alpha)]$$

Proof. In this proof, we use the shorthand $[\xi_i \mapsto s_i]$ for $[\xi_1 \mapsto s_1, \ldots, \xi_n \mapsto s_n]$ and $[\xi_i \mapsto \mathrm{val}_A(s_i)(\alpha)]$ for $[\xi_1 \mapsto \mathrm{val}_A(s_1)(\alpha), \ldots, \xi_n \mapsto \mathrm{val}_A(s_n)(\alpha)]$. We prove the proposition by induction on the structure of t.

Base: Let $t = \xi_j \in N$. Then,

$$\begin{aligned} \mathrm{val}_A(\xi_j[\xi_i \mapsto s_i])(\alpha) &= \mathrm{val}_A(s_j)(\alpha) && \text{[by definition of substitution]} \\ &= \mathrm{val}_A(\xi_j)[\xi_i \mapsto \mathrm{val}_A(s_i)(\alpha)] && \text{[by definition of val]} \end{aligned}$$

Induction: Let $t = \sigma t_1 \cdots t_k$, where $k \geq 0$ and $\sigma \in \Sigma_k$. Then,

$$\begin{aligned} &\mathrm{val}_A(\sigma t_1 \cdots t_k[\xi_i \mapsto s_i])(\alpha) && = && \text{[by def. of substitution]} \\ &\mathrm{val}_A(\sigma t_1[\xi_i \mapsto s_i] \cdots t_k[\xi_i \mapsto s_i])(\alpha) && = && \text{[by definition of val]} \\ &\sigma_A(\mathrm{val}_A(t_1[\xi_i \mapsto s_i])(\alpha), \ldots, \mathrm{val}_A(t_k[\xi_i \mapsto s_i])(\alpha)) && = && \text{[induction hypothesis]} \\ &\sigma_A(\mathrm{val}_A(t_1)[\xi_i \mapsto \mathrm{val}_A(s_i)(\alpha)], \ldots, \mathrm{val}_A(t_k)[\xi_i \mapsto \mathrm{val}_A(s_i)(\alpha)]) && = && \text{[by definition of val]} \\ &\mathrm{val}_A(\sigma t_1 \cdots t_k)[\xi_i \mapsto \mathrm{val}_A(s_i)(\alpha)] \end{aligned}$$

□

As a special case of this, if $s_1, \ldots, s_n \in T_\Sigma$, then $\mathrm{val}_A(t[\xi_1 \mapsto s_1, \ldots, \xi_n \mapsto s_n]) = \mathrm{val}_A(t)[\xi \mapsto \mathrm{val}_A(s_1), \ldots, \xi_n \mapsto \mathrm{val}_A(s_n)]$.

For a more thorough description of algebras see [Coh81].

1.4 Graphs and Trees

Graphs

Definition 1.19. *Let* Σ *and* Γ *be alphabets. A* graph *over* (Σ, Γ) *is a quadruple* $(V, E, \text{nlab}, \text{elab})$*, with* V *a finite set of nodes and* $E \subseteq V \times V$ *the set of edges,* $\text{nlab} : V \to \Sigma$ *is the node-labelling function and* $\text{elab} : E \to \Gamma$ *is the edge-labelling function.*

The set of all graphs over (Σ, Γ) is denoted $\text{GR}(\Sigma, \Gamma)$. We say that G is *a graph* if there are Σ and Γ, such that G is a graph over (Σ, Γ).

So, we consider finite, directed graphs with no multiple edges. Loops are allowed. Σ is the alphabet of node labels and Γ is the alphabet of edge labels. For a given graph G, its nodes, edges, node-labelling function and edge-labelling function are denoted V_G, E_G, nlab_G, and elab_G respectively. Sometimes we do not need edge labels and node labels and we simply leave them out.

If G is a graph, we say that an edge (with label l) runs from node v to node w if $(v, w) \in E_G$ (and $\text{elab}_G(v, w) = l$). This is denoted $v \to_G w$ ($v \xrightarrow{l}_G w$). We say that the edge is *outgoing* from v and *incoming* on w. A sequence $(v_0, v_1, \ldots, v_n) \in V^+$ is a *(directed) path* of length n, *running* from v_0 to v_n, if $(v_i, v_{i+1}) \in E$ for $i \in [0, n-1]$. The length of a path can be 0, so for every node v, there is a path from v to v. We say that a graph G is *cyclic* if there exists a path $(v_0, v_1, \ldots, v_n, v_0)$, for some $n \geq 1$.

Definition 1.20. *Two graphs* G *and* H *over* (Σ, Γ) *are* isomorphic *if there exists a bijection* $f : V_G \to V_H$*, such that for all* $v, w \in V_G$*:* $(v, w) \in E_G$ *iff* $(f(v), f(w)) \in E_H$*,* $\text{nlab}_G(v) = \text{nlab}_H(f(v))$ *and* $\text{elab}_G(v, w) = \text{elab}_H(f(v), f(w))$*.*

A graph is *automorphic* if it is isomorphic to itself through a function $f \neq \text{id}_{V_G}$.

We want to be able to 'cut a piece' out of a graph and obtain a subgraph.

Definition 1.21. *Let* G *be a graph and* $V' \subseteq V_G$*. The graph* H induced by V' *is a subgraph of* G*, with* $V_H = V'$*,* $E_H = E_G \cap (V' \times V')$*,* $\text{nlab}_H = \text{nlab}_G \restriction V_H$ *and* $\text{elab}_H = \text{elab}_G \restriction E_H$*.*

Forests and Trees

Forests and trees are specific kinds of graphs. The trees and forests we consider are rooted and ordered.

Definition 1.22. *Let* Σ *be an operator alphabet and* $\Gamma = \text{rks}(\Sigma)$*. A graph* G *over* (Σ, Γ) *is a* forest *over* Σ *if*

- *it is acyclic,*
- *no node has more than one incoming edge, and*

- *for every node v and for every $i \in [1, \mathrm{rk}(\mathrm{nlab}_G(v))]$, there is exactly one edge outgoing from v with label i, and v has only outgoing edges with labels in $[1, \mathrm{rk}(\mathrm{nlab}_G(v))]$.*

A tree *over Σ is a forest over Σ consisting of one connected component. The set of all trees is denoted* TREES.

In a forest, a node without incoming edges is called a *root*. A forest has as many roots as it has connected components, so a tree t has only one root, denoted $\mathrm{root}(t)$. Thus, a forest is a disjoint union of trees. Edges are sometimes called *branches* and nodes with no outgoing edges are called *leaves*.

Each node of a forest has a *rank* equal to the rank of its label: $\mathrm{rk}(u) = \mathrm{rk}(\mathrm{nlab}(u))$ for each node u. The *depth* or *level* of a node u is the length of the (unique) path from a root to u. The *depth* of a forest is the maximum of the depths of its nodes.

For any nodes v and w, if an edge runs from v to w (with label i), w is called the (i-th) *child* of v and v is the *parent* of w. We use Dewey notation and denote the i-th child of v by $v \cdot i$. Also, it is convenient to define $v \cdot 0 = v$.

If a path runs from v to w, then v is an *ancestor* of w and w is a *descendant* of v. In particular, every node is both its own ancestor and its own descendant. A *proper ancestor (descendant)* of v is an ancestor (descendant) of v not equal to v itself. In a tree, every pair of nodes u and v has common ancestors. The *least common ancestor* of u and v, denoted $\mathrm{lca}(u, v)$ is the common ancestor with the highest depth (closest to nodes u and v).

For a forest G over Σ and a node v of G, the *subtree* of G rooted in v, denoted by $\mathrm{sub}_G(v)$ is the subgraph of G induced by v and all its descendants. Note that $\mathrm{sub}_G(v)$ is a tree over Σ with root v.

For a tree t over Σ and a node v of t, the *context* of v, denoted $\mathrm{ctx}_t(v)$, is the subgraph induced by v and all nodes that are not a descendant of v, in which the label of v is changed into ξ, a fixed variable. Thus, $\mathrm{ctx}_t(v)$ is a tree over $\Sigma(\{\xi\})$.

The *yield* of a tree t over Σ is the string $\mathrm{yield}(t)$ over Σ_0, consisting of the concatenation of the labels of the leaves of t in left-to-right order.

Trees and Terms

It should be apparent that terms have a tree structure and vice-versa. We now formalize that correspondence, and hereafter simply identify terms with trees.

Definition 1.23. *For any operator alphabet Σ,* tree *is the bijection from T_Σ to the set of trees over Σ (modulo isomorphism), recursively defined as follows. Let $t = \sigma t_1 \cdots t_k$, for some $k \geq 0$, $\sigma \in \Sigma_k$, and $t_1, \ldots, t_k \in T_\Sigma$. Then*

$$\mathrm{tree}(\sigma t_1 \cdots t_k) = (V, E, \mathrm{nlab}, \mathrm{elab}),$$

with

$$V = \{v\} \cup \bigcup_{i \in [1,k]} V_{\mathrm{tree}(t_i)}$$

(where we assume that the $V_{\mathrm{tree}(t_i)}$ are mutually disjoint, and that $v \notin \bigcup_{i\in[1,k]} V_{\mathrm{tree}(t_i)}$),

$$E = \left\{ \left(v, \mathrm{root}(\mathrm{tree}(t_i))\right) \mid i \in [1,k] \right\} \cup \bigcup_{i\in[1,k]} E_{\mathrm{tree}(t_i)},$$
$$\mathrm{nlab} = \left\{ (v,\sigma) \right\} \cup \bigcup_{i\in[1,k]} \mathrm{nlab}_{\mathrm{tree}(t_i)}\,, \text{ and}$$
$$\mathrm{elab} = \left\{ \left(\left(v, \mathrm{root}(\mathrm{tree}(t_i))\right), i\right) \mid i \in [1,k] \right\} \cup \bigcup_{i\in[1,k]} \mathrm{elab}_{\mathrm{tree}(t_i)}\,.$$

1.5 Tree Automata

A tree automaton defines a tree language, i.e., a set of trees. We consider only finite state, bottom-up tree automata.

Definition 1.24. *Let Σ be an operator alphabet. A* deterministic tree automaton *over Σ is a tuple $\mathcal{A} = (Q, F)$, where Q is a finite Σ-algebra and $F \subseteq Q$. The language recognized by the tree automaton (Q, F) is*

$$L(\mathcal{A}) = \{t \in T_\Sigma \mid \mathrm{val}_Q(t) \in F\}.$$

A more operational description may be given in terms of automata theory. Elements of Q are called states and F is the set of final states. A tree automaton $\mathcal{A}$ starts reading at each leaf of the tree, in a state determined by the label of that leaf. It then works its way up in the tree. When a node v has label $\sigma \in \Sigma_k$, the automaton will reach state $\sigma_Q(s_1, \ldots, s_k)$ at v, where s_i is the state of $\mathcal{A}$ at the i-th child of v. A tree is accepted if the automaton reaches a final state in the root of the tree.

A tree language L is called *recognizable* or *regular* if there is a tree automaton that recognizes L. The set of all regular tree languages is denoted REGT.

Definition 1.25. *Let Σ be an operator alphabet. A* non-deterministic tree automaton *over Σ is a tuple $\mathcal{A} = (Q, F)$, where Q is a non-deterministic finite Σ-algebra and $F \subseteq Q$ is the set of final states. The language recognized by $\mathcal{A}$ is*

$$L(\mathcal{A}) = \{t \in T_\Sigma \mid \mathrm{val}_{\mathcal{P}(Q)}(t) \cap F \neq \emptyset\}.$$

If you wish, $\mathrm{val}_{\mathcal{P}(Q)}(\mathrm{sub}_t(v))$ gives the states in which the automaton *can* arrive at v, and the automaton succeeds if it *can* arrive at the root in a final state. Non-deterministic tree automata recognize exactly the same tree languages as deterministic tree automata do. See [GS84] for this result and a more comprehensive introduction to tree automata.

Proposition 1.26. *For a tree automaton $\mathcal{A}$ over Σ the following are decidable:*

1. *$t \in L(\mathcal{A})$, for any $t \in T_\Sigma$,*

2. $L(\mathcal{A}) = \emptyset$, *and*

3. $L(\mathcal{A}) = T_\Sigma$.

1.6 Attribute Grammars

Attribute grammars were introduced by [Knu68], as a way to assign a semantics to a context-free language. An overview and extensive bibliography can be found in [DJL88]. To simplify the comparison with other formal models, our definition of attribute grammars is a bit different from Knuth's original.

1.6.1 Definition

As in [Fül81], attribute grammars act on trees over an operator alphabet, instead of parse trees of an underlying context-free grammar. The semantic rules are grouped by operator, instead of by grammar rule. All operators have the same set of attributes, and there are special rules for the inherited attributes of the root. Moreover, for each operator one semantic condition is given.

Definition 1.27. *An* attribute grammar *over* Σ *is a six-tuple*

$$G = \langle \Sigma, \Omega, B, R, C, \alpha_{mean} \rangle,$$

where

- Σ *is an operator alphabet;*
- Ω *is a finite set of sets, the* semantic domains *of the attributes;*
- $B = \langle S, I, W \rangle$, *is the* attribute description. *Here,*
  - S *is a finite set, the set of* synthesized attributes,
  - I, *disjoint with* S, *is a finite set, the set of* inherited attributes,
  - $W : (I \cup S) \to \Omega$ *is the* domain assignment.

  We also use A *for* $I \cup S$, *the set of all attributes;*
- $R = \langle R_{int}, R_{\text{root}} \rangle$ *describes the* semantic rules. *Here,*
  - R_{int} *is a function associating with every* $\sigma \in \Sigma$ *a set of* internal semantic rules. *For every* σ, $R_{int}(\sigma)$ *contains one rule*

    $$\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle),$$

    for every pair $\langle \alpha_0, i_0 \rangle$, *where either* α_0 *is a synthesized attribute and* $i_0 = 0$, *or* α_0 *is an inherited attribute and* $i_0 \in [1, \text{rk}(\sigma)]$. *Furthermore,* $k \geq 0$, $\alpha_1, \ldots, \alpha_k \in A$,

$i_1, \ldots, i_k \in [0, \mathrm{rk}(\sigma)]$, *$f$ is a function from* $W(\alpha_1) \times \cdots \times W(\alpha_k)$ *to* $W(\alpha_0)$, *and the* $\langle \alpha_j, i_j \rangle$ *are mutually distinct.*
Usually, we simply write $R(\sigma)$ *for* $R_{int}(\sigma)$*;*

– R_{root} *is the set of* semantic root rules*;* R_{root} *contains one rule*

$$\langle \alpha_0, 0 \rangle = f(\langle \alpha_1, 0 \rangle, \ldots, \langle \alpha_k, 0 \rangle),$$

for every $\alpha_0 \in I$, *where* $k \geq 0$, $\alpha_1, \ldots, \alpha_k \in A$, *$f$ is a function from* $W(\alpha_1) \times \cdots \times W(\alpha_k)$ *to* $W(\alpha_0)$, *and the* $\langle \alpha_j, i_j \rangle$ *are mutually distinct;*

- *C is a function associating with every* $\sigma \in \Sigma$ *a* semantic condition $C(\sigma)$ *of the form*

$$f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle)$$

where $k \geq 0$, $\alpha_1, \ldots, \alpha_k \in A$, $i_1, \ldots, i_k \in [0, \mathrm{rk}(\sigma)]$, *and f is a function from* $W(\alpha_1) \times \cdots \times W(\alpha_k)$ *to* $\mathbb{B}$*;*

- $\alpha_{mean} \in S$ *is the attribute giving the* meaning *of a tree.*

The semantic conditions can be left out if they are all tautologies. An attribute grammar without semantic conditions is called an *unconditional* attribute grammar. Also, α_{mean} can be left out if it is not needed. If all sets in Ω are finite, then G is said to have *finite semantic domains.* Usually, for an (internal or root) rule

$$\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle),$$

the function f is given as $f = \lambda x_1, \ldots, x_k.e$ for some expression e with variables in $\{x_1, \ldots, x_k\}$. We will then informally denote the rule by

$$\langle \alpha_0, i_0 \rangle = e',$$

where e' is obtained from e by substituting $\langle \alpha_j, i_j \rangle$ for x_j for all j.

For an operator $\sigma \in \Sigma$, we define the set $A(\sigma)$ of attributes of σ as $\{\langle \alpha, i \rangle \mid \alpha \in A \text{ and } i \in [0, \mathrm{rk}(\sigma)]\}$. If $\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle)$ is a rule in $R(\sigma)$, we say $\langle \alpha_0, i_0 \rangle$ is *defined in terms of* $\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle$ (at σ). Likewise, if $\langle \alpha_0, 0 \rangle = f(\langle \alpha_1, 0 \rangle, \ldots, \langle \alpha_k, 0 \rangle)$ is a rule in R_{root}, then $\langle \alpha_0, 0 \rangle$ is said to be defined in terms of $\langle \alpha_1, 0 \rangle, \ldots, \langle \alpha_k, 0 \rangle$ (at the root). For attributes α and β in A, if there are i, j such that $\langle \alpha, i \rangle$ is defined in terms of $\langle \beta, j \rangle$ at some σ, or at the root, then we say α is defined in terms of β.

Remark 1.28. We do not require *Bochmann normal form* [Boc76].

1.6.2 Semantic Instructions and Tests

Let t be a tree over Σ. The set of attributes of t is

$$A(t) = A \times V_t.$$

Definition 1.29. *For every tree $t \in T_\Sigma$, and node $u \in V_t$, if* $\mathrm{nlab}(u) = \sigma$, *and*

$$\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle)$$

is a rule in $R(\sigma)$, then

$$\langle \alpha_0, u \cdot i_0 \rangle = f(\langle \alpha_1, u \cdot i_1 \rangle, \ldots, \langle \alpha_k, u \cdot i_k \rangle)$$

is an internal semantic instruction *of t. Analogously, if $\langle \alpha_0, 0 \rangle = f(\langle \alpha_1, 0 \rangle, \ldots, \langle \alpha_k, 0 \rangle)$ is a root rule of t, then $\langle \alpha_0, \mathrm{root}(t) \rangle = f(\langle \alpha_1, \mathrm{root}(t) \rangle, \ldots, \langle \alpha_k, \mathrm{root}(t) \rangle)$ is a* semantic root instruction *of t. Likewise for the semantic conditions: if $C(\sigma) = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle))$, then $f(\langle \alpha_1, u \cdot i_1 \rangle, \ldots, \langle \alpha_k, u \cdot i_k \rangle))$ is a* semantic test *of t.*

The set of all internal semantic instructions of t is $R_{\mathrm{int}}(t)$. The set of all semantic root instructions is $R_{\mathrm{root}}(t)$. The set of all semantic instructions of t is $R(t) = R_{\mathrm{int}}(t) \cup R_{\mathrm{root}}(t)$. The set of all semantic tests of t is $C(t)$. So, semantic rules and conditions are associated with labels, and instructions and tests are associated with nodes of a tree.

1.6.3 Decoration

We define how to give the correct values to the attributes of the tree.

Definition 1.30. *Let Σ be an operator alphabet, G an attribute grammar over Σ, and t a tree over Σ. Let* dec *be a function from $A(t)$ to $\cup\Omega$, such that* $\mathrm{dec}(\langle \alpha, u \rangle) \in W(\alpha)$ *for all $\langle \alpha, u \rangle \in A(t)$. The function* dec *is a* decoration *of t if the following two conditions hold.*

1. *All semantic root instructions are obeyed, i.e., for every instruction*

$$\langle \alpha_0, \mathrm{root}(t) \rangle = f(\langle \alpha_1, \mathrm{root}(t) \rangle, \ldots, \langle \alpha_k, \mathrm{root}(t) \rangle)$$

in $R_{root}(t)$,

$$\mathrm{dec}(\langle \alpha_0, \mathrm{root}(t) \rangle) = f(\mathrm{dec}(\langle \alpha_1, \mathrm{root}(t) \rangle), \ldots, \mathrm{dec}(\langle \alpha_k, \mathrm{root}(t) \rangle)).$$

2. *The internal semantic instructions are obeyed, i.e., for every instruction*

$$\langle \alpha_0, u \cdot i_0 \rangle = f(\langle \alpha_1, u \cdot i_1 \rangle, \ldots, \langle \alpha_k, u \cdot i_k \rangle)$$

in $R_{\mathrm{int}}(t)$,

$$\mathrm{dec}(\langle \alpha_0, u \cdot i_0 \rangle) = f(\mathrm{dec}(\langle \alpha_1, u \cdot i_1 \rangle), \ldots, \mathrm{dec}(\langle \alpha_k, u \cdot i_k \rangle)).$$

Moreover, dec *is a* valid decoration *if the following condition holds:*

3. *The semantic tests are passed, i.e., for every test*

$$f(\langle \alpha_1, u \cdot i_1 \rangle, \ldots, \langle \alpha_k, u \cdot i_k \rangle)$$

in $C(t)$*, we have*

$$f(\mathrm{dec}(\langle \alpha_1, u \cdot i_1 \rangle), \ldots, \mathrm{dec}(\langle \alpha_k, u \cdot i_k \rangle)) = \mathrm{true}\,.$$

Note that the first two conditions can be merged to the following: for every instruction

$$\langle \alpha_0, u_0 \rangle = f(\langle \alpha_1, u_1 \rangle, \ldots, \langle \alpha_k, u_k \rangle)$$

in $R(t)$,

$$\mathrm{dec}(\langle \alpha_0, u_0 \rangle) = f(\mathrm{dec}(\langle \alpha_1, u_1 \rangle), \ldots, \mathrm{dec}(\langle \alpha_k, u_k \rangle)).$$

1.6.4 Weak Non-circularity

We introduce a condition that implies that an attribute grammar has a unique decoration on every tree. This criterion is a bit more relaxed than the classical concept of non-circularity, but it suffers from a computability problem.

For classical non-circularity [Knu68], we demand that the dependency graph is acyclic for every tree. For some of our purposes, specifically in Chapter 4 (where we consider conditional semantic rules in combination with decomposition in phases), this condition is too conservative. Therefore we introduce the concept of a weak-dependency graph, that has edges corresponding to 'real' dependencies only.

Definition 1.31. *A function* f depends *on its* i*th argument if*

$$\exists a_1, \ldots, a_{i-1}, a, a', a_{i+1}, \ldots, a_k : f(a_1, \ldots, \underset{i}{a}, \ldots, a_k) \neq f(a_1, \ldots, \underset{i}{a'}, \ldots, a_k).$$

For example, the function $\lambda a, b, c.(a+b)/c$ does depend on it's third argument, but the function $\lambda a, b, c.a + b$ does not depend on its third argument.

It is undecidable whether f depends on its ith argument. This follows from Rice's theorem ('every non-trivial property of a computable function is undecidable'), see e.g. [Wei87].

In a weak-dependency graph we depict how one attribute of a node depends on another.

Definition 1.32. *For an attribute grammar* G*, the* weak-dependency graph *of a tree* t *over* Σ *is the unlabelled graph* $\mathrm{WD}_G(t) = (V, E)$*, where*

$$\begin{aligned} V &= A(t) \\ E &= \{(\langle \alpha, u \rangle, \langle \alpha', u' \rangle) \mid \exists i : (\langle \alpha', u' \rangle = f(\ldots, \underset{i}{\langle \alpha, u \rangle}, \ldots)) \in R(t) \\ &\qquad\qquad \textit{and } f \textit{ depends on its } i\textit{th argument}\}. \end{aligned}$$

Now, we call an attribute grammar G *weakly non-circular* if for every t, $\mathrm{WD}_G(t)$ is acyclic.

Unfortunately, because it is undecidable whether f depends on its ith argument, the weak-dependency graph is not computable, and weak non-circularity is in general not decidable. In the special case of attribute grammars with finite semantic domains, however, weak non-circularity is decidable, because dependencies are computable for finite functions. Note that weak non-circularity is indeed implied by classical non-circularity.

Proposition 1.33. *If an attribute grammar G is weakly non-circular, then for every t, there is exactly one decoration of t.*

The truth of this proposition can most easily be seen by the fact that any weakly non-circular AG can be changed into a classically non-circular AG with the same decorations, in the following straightforward manner. For every semantic rule

$$\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle)$$

and $j \in [1, k]$, we remove attribute $\langle \alpha_j, i_j \rangle$ if f does not depend on j (and changing f accordingly). This does not change the effect of the rules, but makes the attribute grammar classically non-circular. Weak non-circularity only makes a real difference if we use a *decomposition in phases*, defined below in Subsection 1.6.7. There, we will present an example in which weak non-circularity really differs from classical non-circularity.

If t has a unique decoration, we denote it by $\mathrm{dec}_{G,t}$. We will also write $\langle \alpha, u \rangle_{G,t}$ for $\mathrm{dec}_{G,t}(\langle \alpha, u \rangle)$. We will leave out the subscripts whenever they are clear from context.

Note that weak non-circularity is a sufficient, but not a necessary condition for uniqueness of the decoration, see for example [CM79].

Definition 1.34. *A weakly non-circular attribute grammar G defines a partial function $\mathcal{G} : T_\Sigma \rightharpoonup W(\alpha_{mean})$, as follows*

$$\mathcal{G}(t) = \mathrm{dec}_{G,t}(\langle \alpha_{mean}, \mathrm{root}(t) \rangle), \quad \textit{if } \mathrm{dec}_{G,t} \textit{ is valid.}$$

For weakly non-circular unconditional attribute grammars, $\mathcal{G}$ is a total function (because in that case, all decorations are valid).

1.6.5 Weak Single Use Requirement

The 'weak single use requirement' [Gie88] states that in no tree an attribute should be used more than once. We will use this in Chapter 4 to limit the power of tree transducers.

Definition 1.35. *An AG G over Σ is* WSUR *if for every $t \in T_\Sigma$, $\mathrm{WD}_G(t)$ has no node with more than one outgoing edge.*

WSUR is in general not decidable, because dependencies are not computable.

1.6.6 Computing the decoration

We show here how we can compute a valid decoration bottom-up, in a non-deterministic way, if one exists. This is not the way it is usually done, but we need this for Chapter 2, where we will have a tree automaton simulate an attribute grammar. See [Eng84], but also [FV95] for the usual ways to compute decorations.

Definition 1.36. *Let Σ be an operator alphabet, G a weakly non-circular attribute grammar over Σ and t a tree over Σ. A* top-less decoration *of t is a function* dec *assigning to every $\langle \alpha, u\rangle \in A(t)$ a value $\mathrm{dec}(\langle \alpha, u\rangle) \in W(\alpha)$, such that conditions 2 and 3 of Definition 1.30 are satisfied.*

Loosely speaking, a top-less decoration is a valid decoration that does not necessarily satisfy the root rules. A tree can have more than one top-less decoration (given an AG). Intuitively, this is because the inherited attributes of the root are not prescribed. However, a top-less decoration that satisfies condition 1 of Definition 1.30, is a valid decoration.

Using top-less decorations we can build a valid decoration bottom-up. We will have to do this non-deterministically, guessing the right top-less decoration out of the possible ones, on every step up. The following lemma shows how to make a step up in the tree.

Lemma 1.37. *Let $t = \sigma t_1 \cdots t_k$ be a tree over Σ, and G a weakly non-circular* AG *over Σ. A function* dec *is a top-less decoration of t iff the following conditions hold:*

- $\mathrm{dec} \restriction A(t_i)$ *is a top-less decoration of t_i, for all $i \in [1, k]$,*
- *for every rule*
$$\langle \alpha_0, i_0\rangle = f(\langle \alpha_1, i_1\rangle, \ldots, \langle \alpha_r, i_r\rangle)$$
in $R(\sigma)$,
$$\mathrm{dec}(\langle \alpha_0, \mathrm{root}(t)\cdot i_0\rangle) = f\Big(\mathrm{dec}(\langle \alpha_1, \mathrm{root}(t)\cdot i_1\rangle), \ldots, \mathrm{dec}(\langle \alpha_r, \mathrm{root}(t)\cdot i_r\rangle)\Big), \text{ and}$$
- *if $C(\sigma) = f(\langle \alpha_1, i_1\rangle, \ldots, \langle \alpha_r, i_r\rangle)$, then*
$$f\Big(\mathrm{dec}(\langle \alpha_1, \mathrm{root}(t)\cdot i_1\rangle), \ldots, \mathrm{dec}(\langle \alpha_r, \mathrm{root}(t)\cdot i_r\rangle)\Big) = \mathrm{true}\,.$$

The straightforward proof is omitted.

1.6.7 Decomposition in Phases

In an attribute grammar, there may be a (natural) order in the attributes. Sometimes the value of one attribute does not depend on the value of another. Then, the value of all occurrences of one attribute can be evaluated before the value of any occurrence of another attribute is known. In such a case, the evaluation of the attributes can be decomposed into phases.

Definition 1.38. *Let* $G = \langle \Sigma, \Omega, B, R, C, \alpha_{mean} \rangle$ *be a* not necessarily weakly non-circular *attribute grammar. An ordered partition* $P = (A_1, \ldots, A_n)$ *of* A *is a* decomposition in phases *if for all* $\alpha, \beta \in A$*, if* α *is defined in terms of* β*, and* $\alpha \in A_k$*, then* $\beta \in A_j$ *for some* $j \leq k$*. If* P *is a decomposition in phases, we define* $\text{phase}_P(\alpha) = p$ *iff* $\alpha \in A_p$*.*

A decomposition in phases not only separates the attributes, but also the evaluation of the attributes. We can first evaluate the attributes in phase 1, then the attributes in phase 2, and so on, resulting in the original decoration (if it exists).

For every phase $p \in [1, n]$ and tree $t \in T_\Sigma$, we have the set of tree attributes $A_p(t) = A_p \times V_t$. Furthermore, we have the sets of semantic instructions $R^p_{\text{int}}(t)$, $R^p_{\text{root}}(t)$, and $R^p(t)$, and the decoration $\text{dec}^p_{G,t}$ (if it exists). We will define them using simultaneous induction on p, assuming $\text{dec}^q_{G,t}$ exists for all $q < p$.

- First, $R^p_{\text{int}}(t)$ consists of the instructions in $R_{\text{int}}(t)$ with left-hand side $\langle \alpha, i \rangle$ for certain i and $\alpha \in A_p$, where $\text{dec}^q(\langle \beta, j \rangle)$ takes the place of $\langle \beta, j \rangle$, for all j and β with $\text{phase}(\beta) = q < p$. Technically, if

  $$\langle \alpha_0, u_0 \rangle = f(\langle \alpha_1, u_1 \rangle, \ldots, \langle \alpha_k, u_k \rangle) \in R_{\text{int}}(t),$$

  and $\alpha_0 \in A_p$, then

  $$\langle \alpha_0, u_0 \rangle = f'(\langle \alpha_1, u_1 \rangle, \ldots, \langle \alpha_k, u_k \rangle) \in R^p_{\text{int}}(t),$$

  where for all $w_1 \in W(\alpha_1), \ldots, w_k \in W(\alpha_k)$, $f'(w_1, \ldots, w_k) = f(x_1, \ldots, x_k)$, where

  $$x_l = \begin{cases} w_l & \text{if } \text{phase}(\alpha_l) \geq p, \\ \text{dec}^q(\langle \alpha_l, i_l \rangle) & \text{if } \text{phase}(\alpha_l) = q < p. \end{cases}$$

  Note that in the second case, f' does not depend on its lth argument. Analogously we define $R^p_{\text{root}}(t)$, and last, $R^p(t) = R^p_{\text{int}}(t) \cup R^p_{\text{root}}(t)$.

- A phase-p decoration is defined like in Definition 1.30: it is a function from $A_p(t)$ to $\bigcup \Omega$ (that assigns values from the appropriate semantic domains) such that

  1. all semantic root instructions in $R^p_{\text{root}}(t)$ are obeyed, and
  2. all internal semantic instructions in $R^p_{\text{int}}(t)$ are obeyed.

  If t has a unique a phase-p decoration, it is denoted $\text{dec}^p_{G,t}$.

Note that $\text{dec}^p_{G,t}$ exists for all p iff $\text{dec}_{G,t}$ exists, and that if that is the case, $\text{dec}_{G,t} = \bigcup_{p \in [1,n]} \text{dec}^p_{G,t}$. Note that for any p, phases 1 up to p constitute an attribute grammar in its own right.

The fact that we have attributes and instructions for every phase means that we can define a weak-dependency graph for every phase.

Definition 1.39. *For* AG G *with decomposition in phases* $P = (A_1, \ldots, A_n)$*, the* phase-p weak-dependency graph *of a tree* t *over* Σ *is the unlabelled graph* $\mathrm{WD}^p_{G,P}(t) = (V, E)$*, where*

$$V = A_p(t)$$
$$E = \{(\langle \alpha, u\rangle, \langle \alpha', u'\rangle) \mid \exists i : \langle \alpha', u'\rangle = f(\ldots, \underset{i}{\langle \alpha, u\rangle}, \ldots) \in R^p(t)$$
$$\text{and } f \text{ depends on its } i\text{th argument}\}.$$

This means the concepts of weak non-circularity and WSUR generalize naturally to phases.

Definition 1.40. *Let* G *be a (not necessarily weakly non-circular) attribute grammar, and* P *a decomposition in phases of* G*. Attribute grammar* G *is* phase-p WSUR with respect to P *if for every tree* t*, no node in* $\mathrm{WD}^p(t)$ *has more than one outgoing edge. It is* phase-p weakly non-circular with respect to P *if for every* t*,* $\mathrm{WD}^p(t)$ *is acyclic. If* G *is phase-*p *weakly non-circular for every phase* p*,* G *is* weakly non-circular with respect to P*.*

Lemma 1.41. *If an attribute grammar* G *is weakly non-circular with respect to a decomposition in phases* P*, then, for every* t*, there is exactly one decoration of* t*.*

The proof is analogous to the classical proof that non-circular attribute grammars induce a unique decoration on every tree.

Note that if an attribute grammar is weakly non-circular, it is weakly non-circular with respect to any decomposition in phases, because $\mathrm{WD}^p_{G,P}$ is a subgraph of $\mathrm{WD}_G(t)$. The opposite need not hold: an attribute grammar can be non-circular with respect to a decomposition in phases while it is not non-circular. As shown in the next example, this happens in particular when conditional semantic rules are used (as will be done in Chapter 4).

Example 1.42. Consider the attribute grammar $G = \langle \Sigma, \Omega, (S, I, W), R\rangle$, with $\Sigma = \Sigma_0 \cup \Sigma_1$, $\Sigma_0 = \{*, \#\}$, $\Sigma_1 = \{\sigma\}$, $\Omega = \{\mathbb{N}\}$, $S = \{\alpha, \beta, \gamma\}$, $I = \emptyset$, W maps every element of S to $\mathbb{N}$, and the semantic rules are as follows. The set of rules $R(*)$ is

$$\langle \alpha, 0\rangle = 0$$
$$\langle \beta, 0\rangle = 0$$
$$\langle \gamma, 0\rangle = 0$$

The set of rules $R(\#)$ is

$$\langle \alpha, 0\rangle = 1$$
$$\langle \beta, 0\rangle = 1$$
$$\langle \gamma, 0\rangle = 1$$

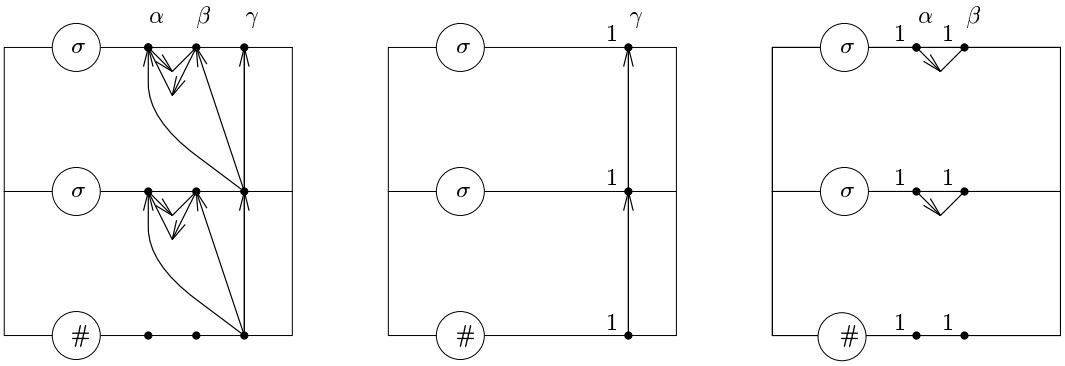

Figure 1.2: The dependency graph of $\sigma\sigma\#$, its phase-1 dependency graph and its phase-2 dependency graph

The set of rules $R(\sigma)$ is

$$\langle \alpha, 0 \rangle = \begin{cases} \langle \beta, 0 \rangle & \text{if } \langle \gamma, 1 \rangle = 0 \\ 1 & \text{otherwise} \end{cases}$$

$$\langle \beta, 0 \rangle = \begin{cases} 0 & \text{if } \langle \gamma, 1 \rangle = 0 \\ \langle \alpha, 0 \rangle & \text{otherwise} \end{cases}$$

$$\langle \gamma, 0 \rangle = \langle \gamma, 1 \rangle$$

The leftmost picture in Figure 1.2 shows the weak-dependency graph $\mathrm{WD}(t)$ of the tree $t = \sigma\sigma\#$. Clearly, the attribute grammar is circular. We can however decompose it, with decomposition $P = (\{\gamma\}, \{\alpha, \beta\})$. If we do this, we see that the weak-dependency graphs of both phases are acyclic, for example, in the case of tree t, the phase-1 and phase-2 dependency graphs $\mathrm{WD}^1(t)$ and $\mathrm{WD}^2(t)$ are depicted in Figure 1.2. Note that the value of γ is known in the second phase, and hence in both the rules for α and β, one of the cases falls away, which means that some dependencies no longer exist.

It follows that G is weakly non-circular with respect to P. Indeed, there is a unique decoration for every tree: A tree of the form $\sigma\sigma\cdots\sigma*$ has $\mathrm{dec}_G(\langle \alpha, u \rangle) = \mathrm{dec}_G(\langle \beta, u \rangle) = \mathrm{dec}_G(\langle \gamma, u \rangle) = 0$ for all nodes u, and a tree of the form $\sigma\sigma\cdots\sigma\#$ has $\mathrm{dec}_G(\langle \alpha, u \rangle) = \mathrm{dec}_G(\langle \beta, u \rangle) = \mathrm{dec}_G(\langle \gamma, u \rangle) = 1$ for all nodes u. □

1.7 Monadic Second Order Logic on Graphs

Monadic second order logic is used to describe properties of graphs. We will informally introduce it here. The interested reader is refered to [Cou90, Eng91, Oos89].

Syntax

For alphabets Σ and Γ, we use the language $\mathrm{MSOL}(\Sigma, \Gamma)$, of monadic second order (MSO) formulas over (Σ, Γ). Formulas in $\mathrm{MSOL}(\Sigma, \Gamma)$ describe properties of graphs over (Σ, Γ). This logical language has node variables $x, y, \ldots$, and node-set variables $X, Y, \ldots$. For a given graph G over (Σ, Γ), node variables range over the elements of V_G, and node-set variables range over the subsets of V_G.

There are three types of atomic formulas in $\mathrm{MSOL}(\Sigma, \Gamma)$:

1. $\mathrm{lab}_\sigma(x)$, for every $\sigma \in \Sigma$, denoting that x has label σ,
2. $\mathrm{edg}_\gamma(x, y)$, for every $\gamma \in \Gamma$, denoting that an edge labelled γ runs from x to y, and
3. $x \in X$, denoting that x is an element of X.

The formulas are built from the atomic formulas using the connectives $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation), and $\rightarrow$ (implication), as usual. We can quantify both node variables and node-set variables, using the quantifiers $\exists$ and $\forall$. The quantifiers and negation bind more strongly than the binary connectives do.

For every n, the set of MSO formulas over (Σ, Γ) with n free node variables and no free set variables is denoted $\mathrm{MSOL}_n(\Sigma, \Gamma)$. Because we are predominantly interested in trees, an MSO formula over $(\Sigma, \mathrm{rks}(\Sigma))$ will also simply be called an MSO formula over Σ (where Σ is an operator alphabet). Also, $\mathrm{MSOL}(\Sigma, \mathrm{rks}(\Sigma))$ and $\mathrm{MSOL}_n(\Sigma, \mathrm{rks}(\Sigma))$ will be abbreviated to $\mathrm{MSOL}(\Sigma)$ and $\mathrm{MSOL}_n(\Sigma)$, respectively.

Semantics

For a closed formula $\phi \in \mathrm{MSOL}_0(\Sigma, \Gamma)$ and a graph $G \in \mathrm{GR}(\Sigma, \Gamma)$, we write $G \models \phi$ if G satisfies ϕ.

Given a graph G, a *valuation* b is a function that assigns to each node variable an element of V_G, and to each node-set variable a subset of V_G. We write $(G, b) \models \phi$, if ϕ holds in G, where the free variables of ϕ are assigned values according to the valuation b.

If a formula ϕ has free variables, say, x, X, y and no others, we also write $\phi(x, X, y)$. If it is not ambiguous, we write $(G, u, U, v) \models \phi(x, X, y)$ for $(G, [x \mapsto u, X \mapsto U, y \mapsto v]) \models \phi(x, X, y)$, listing the values for the variables in alphabetical order of the variables (capital letters come immediately after their lower-case counterparts).

An MSO formula defines a relation on the nodes of any graph.

Definition 1.43. *Given a graph $G \in \mathrm{GR}(\Sigma, \Gamma)$, a formula $\phi(x_1, \ldots, x_m) \in \mathrm{MSOL}_m(\Sigma, \Gamma)$ defines a relation*

$$R_G(\phi) = \{(u_1, \ldots, u_m) \in V_G^m \mid (G, u_1, \ldots, u_m) \models \phi(x_1, \ldots, x_m)\}.$$

Associated with an MSO formula are its graph and tree semantics.

Definition 1.44. *Let Σ, Γ be alphabets, $m \geq 0$, and $\phi(x_1, \ldots, x_m) \in \mathrm{MSOL}_m(\Sigma, \Gamma)$. The* graph semantics *of $\phi(x_1, \ldots, x_k)$ is*

$$L_{\mathrm{GR}}(\phi) = \{(G, u_1, \ldots, u_m) \mid G \in \mathrm{GR}(\Sigma, \Gamma), u_1, \ldots, u_m \in V_G, \text{ and } (G, u_1, \ldots, u_m) \models \phi(x_1, \ldots, x_m)\},$$

and, if Σ is an operator alphabet and $\Gamma = \mathrm{rks}(\Sigma)$, the tree semantics *of $\phi(x_1, \ldots, x_k)$ is*

$$L(\phi) = L_{\mathrm{GR}}(\phi) \restriction T_\Sigma.$$

For any Σ, Γ, a graph language $L \subseteq \mathrm{GR}(\Sigma, \Gamma)$ is called MSO definable if there is a closed formula ϕ such that $L_{\mathrm{GR}}(\phi) = L$. Analogously, for any Σ, a tree language $L \subseteq T_\Sigma$ is called MSO definable if there is a closed formula ϕ such that $L(\phi) = L$.

The set of all MSO definable tree languages is denoted MSOT. It is equal to the set of regular tree languages.

Proposition 1.45. *[Don70, TW68]*

$$\mathrm{MSOT} = \mathrm{REGT}$$

Because the proof of this lemma is constructive, from Proposition 1.26 we can conclude the following.

Proposition 1.46. *The following are decidable for any formula $\phi \in \mathrm{MSOL}_0(\Sigma)$:*

1. *Membership: $t \in L(\phi)$,*
2. *ϕ is a contradiction: $L(\phi) = \emptyset$, and*
3. *ϕ is a tautology: $L(\phi) = T_\Sigma$.*

More Notation

We use syntactic substitution on formulas, in the same manner as we did for terms in Section 1.3. We will also abbreviate substitution. For example, if ϕ has free variables x, y, we write $\phi(z, y)$ for $\phi[x \mapsto z, y \mapsto y]$, listing the substitutes in alphabetical order of the substituted variables.

If $A = \{a_1, \ldots, a_n\}$ is a fixed set and $\phi[a \mapsto a_j]$ is an MSO formula for all j, we write $\bigvee_{a \in A} \phi$, or $\exists a \in A : \phi$, for $\phi[a \mapsto a_1] \vee \cdots \vee \phi[a \mapsto a_n]$. Also, we write $\bigwedge_{a \in A} \phi$, or $\forall a \in A : \phi$ for $\phi[a \mapsto a_1] \wedge \cdots \wedge \phi[a \mapsto a_n]$. For example, $\exists a \in \{\gamma, \delta\} \, \mathrm{edg}_a(x, y)$ denotes $\mathrm{edg}_\gamma(x, y) \vee \mathrm{edg}_\delta(x, y)$.

We use some more abbreviations of formulas in $\mathrm{MSOL}(\Sigma, \Gamma)$, listed in Table 1.1. The meaning of most of them is obvious. The abbreviation $x = y \cdot k$, for a constant k means that x is the kth child of y if $k > 0$, and $x = y$ if $k = 0$. If $(G, U) \models \mathrm{closed}_{\phi(x,y)}(X)$, then U is closed under $R_G(\phi)$. The formula $\mathrm{path}_\gamma(x, y)$ states that x is connected to y through

abbreviation	**stands for**
$\phi \leftrightarrow \psi$	$(\phi \to \psi) \wedge (\psi \to \phi)$
$x = y$	$\forall X : (x \in X \leftrightarrow y \in X)$
$x \neq y$	$\neg(x = y\ )$
false	$\exists x : x \neq x$
true	$\neg$ false
$\exists! x : \phi$	$\exists x(\phi \wedge \forall y(\phi[x \mapsto y] \to y = x))$
$x = y \cdot k$	$x = y$ if $k = 0$, and $\mathrm{edg}_k(y, x)$ if $k \in \mathbb{N}_+$
$\mathrm{closed}_{\phi(x,y)}(X)$	$\forall x, y\Big((x \in X \wedge \phi(x, y)) \to y \in X\Big)$
$\mathrm{udg}_\gamma(x, y)$	$\mathrm{edg}_\gamma(x, y) \vee \mathrm{edg}_\gamma(y, x)$ (undirected edge)
$\mathrm{edg}(x, y)$	$\exists \gamma \in \Gamma : \mathrm{edg}_\gamma(x, y)$
$\mathrm{udg}(x, y)$	$\exists \gamma \in \Gamma : \mathrm{udg}_\gamma(x, y)$
$\mathrm{path}(x, y)$	$\forall X : ((x \in X \wedge \mathrm{closed}_{\mathrm{edg}(x,y)}(X)) \to y \in X)$
$\mathrm{path}_\gamma(x, y)$	$\forall X : ((x \in X \wedge \mathrm{closed}_{\mathrm{edg}_\gamma(x,y)}(X)) \to y \in X)$
$\mathrm{acpath}(x, y)$	$\exists X : (x \in X \wedge \forall z \in X(z = y \vee \exists z' \in X : \mathrm{edg}(z, z')))$
$\mathrm{upath}(x, y)$	$\forall X : ((x \in X \wedge \mathrm{closed}_{\mathrm{udg}(x,y)}(X)) \to y \in X)$
$\mathrm{root}(x)$	$\neg\exists y : \mathrm{edg}(y, x)$
$\mathrm{leaf}(x)$	$\neg\exists y : \mathrm{edg}(x, y)$

Table 1.1: Abbreviations for MSO

a path of edges with label γ only. In an acyclic graph, acpath is equivalent to path. The abbreviations $\mathrm{root}(x)$ and $\mathrm{leaf}(x)$ mean that x is a root (no incoming edges) or leaf (no outgoing edges) respectively.

Transitive, Reflexive Closure

For any formula $\phi(x, y)$, we define the formula

$$\phi^*(x, y) = \forall X((x \in X \wedge \mathrm{closed}_{\phi(x,y)}(X)) \to y \in X).$$

The relation defined by $\phi^*(x, y)$ defines the transitive, reflexive closure of the relation defined by $\phi(x, y)$ ([Cou90]).

Lemma 1.47. *Let Σ, Γ be alphabets, $G \in \mathrm{GR}(\Sigma, \Gamma)$, and $\phi(x, y) \in \mathrm{MSOL}_2(\Sigma, \Gamma)$. Then, $R_G(\phi^*) = R_G(\phi)^*$.*

Proof. First we prove $(u, v) \in R_G(\phi^*) \Rightarrow (u, v) \in R_G(\phi)^*$, and then we prove $(u, v) \in R_G(\phi)^* \Rightarrow (u, v) \in R_G(\phi^*)$.

- Suppose $(G, u, v) \models \forall X : ((x \in X \wedge \mathrm{closed}_{\phi(x,y)}(X)) \to y \in X)$. The set $U = \{w \in V_G \mid (u, w) \in R_G(\phi)^*\}$ is closed under $R_G(\phi)$. So, since $u \in U$, it follows that $v \in U$, and thus $(u, v) \in R_G(\phi)^*$.

- Let $(u, v) \in R_G(\phi)^*$. We have to prove $(G, u, v) \models \forall X : ((x \in X \wedge \text{closed}_{\phi(x,y)}(X)) \rightarrow y \in X)$. Take a $U \subseteq V_G$, with $u \in U$ and U closed under $R_G(\phi)$. Because $(u, v) \in R_G(\phi)^*$, there are $u_1, \ldots, u_n$, with $u_1 = u$, $u_n = v$, and $(u_i, u_{i+1}) \in R_G(\phi)$. Because U is closed under $R_G(\phi)$, and $u_1 \in U$, by induction $u_2, \ldots, u_n = v \in U$.

□

Chapter 2

Tree-Node Languages

We can use MSO formulas with one free variable to select nodes of a tree. For the same purpose, we can also use an attribute grammar with a designated boolean attribute δ. Given a tree t, the attribute grammar recognizes those nodes v of t that have $\mathrm{dec}_t(\langle \delta, v \rangle) =$ true. These two methods will turn out to be equivalent when we restrict the attribute grammar to have finite semantic domains.

In Section 2.5 we will go into the complexity of calculating whether an MSO formula with free variables holds for given tree and nodes, and the complexity of computing the relation defined by an MSO formula with free variables, given a tree.

2.1 Definitions

First we define tree-node languages, i.e., sets of tree nodes.

Definition 2.1. *Let Σ be an operator alphabet. A* tree-node language *T over Σ is a set $T \subseteq \{(t, v) \mid t \in T_\Sigma$ and $v \in V_t\}$.*

Tree-node languages can be *defined by* an MSO formula with one free variable. According to Definition 1.44, for an operator alphabet Σ, a formula $\phi(x) \in \mathrm{MSOL}_1(\Sigma)$ defines the tree-node language $L(\phi(x)) = \{(t, v) \mid t \in T_\Sigma,\ v \in V_t$, and $(t, v) \models \phi(x)\}$.

The set of all tree-node languages T, for which there is an operator alphabet Σ and a formula $\phi(x) \in \mathrm{MSOL}_1(\Sigma)$ such that $L(\phi(x)) = T$, is the set of MSO-*definable tree-node languages*, denoted MSO-TN.

Tree-node languages can also be *recognized by* an attribute grammar with finite semantic domains. A node-recognizing attribute grammar has a designated attribute δ with $W(\delta) = \mathbb{B}$ (a 'boolean attribute'). It recognizes the tree nodes v of a tree t with $\langle \delta, v \rangle_t =$ true.

Definition 2.2. *Let Σ be an operator alphabet, G a weakly non-circular attribute grammar over Σ with finite semantic domains, and δ a boolean attribute of G. Then, the* node-recognizing attribute grammar *(G, δ) recognizes the tree-node language*

$$L(G, \delta) = \{(t, v) \mid t \in T_\Sigma, v \in V_t \text{ and } \langle \delta, v \rangle_{G,t} = \text{true}\}.$$

The set of all tree-node languages T for which there is a node-recognizing attribute grammar (G, δ), such that $L(G, \delta) = T$, is the set of all AG*-recognizable tree-node languages, denoted* AG-TN*.*

It should be clear that it makes no difference whether we use weakly non-circular or classically non-circular in the above definition.

We claim that MSO-TN = AG-TN and will proceed to prove this in the next sections. First we introduce some more notation.

We define how to mark nodes in a tree. Let $B_1 = \{1\}$. The operator alphabet $\Sigma \cup (\Sigma \times B_1)$ is the alphabet for which $\mathrm{rk}_{\Sigma\cup(\Sigma\times B_1)}(\sigma) = \mathrm{rk}_{\Sigma\cup(\Sigma\times B_1)}(\langle\sigma, 1\rangle) = \mathrm{rk}_\Sigma(\sigma)$.

We will use $\Sigma\cup(\Sigma\times B_1)$ to mark certain nodes in a tree: For any tree t over $\Sigma\cup(\Sigma\times B_1)$, a node v is called *marked* if its label is $\langle\sigma, 1\rangle$ for some $\sigma \in \Sigma$.

For every tree $t \in \Sigma \cup (\Sigma \times B_1)$, the underlying tree $\mathrm{und}(t)$ over Σ is obtained by replacing every label $\langle\sigma, 1\rangle$ in t by its first component σ, and leaving the other labels unchanged.

There is an obvious bijection between tree nodes and certain marked trees.

Definition 2.3. *For every tree $t \in T_\Sigma$ and node v of t, $\mathrm{mark}(t, v) \in T_{\Sigma\cup(\Sigma\times B_1)}$ is the corresponding marked tree, defined as follows*

$$\begin{aligned} \mathrm{mark}(t, v) &= (V_t, E_t, \mathrm{nlab}', \mathrm{elab}_t), \quad \textit{where} \\ \mathrm{nlab}' &= \lambda w \in V_t. \begin{cases} \mathrm{nlab}_t(w) & \textit{if } w \neq v \\ \langle \mathrm{nlab}_t(w), 1\rangle & \textit{if } w = v \end{cases} \end{aligned}$$

So, $\mathrm{mark}(t, v)$ has exactly one marked node: v. We use labels σ instead of $\langle\sigma, 0\rangle$, because now any subtree of $\mathrm{mark}(t, v)$ rooted in a node u that does not have a marked node is equal to the subtree of t rooted in u.

Definition 2.4. *Let T be a tree-node language over Σ. The* test language *of T is a tree language* $\mathrm{test}(T)$ *over $\Sigma \cup (\Sigma \times B_1)$:*

$$\mathrm{test}(T) = \{t \in T_{\Sigma\cup(\Sigma\times B_1)} \mid \forall v \in V_t : \textit{ if } \exists\sigma \in \Sigma : \mathrm{nlab}(v) = \langle\sigma, 1\rangle, \textit{ then } (\mathrm{und}(t), v) \in T\}$$

Since $\mathrm{und}(\mathrm{mark}(t, v)) = t$ for all t and v, obviously $(t, v) \in T$ iff $\mathrm{mark}(t, v) \in \mathrm{test}(T)$. We will use this observation twice in the following sections.

We prove MSO-TN = AG-TN in two parts. In Section 2.2 we will prove that for any MSO formula $\phi(x)$ that defines a tree-node language T, there is a closed MSO formula ψ that defines the test language of T, and vice-versa. This is equivalent to $T \in$ MSO-TN iff $\mathrm{test}(T) \in$ MSOT. Next, in Section 2.3 we prove that for any node-recognizing attribute grammar that recognizes a tree-node language T there is a tree automaton that recognizes the test language of T, or, in other words, that $T \in$ AG-TN iff $\mathrm{test}(T) \in$ REGT. Because we know that MSOT = REGT (Proposition 1.45), we conclude in Section 2.4 that MSO formulas with one free variable and node-recognizing attribute grammars recognize the same languages, i.e., MSO-TN = AG-TN.

2.2 MSO Definable Tree-Node Languages and Test Languages

A tree-node language is MSO definable if and only if its test language is. This is proven by rewriting the formula defining a language into a formula defining the corresponding test language and vice-versa.

Let Σ be an operator alphabet, and let T be a tree-node language over Σ. Let $\phi(x)$ be an MSO formula over Σ, with one free node variable x, such that $T = L(\phi(x))$. From $\phi(x)$ we can obtain the closed MSO formula ψ over $\Sigma \cup (\Sigma \times B_1)$:

$$\psi = \forall x((\exists \sigma \in \Sigma : \mathrm{lab}_{\langle \sigma,1 \rangle}(x)) \rightarrow \phi'(x)),$$

where the MSO formula $\phi'(x)$ over $\Sigma \cup (\Sigma \times B_1)$ is obtained from $\phi(x)$ by substituting $(\mathrm{lab}_\sigma(y) \vee \mathrm{lab}_{\langle \sigma,1 \rangle}(y))$ for $\mathrm{lab}_\sigma(y)$, for every $\sigma \in \Sigma$ and node variable y.

Clearly, $(t, v) \models \phi'(x)$ iff $(\mathrm{und}(t), v) \models \phi(x)$, and therefore $L(\psi) = \mathrm{test}(T)$.

Example 2.5. Let $\Sigma = \Sigma_0 \cup \Sigma_2$, with $\Sigma_0 = \{*, \#\}$, and $\Sigma_2 = \{\sigma\}$. Now consider the following MSO formula from $\mathrm{MSOL}_1(\Sigma)$.

$$\phi(x) = \exists! y : \mathrm{lab}_*(y) \wedge \exists y(\mathrm{path}(x, y) \wedge \mathrm{lab}_*(y)).$$

For a tree t and a node u of t, $(t, u) \models \phi(x)$ iff t has exactly one leaf labelled $*$, and u is on the path from the root of t to that leaf. The tree-node language defined by $\phi(x)$ is

$$L(\phi(x)) = \{(t, u) \mid t \in T_\Sigma, (t, u) \models \phi(x)\},$$

which is the language consisting of all tuples (t, u), where t is a tree with exactly one leaf v labelled $*$, and u is on the path from $\mathrm{root}(t)$ to v. The test language corresponding to $L(\phi)$ is

$$\mathrm{test}(L(\phi)) = \{t \in T_{\Sigma \cup (\Sigma \times B_1)} \mid \forall v \in V_t : \text{ if } \exists \sigma \in \Sigma : \mathrm{nlab}(v) = \langle \sigma, 1 \rangle, \text{ then } (\mathrm{und}(t), v) \models \phi\},$$

which is the language consisting of all trees $t \in T_{\Sigma \cup (\Sigma \times B_1)}$, that have exactly one leaf labelled $*$, and in which only on nodes on the path from the root to that leaf are marked, and all trees from $T_{\Sigma \cup (\Sigma \times B_1)}$ that have no marked nodes at all (i.e., T_Σ). The closed MSO formula that recognizes this language is

$$\psi = \forall x\Big(\exists \sigma \in \Sigma : \mathrm{lab}_{\langle \sigma,1 \rangle}(x) \rightarrow \Big(\exists! y(\mathrm{lab}_*(y) \vee \mathrm{lab}_{\langle *,1 \rangle}(y)) \wedge \exists y(\mathrm{path}(x, y) \wedge (\mathrm{lab}_*(y) \vee \mathrm{lab}_{\langle *,1 \rangle}(y)))\Big)\Big).$$

□

We now prove that for any closed MSO formula ψ that defines the language test(T) for some tree-node language T, there is a formula $\phi(x)$ that defines T. We do this in a similar manner. Let T be a tree-node language over Σ and let ψ be a closed MSO formula over $\Sigma \cup (\Sigma \times B_1)$ such that $L(\psi) = \text{test}(T)$. Assuming x is not a variable of ψ, we construct a formula $\phi(x)$ over Σ from ψ, by replacing, for each node variable y,

1. all occurrences of $\text{lab}_{\langle\sigma,1\rangle}(y)$ by $(\text{lab}_\sigma(y) \wedge y = x)$, and
2. all occurrences of $\text{lab}_\sigma(y)$ by $(\text{lab}_\sigma(y) \wedge y \neq x)$.

Clearly, $(t, v) \in L(\phi(x))$ iff $\text{mark}(t, v) \in L(\psi)$, and hence $L(\phi(x)) = T$.
From these two observations we conclude the following result.

Proposition 2.6. *For every operator alphabet Σ and tree-node language T over Σ,*

$$T \in \text{MSO-TN} \iff \text{test}(T) \in \text{MSOT}\,.$$

2.3 AG Recognizable Tree-Node Languages and Test Languages

In this section we prove that a node language is AG-recognizable iff its test language can be recognized by a tree automaton. First we prove that $T \in$ AG-TN implies test$(T) \in$ REGT and then we prove that $T \in$ AG-TN is implied by test$(T) \in$ REGT.

If a Tree-Node Language is AG Recognizable, its Test Language is Regular

Let (G, δ) be a node-recognizing attribute grammar over Σ, and let $G = (\Sigma, \Omega, (S, I, W), R)$. Note that by Definition 2.2 we may assume G to be unconditional and without α_{mean}. Let $T = L(G, \delta)$ be the node language recognized by the node-recognizing attribute grammar. We want to construct a tree automaton $\mathcal{A}_{(G,\delta)} = (Q, F)$ over $\Sigma \cup (\Sigma \times B_1)$, such that $L(\mathcal{A}_{(G,\delta)}) = \text{test}(T)$.

The tree automaton will be a non-deterministic, bottom-up one. Acting on t, it simulates the attribute grammar on und(t), by calculating the decoration on its way up, in a non-deterministic way. In the construction we use the letter d to denote the states of the automaton, which are functions encoding a top-less decoration of a node of the tree (see Definition 1.36). The automaton accepts as final states all decorations that satisfy the root rules. The set of states and the set of final states are

$$Q = \{d : A_G \to \cup\Omega \mid \forall \alpha \in A_G : d(\alpha) \in W(\alpha)\},$$
$$F = \{d \in Q \mid \forall \langle \alpha_0, 0\rangle = f(\langle \alpha_1, 0\rangle, \ldots, \langle \alpha_r, 0\rangle) \in R_{\text{root}} : d(\alpha_0) = f(d(\alpha_1), \ldots, d(\alpha_r))\}.$$

The automaton has the following operations. For $\sigma \in \Sigma_k$, it makes sure that all rules at σ are satisfied:

$$\sigma_Q = \{((d_1, \ldots, d_k), d_0) \mid \forall \langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_r, i_r \rangle) \in R(\sigma) : \\ d_{i_0}(\alpha_0) = f(d_{i_1}(\alpha_1), \ldots, d_{i_r}(\alpha_r))\},$$

and, for all $\langle \sigma, 1 \rangle \in \Sigma \times B_1$, it does the same, but at the same time makes sure that the designated attribute is set:

$$\langle \sigma, 1 \rangle_Q = \{((d_1, \ldots, d_k), d_0) \mid \forall \langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_r, i_r \rangle) \in R(\sigma) : \\ d_{i_0}(\alpha_0) = f(d_{i_1}(\alpha_1), \ldots, d_{i_r}(\alpha_r)) \text{ and } d_0(\delta) = \text{true}\}.$$

Please note that Q is indeed finite, because G has finite semantic domains. For any marked tree t, the automaton $\mathcal{A}_{(G,\delta)}$ tries to find a top-less decoration of $\text{und}(t)$, in which $\langle \delta, v \rangle_t = \text{true}$ for every node v that is marked in t. It does so by guessing decorations of the nodes of the underlying tree. Moving up in the tree, it makes sure that all internal rules of G are satisfied. The tree automaton can either succeed, in which case it has found the unique decoration of $\text{und}(t)$ which satisfies this condition and the root rules, or fail, in which case there is no such decoration. The decoration of a node v is encoded as a function d, such that $d(\alpha)$ represents $\langle \alpha, v \rangle$.

Lemma 2.7. $L(\mathcal{A}_{(G,\delta)}) = \text{test}(T)$.

Proof. Let t be a marked tree. The top-less decorations are correct: $d \in \text{val}_{\mathcal{P}(Q)}(t)$ iff there is a top-less decoration dec of $\text{und}(t)$, with for every $\alpha \in A_G$, $d(\alpha) = \text{dec}(\langle \alpha, \text{root}(t) \rangle)$, and $\text{dec}(\langle \delta, u \rangle) = \text{true}$ for every marked node $u \in V_t$. This can easily be seen by induction on t, using Lemma 1.37.

Now, $\text{val}_{\mathcal{P}(Q)}(t) \cap F \neq \emptyset$ iff for every marked node $u \in V_t$, the unique decoration $\text{dec}_{G,\text{und}(t)}$ of $\text{und}(t)$ by G has $\text{dec}_{G,\text{und}(t)}(\langle \delta, u \rangle) = 1$, or, equivalently, $(\text{und}(t), u) \in L(G, \delta)$ for every marked node $u \in V_t$. □

So, we have proved that the test language for an AG recognizable language is indeed regular.

Lemma 2.8. *For any operator alphabet Σ, and tree-node language T over Σ,*

$$T \in \text{AG-TN} \Longrightarrow \text{test}(T) \in \text{REGT}\,.$$

A Tree-Node Language is AG Recognizable if its Test Language is Regular

We will now show that if we have a test language $\text{test}(T)$, that is recognizable, we can find a node-recognizing attribute grammar that recognizes T.

Let Σ be an operator alphabet, and let T be a tree-node language over Σ. Let $\mathcal{A} = (Q, F)$ be a deterministic finite tree automaton over $\Sigma \cup (\Sigma \times B_1)$ with $L(\mathcal{A}) = \text{test}(T)$. We construct an attribute grammar $G_{\mathcal{A}} = (\Sigma, \Omega, (S, I, W), R)$ over Σ with designated attribute δ, such that $L(G_{\mathcal{A}}, \delta) = T$.

The attribute grammar has inherited attribute β, with semantic domain $\mathcal{P}(Q)$ and synthesized attributes α and δ, with semantic domain Q and $\mathbb{B}$ respectively. Formally, $\Omega = \{Q, \mathcal{P}(Q), \mathbb{B}\}$, $S = \{\alpha, \delta\}$, $I = \{\beta\}$, and W is defined as $W(\alpha) = Q$, $W(\beta) = \mathcal{P}(Q)$, and $W(\delta) = \mathbb{B}$.

Intuitively, for a tree t over T_Σ, α simulates the behavior of the tree automaton on t, assuming that no node is marked. Attribute β of a node v holds those states q, for which the tree automaton accepts the tree, assuming that it has reached v in state q and that the context of v contains no marked nodes. For every node $v \in V_t$, with label σ, $\text{mark}(t, v)$ is accepted by $\mathcal{A}$ if $\langle \sigma, 1 \rangle_Q$ applied to the α-values of its children yields a successful value, i.e., a value in β. Attribute δ of v is true if and only if this condition holds. The internal semantic rules for every $\sigma \in \Sigma_k$ are

$$\begin{aligned}
\langle \alpha, 0 \rangle &= \sigma_Q(\langle \alpha, 1 \rangle, \langle \alpha, 2 \rangle, \ldots, \langle \alpha, k \rangle) \\
\langle \beta, i \rangle &= \{q \in Q \mid \sigma_Q(\langle \alpha, 1 \rangle, \ldots, \underset{i}{q}, \ldots, \langle \alpha, k \rangle) \in \langle \beta, 0 \rangle\} \quad \text{for } 1 \leq i \leq k \\
\langle \delta, 0 \rangle &= \Big(\langle \sigma, 1 \rangle_Q(\langle \alpha, 1 \rangle, \langle \alpha, 2 \rangle, \ldots, \langle \alpha, k \rangle) \in \langle \beta, 0 \rangle\Big)
\end{aligned}$$

We have one semantic root rule:

$$\langle \beta, 0 \rangle = F.$$

The attribute grammar $G_{\mathcal{A}}$ is weakly non-circular, because it is non-circular in the classical sense [Knu68]. In fact, it is a two-pass AG [Boc76], with decomposition in phases $A = (\{\alpha\}, \{\beta, \delta\})$.

We want to prove that $G_{\mathcal{A}}$ is correct, i.e., given a tree t, a node v is recognized by $(G_{\mathcal{A}}, \delta)$ iff $(t, v) \in T$. To this aim we have the following lemmata.

Lemma 2.9. *Let t be a tree over Σ. For every node v of t,*

$$\langle \alpha, v \rangle = \text{val}_Q(\text{sub}_t(v)).$$

Proof. We prove this by induction on the depth of $\text{sub}_t(v)$.
Base: v is a leaf, so $\text{sub}_t(v) = \sigma$ for some $\sigma \in \Sigma_0$. Then

$$\begin{array}{rcll}
\langle \alpha, v \rangle & = & \sigma_Q & \text{[by definition of } \alpha\text{]} \\
& = & \text{val}_Q(\sigma) & \text{[by definition of val]} \\
& = & \text{val}_Q(\text{sub}_t(v)) &
\end{array}$$

Induction: Let $\text{sub}_t(v) = \sigma t_1 t_2 \cdots t_k$, where $k = \text{rk}(\sigma)$. By induction, $\langle \alpha, y \rangle = \text{val}_Q(\text{sub}_t(y))$ for every i and every node y of t_i. So in particular $\langle \alpha, v \cdot i \rangle = \text{val}_Q(\text{sub}_t(v \cdot i)) = \text{val}_Q(t_i)$.

Hence,

$$\begin{array}{rcll} \langle \alpha, v \rangle & = & \sigma_Q(\langle \alpha, v \cdot 1 \rangle, \ldots, \langle \alpha, v \cdot k \rangle) & \text{[by definition of } \alpha\text{]} \\ & = & \sigma_Q(\mathrm{val}_Q(t_1), \ldots, \mathrm{val}_Q(t_k)) & \text{[follows from the induction hypothesis]} \\ & = & \mathrm{val}_Q(\sigma t_1 \ldots t_k) & \text{[by definition of val]} \\ & = & \mathrm{val}_Q(\mathrm{sub}_t(v)) & \end{array}$$

□

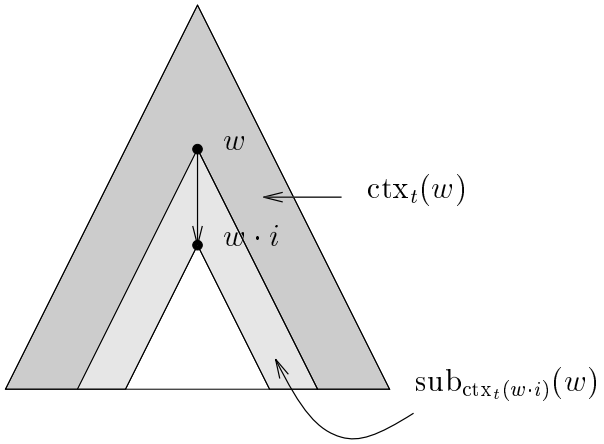

Figure 2.1: Contexts and subtrees

The β attribute of a node v of t holds exactly those states the subtree rooted in v may yield, so that $\mathrm{val}_Q(t) \in F$, assuming the context of v holds no marked nodes. In other words,

Lemma 2.10. *Let t be a tree over Σ. For every node v of t,*

$$\langle \beta, v \rangle = \{q \in Q \mid \mathrm{val}_Q(\mathrm{ctx}_t(v))[\xi \mapsto q] \in F\}$$

Proof. We prove this by induction on the depth of v.
Base: The depth is 0, so $v = \mathrm{root}(t)$. Then,

$$\begin{array}{rcll} \mathrm{val}_Q(\mathrm{ctx}_t(v))[\xi \mapsto q] \in F & \Leftrightarrow & \mathrm{val}_Q(\xi)[\xi \mapsto q] \in F & [v = \mathrm{root}(t)] \\ & \Leftrightarrow & q \in F & \text{[by definition of val]} \\ & \Leftrightarrow & q \in \langle \beta, \mathrm{root}(t) \rangle & \text{[because of the root rule]} \\ & \Leftrightarrow & q \in \langle \beta, v \rangle & \end{array}$$

Induction: $v = w \cdot i$ for some node w of t and some $i \in [1, k]$, where $k = \mathrm{rk}(w)$. For better understanding, please note that

$$\mathrm{ctx}_t(w \cdot i) = \mathrm{ctx}_t(w)[\xi \mapsto \mathrm{sub}_{\mathrm{ctx}_t(w \cdot i)}(w)], \text{ and} \tag{2.1}$$

$$\mathrm{sub}_{\mathrm{ctx}_t(w \cdot i)}(w) = \mathrm{nlab}_t(w)\, \mathrm{sub}_t(w \cdot 1) \cdots \underset{i}{\xi} \cdots \mathrm{sub}_t(w \cdot k), \tag{2.2}$$

(see Figure 2.1), and thus,

$$\mathrm{val}_Q(\mathrm{sub}_{\mathrm{ctx}_t(w\cdot i)}(w))[\xi \mapsto q]$$

=

$$\mathrm{val}_Q(\mathrm{nlab}_t(w)\,\mathrm{sub}_t(w\cdot 1)\cdots\underset{i}{\xi}\cdots\mathrm{sub}_t(w\cdot k))[\xi \mapsto q]$$

= [by definition of val, and because the subtrees of w do not contain label ξ]

$$(\mathrm{nlab}_t(w))_Q(\mathrm{val}_Q(\mathrm{sub}_t(w\cdot 1)),\ldots,\underset{i}{q},\ldots,\mathrm{val}_Q(\mathrm{sub}_t(w\cdot k)))$$

= [by Lemma 2.9]

$$(\mathrm{nlab}_t(w))_Q(\langle\alpha, w\cdot 1\rangle,\ldots,\underset{i}{q},\ldots,\langle\alpha, w\cdot k\rangle). \tag{2.3}$$

The proof follows.

$$\mathrm{val}_Q(\mathrm{ctx}_t(w\cdot i))[\xi \mapsto q] \in F$$

$\iff$ [by Equation 2.1]

$$\mathrm{val}_Q(\mathrm{ctx}_t(w)[\xi \mapsto \mathrm{sub}_{\mathrm{ctx}_t(w\cdot i)}(w)])[\xi \mapsto q] \in F$$

$\iff$ [by Proposition 1.18]

$$\mathrm{val}_Q(ctx_t(w))[\xi \mapsto \mathrm{val}_Q(\mathrm{sub}_{\mathrm{ctx}_t(w\cdot i)}(w))[\xi \mapsto q]] \in F$$

$\iff$ [by Equation 2.3]

$$\mathrm{val}_Q(\mathrm{ctx}_t(w))[\xi \mapsto (\mathrm{nlab}_t(w))_Q(\langle\alpha, w\cdot 1\rangle,\ldots,\underset{i}{q},\ldots,\langle\alpha, w\cdot k\rangle)] \in F$$

$\iff$ [by induction]

$$(\mathrm{nlab}_t(w))_Q(\langle\alpha, w\cdot 1\rangle,\ldots,\underset{i}{q},\ldots,\langle\alpha, w\cdot k\rangle) \in \langle\beta, w\rangle$$

$\iff$ [by definition of β]

$$q \in \langle\beta, w\cdot i\rangle$$

□

The following lemma proves the correctness of the attribute grammar. It states that δ of a node v is true iff $\mathcal{A}$ succeeds on $\mathrm{mark}(t, v)$.

Lemma 2.11. *Let t be a tree over Σ, v a node of t, and $k = \mathrm{rk}(v)$. Then*

$$\langle \delta, v \rangle = \mathrm{true} \Leftrightarrow \mathrm{val}_Q(\mathrm{mark}(t, v)) \in F$$

Proof.

$$\langle \delta, v \rangle = \mathrm{true}$$

$\Longleftrightarrow$ [by definition of δ]

$$\langle \mathrm{nlab}_t(v), 1 \rangle_Q(\langle \alpha, v \cdot 1 \rangle, \ldots, \langle a, v \cdot k \rangle) \in \langle \beta, v \rangle$$

$\Longleftrightarrow$ [by Lemma 2.10]

$$\mathrm{val}_Q(\mathrm{ctx}_t(v))\Big[\xi \mapsto \langle \mathrm{nlab}_t(v), 1 \rangle_Q(\langle \alpha, v \cdot 1 \rangle, \ldots, \langle \alpha, v \cdot k \rangle)\Big] \in F$$

$\Longleftrightarrow$ [by Lemma 2.9]

$$\mathrm{val}_Q(\mathrm{ctx}_t(v))\Big[\xi \mapsto \langle \mathrm{nlab}_t(v), 1 \rangle_Q(\mathrm{val}_Q(\mathrm{sub}_t(v \cdot 1)), \ldots, \mathrm{val}_Q(\mathrm{sub}_t(v \cdot k)))\Big] \in F$$

$\Longleftrightarrow$ [by definition of val]

$$\mathrm{val}_Q(\mathrm{ctx}_t(v))\Big[\xi \mapsto \mathrm{val}_Q(\langle \mathrm{nlab}_t(v), 1 \rangle(\mathrm{sub}_t(v \cdot 1) \ldots \mathrm{sub}_t(v \cdot k)))\Big] \in F$$

$\Longleftrightarrow$ [by Proposition 1.18]

$$\mathrm{val}_Q\Big(\mathrm{ctx}_t(v)[\xi \mapsto \langle \mathrm{nlab}_t(v), 1 \rangle(\mathrm{sub}_t(v \cdot 1) \ldots \mathrm{sub}_t(v \cdot k))]\Big) \in F$$

$\Longleftrightarrow$

$$\mathrm{val}_Q(\mathrm{mark}(t, v)) \in F$$

□

From this lemma it follows that $(t, v) \in L(G_{\mathcal{A}}, \delta)$ iff $\mathrm{mark}(t, v) \in \mathrm{test}(T)$, so $L(G_{\mathcal{A}}, \delta) = T$. So now we can conclude that if a tree-node language is AG recognizable, its test language is indeed regular.

Lemma 2.12. *For any operator alphabet Σ, and tree-node language T over Σ,*

$$\mathit{test}(T) \in \textsc{regt} \Longrightarrow T \in \textsc{ag-tn}\,.$$

And, combining this with Lemma 2.8, we get the following equivalence.

Proposition 2.13. *For any operator alphabet Σ, and tree-node language T over Σ,*

$$T \in \textsc{ag-tn} \Longleftrightarrow \mathrm{test}(T) \in \textsc{regt}\,.$$

2.4 MSO Definable and AG Recognizable Tree-Node Languages are the Same

We know that for any operator alphabet Σ and tree-node language T over Σ,

1. $T \in$ MSO-TN $\iff$ test$(T) \in$ MSOT, from Proposition 2.6, and
2. $T \in$ AG-TN $\iff$ test$(T) \in$ REGT, from Proposition 2.13.

Now, because MSOT = REGT (Proposition 1.45), we can conclude that MSO definable tree-node languages are the same as AG recognizable tree-node languages.

Theorem 2.14.

$$\text{AG-TN} = \text{MSO-TN}$$

This has the practical consequence that for any boolean attribute δ of a weakly non-circular attribute grammar G with finite semantic domains, we can assume that there is an MSO formula $\phi(x)$ with one free node variable, such that for any tree t and node u,

$$\text{dec}_{G,t}(\langle \delta, u \rangle) = \text{true} \Leftrightarrow (t, u) \models \phi(x),$$

and vice-versa.

This result implies that we can use an MSO formula to test the value of any attribute in an attribute grammar with finite semantic domains.

Lemma 2.15. *Let G be a weakly non-circular attribute grammar with finite semantic domains. Let γ be an attribute of G with $W(\gamma) = \{a_1, \ldots, a_n\}$. For all i, there is an* MSO *formula $\phi_{\gamma=a_i}(x)$, with $(t, u) \models \phi_{\gamma=a_i}(x)$ iff $\text{dec}_{G,t}(\langle \gamma, u \rangle) = a_i$.*

Proof. For all $i \in [1, n]$, we can make a boolean attribute δ_i, such that for all u, $\langle \delta_i, u \rangle =$ true iff $\langle \gamma, u \rangle = a_i$. Hence, we can construct an MSO formula $\phi_{\gamma=a_i}(x)$, with $(t, u) \models \phi_{\gamma=a_i}(x)$ iff $\text{dec}_{G,t}(\langle \gamma, u \rangle) = a_i$. □

2.5 Complexity

In this section we address two points. First, suppose we have a fixed formula $\phi(x_1, \ldots, x_k)$ with $k \geq 0$ free node variables. What is the complexity of checking whether or not $(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k)$, for a given tree t and nodes $u_1, \ldots, u_k \in V_t$.

Second, for a fixed formula $\phi(x_1, \ldots, x_k)$ with $k \geq 1$ free node variables, what is the complexity of finding $R_t(\phi(x_1, \ldots, x_k))$, given a tree t (see Definition 1.43).

Basic Lemma

We generalize some of the notation and some of the results of the previous sections.

Definition 2.16. *Let Σ be an operator alphabet, and $k \geq 1$. The set B_k is equal to $\mathbb{B}^k \setminus \{0\}^k$. The operator alphabet $\Sigma \cup (\Sigma \times B_k)$ is the operator alphabet with, for any $\sigma \in \Sigma$, $\mathrm{rk}_{\Sigma \cup (\Sigma \times B_k)}(\sigma) = \mathrm{rk}_\Sigma(\sigma)$, and for all $\langle \sigma, b_1, \ldots, b_k \rangle \in \Sigma \times B_k$, $\mathrm{rk}_{\Sigma \cup (\Sigma \times \mathbb{B}^k)}(\langle \sigma, b_1, \ldots, b_k \rangle) = \mathrm{rk}_\Sigma(\sigma)$. Note that label $\langle \sigma, 0, \ldots, 0 \rangle$ is excluded.*

We can use this alphabet to attach k different marks to the labels of the nodes of a tree, cf. Definition 2.3.

Definition 2.17. *Let t be a tree over Σ, and let $(u_1, \ldots, u_k) \in V_t^k$. The marked tree $\mathrm{mark}(t, u_1, \ldots, u_k)$ over $\Sigma \cup (\Sigma \times B_k)$ is defined as follows.*

$$\mathrm{mark}(t, u_1, \ldots, u_k) = (V_t, E_t, \mathrm{nlab}', \mathrm{elab}_t),$$

where

$$\mathrm{nlab}' = \lambda w \in V_t. \begin{cases} \mathrm{nlab}_t(w) & \text{if } w \neq u_i \text{ for all } i \\ \langle \mathrm{nlab}_t(w), (w = u_1), \ldots, (w = u_k) \rangle & \text{otherwise.} \end{cases}$$

Recall that for any u, v, $(u = v)$ is equal to 1 (or true) if u is equal to v, and 0 (false) otherwise. We say that a node has mark j if its label is $\langle \sigma, b_1, \ldots, \underset{j}{1}, \ldots, b_k \rangle$ for some σ and b_i ($i \in [1, k]$). A node can have more than one mark. A node without any mark has a label σ.

We now generalize the results from Section 2.2. The following lemma states that we can move arbitrarily between free variables in the formula and marks in the tree.

Lemma 2.18. *For any $\phi(x_1, \ldots, x_k) \in \mathrm{MSOL}_k(\Sigma)$, and $j \in [1, k]$, there is an* MSO *formula $\psi(x_{j+1}, \ldots, x_k) \in \mathrm{MSOL}_{k-j}(\Sigma \cup (\Sigma \times B_j))$, such that*

$$(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k) \text{ iff } (\mathrm{mark}(t, u_1, \ldots, u_j), u_{j+1}, \ldots, u_k) \models \psi(x_{j+1}, \ldots x_k),$$

and vice versa.

Proof. We proceed to prove the lemma in two parts. First we show how to construct ψ from ϕ, and then we show how to construct ϕ from ψ.

- From an MSO formula $\phi(x_1, \ldots, x_k)$ we construct $\psi(x_{j+1}, \ldots, x_k)$ as follows:

  $$\psi(x_{j+1}, \ldots, x_k) = \forall x_1 \ldots x_j((\mathrm{marked}_1(x_1) \wedge \cdots \wedge \mathrm{marked}_j(x_j)) \rightarrow \phi'(x_1, \ldots, x_k)),$$

  where

  $$\mathrm{marked}_i(x) = \exists(\sigma, b_1, \ldots, b_j) \in \Sigma \times B_j : \mathrm{lab}_{\langle \sigma, b_1, \ldots, \underset{i}{1}, \ldots, b_j \rangle}(x),$$

and $\phi'(x_1, \ldots, x_k)$ is obtained from $\phi(x_1, \ldots, x_k)$ by replacing each occurrence of $\text{lab}_\sigma(z)$ by $(\text{lab}_\sigma(z) \vee \exists(b_1, \ldots, b_j) \in B_j : \text{lab}_{\langle\sigma, b_1, \ldots, b_j\rangle}(z))$, for all σ and z.

Clearly,

$(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k)$ iff $(\text{mark}(t, u_1, \ldots, u_j), u_{j+1}, \ldots, u_k) \models \psi(x_{j+1}, \ldots, x_k)$.

- The other way around is similar. Let us assume $x_1, \ldots, x_j$ are not variables of $\psi(x_{j+1}, \ldots, x_k)$. We construct $\phi(x_1, \ldots, x_k)$ by replacing, for each node variable y,

  1. all occurrences of $\text{lab}_\sigma(y)$ by $(\text{lab}_\sigma(y) \wedge \forall i \in [1, j] : y \neq x_i)$, and
  2. all occurrences of $\text{lab}_{\langle\sigma, b_1, \ldots, b_j\rangle}(y)$ by $(\text{lab}_\sigma(y) \wedge \forall i \in [1, j] : (b_i = 1 \rightarrow y = x_i))$.

  It should be obvious that

  $(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k)$ iff $(\text{mark}(t, u_1, \ldots, u_j), u_{j+1}, \ldots, u_k) \models \psi(x_{j+1}, \ldots, x_k)$.

□

Checking a Formula

We compute the complexity of checking a fixed formula on a given tree and nodes.

Theorem 2.19. *Let Σ be an operator alphabet, and let $\phi(x_1, \ldots, x_k) \in \text{MSOL}_k(\Sigma)$ be a fixed* MSO *formula with $k \geq 0$ free node variables. The complexity of checking whether or not $(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k)$, for a given tree t and nodes $u_1, \ldots, u_k \in V_t$ is $O(n)$.*

Proof. Please note that the size of the input (the tree together with k nodes) is of the same order as the size of the tree.

The tree t with the nodes $u_1, \ldots, u_k$ can be converted to a marked tree. This is a relabelling, which can be done in $O(n)$ time, where n is the size of the tree. By Lemma 2.18 (with $j = k$), the MSO formula $\phi(x_1, \ldots, x_k)$ with k free node variables can be converted to a closed MSO formula ψ over marked trees such that $(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k)$ iff $\text{mark}(t, u_1, \ldots, u_k) \models \psi$. Since MSOT = REGT, there is a finite deterministic tree automaton, that checks whether the marked tree satisfies ψ in $O(n)$ time, so the total time involved in checking the formula is $O(n)$. □

Finding all Node Sequences Satisfying a Formula

We determine the complexity of finding $R_t(\phi(x_1, \ldots, x_k))$, given a formula $\phi(x_1, \ldots, x_k)$.

Theorem 2.20. *Let Σ be an operator alphabet and let $\phi(x_1, \ldots, x_k) \in \text{MSOL}_k(\Sigma)$ be a fixed formula with $k \geq 1$ free node variables. The complexity of finding $R_t(\phi(x_1, \ldots, x_k))$, given a tree t, is $O(n^k)$.*

Proof. First we consider the problem for formulas with only one free node variable, i.e., $k = 1$. With the help of Theorem 2.14 we can transform $\phi(x_1)$ to a node-recognizing AG (G, δ), with $L(\phi(x_1)) = L(G, \delta)$, or, in other words, $(t, u) \models \phi(x_1)$ iff $\langle \delta, u \rangle_{G,t} = \text{true}$.

Since attribute evaluation takes linear time for a non-circular attribute grammar with finite semantic domains [Eng84], it takes linear time to find all v such that $(t, v) \models \phi(x_1)$.

Suppose $k > 1$. Using Lemma 2.18 (with $j = k - 1$), $\phi(x_1, \ldots, x_k)$ can be transformed into an MSO formula $\psi(x)$ over $\Sigma \cup (\Sigma \times B_{k-1})$ with one free node variable, such that $(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k)$ iff $(\text{mark}(t, u_1, \ldots, u_{k-1}), u_k) \models \psi(x)$.

Now, for any $u_1, \ldots, u_{k-1} \in V_t$, we can find $\text{mark}(t, u_1, \ldots, u_{k-1})$ in $O(n)$ time. We can then find all u_k such that $(\text{mark}(t, u_1, \ldots, u_{k-1}), u_k) \models \psi(x)$ in $O(n)$ time, by using an attribute grammar, as was shown above. There are $O(n^{k-1})$ possible combinations for $u_1, \ldots, u_{k-1}$, so the time needed to find all $u_1, \ldots, u_k$ such that $(\text{mark}(t, u_1, \ldots, u_{k-1}), u_k) \models \psi(x)$ is $O(n^k)$. □

Chapter 3

Tree-Node Relations

In this chapter we will consider graph-node and tree-node relations, i.e., binary relations between nodes in a graph or tree. We consider two ways of defining such relations. The first way is by MSO formulas with two free node variables. Two nodes are in the relation defined by such a formula, if they satisfy it. The other way is by regular path languages: regular languages consisting of directives on how to walk from one node to another through the graph, while checking MSO properties of nodes. Two nodes are in the relation defined by a path language if you can walk from the one node to the other following the directions of the language. For graphs, MSO will turn out to be stronger at defining relations than regular path languages. For trees, the strength will turn out to be the same.

In Section 3.6 we will go into the complexity of computing the relation defined by an MSO formula, when one of the free variables of the formula depends on the others.

3.1 Definitions

Let Σ and Γ be alphabets. A *graph-node relation* over (Σ, Γ) is a subset of $\{(G, u, v) \mid G \in \mathrm{GR}(\Sigma, \Gamma)$ and $u, v \in V_G\}$.

MSO-definable graph node relations

Let Σ and Γ be alphabets. According to Definition 1.44, an MSO formula $\phi(x, y) \in \mathrm{MSOL}_2(\Sigma, \Gamma)$ with two free node variables defines the graph-node relation

$$L_{\mathrm{GR}}(\phi(x, y)) = \{(G, v, w) \mid G \in \mathrm{GR}(\Sigma, \Gamma),\ v, w \in V_G, \text{ and } (G, v, w) \models \phi(x, y)\}.$$

Definition 3.1. *The set of all graph-node relations L', for which there are alphabets Σ, Γ and a formula $\phi(x, y) \in \mathrm{MSOL}_2(\Sigma, \Gamma)$ with $L_{\mathrm{GR}}(\phi(x, y)) = L'$ is the set of* MSO-definable graph-node relations*, denoted* MSO-NR.

Regular Graph-Node Relations

We can define graph-node relations with the help of path languages. They are akin to 'regular 2-paths' in [Oos89], 'routing languages' in [KS93], and 'regular tree embeddings' in [Eng89].

Definition 3.2. *Let* Σ *and* Γ *be alphabets. A* path language over (Σ, Γ) *is a string language over a set of* directives*, a finite subset of*

$$D_{\Sigma,\Gamma} = \bigcup_{\gamma \in \Gamma} \{\downarrow_\gamma, \uparrow_\gamma\} \cup \mathrm{MSOL}_1(\Sigma, \Gamma).$$

For a path language Π over Σ, Γ, $D(\Pi)$ is the smallest subset of $D_{\Sigma,\Gamma}$ such that $\Pi \subseteq D(\Pi)^*$. It is always finite.

A string from a path language defines a relation between nodes of a graph. The string gives a prescription of how to move from one node to another: $\downarrow_\gamma$ means "move along an edge labelled γ", $\uparrow_\gamma$ means "move against an edge labelled γ", and $\psi(x)$ means "check if ψ holds for the current node". A pair of nodes (v, w) is in the relation defined by the string if you can get from v to w following the prescription. It may seem a bit silly to use arrows up and down instead of right and left, but the reason for that will become obvious when we constrain ourselves to trees.

Formally, we can define this as follows. Let $G \in \mathrm{GR}(\Sigma, \Gamma)$. A string π over $D_{\Sigma,\Gamma}$ defines the relation $R_G(\pi) \subseteq V_G \times V_G$, which is defined by induction on the structure of π, as follows (where π_1 and π_2 are other strings over $D_{\Sigma,\Gamma}$):

$$\begin{aligned}
R_G(\varepsilon) &= \mathrm{id}_{V_G} \\
R_G(\downarrow_\gamma) &= \{(v, w) \mid v \xrightarrow{\gamma}_G w\} \\
R_G(\uparrow_\gamma) &= \{(v, w) \mid w \xrightarrow{\gamma}_G v\} \\
R_G(\psi(x)) &= \{(v, v) \mid (G, v) \models \psi(x)\} \\
R_G(\pi_1 \cdot \pi_2) &= R_G(\pi_1); R_G(\pi_2)
\end{aligned}$$

For a path language we define something similar. A pair of nodes is in the relation defined by the language if it is in a relation defined by one of its constituents. So, for a path language Π over (Σ, Γ), we define $R_G(\Pi) = \bigcup\{R_G(\pi) \mid \pi \in \Pi\}$. Now, a path language Π defines the graph-node relation

$$L_{\mathrm{GR}}(\Pi) = \{(G, v, w)) \mid G \in \mathrm{GR}(\Sigma, \Gamma) \text{ and } (v, w) \in R_G(\Pi)\}.$$

Definition 3.3. *A path language over* (Σ, Γ) *is* regular *if it is a regular string language over a finite subset of* $D_{\Sigma,\Gamma}$*. If* Π *is a regular path language, then* $L_{\mathrm{GR}}(\Pi)$ *is a* regular graph-node relation over (Σ, Γ)*. The set of all regular graph-node relations over any* (Σ, Γ) *is denoted* RP-NR.

Example 3.4. Let $\Sigma = \{\sigma, \rho\}$, and $\Gamma = \{\gamma, \delta\}$. In Figure 3.1 you find graph $G \in \mathrm{GR}(\Sigma, \Gamma)$. Consider regular expression $r_1 = {\downarrow_\gamma^*}$. It defines the relation

$$R_G(\|r_1\|) = \{(u_1, u_1), (u_2, u_2), (u_3, u_3), (u_4, u_4), (u_1, u_2), (u_1, u_4), (u_2, u_1), (u_2, u_4)\}.$$

Also, consider the expressions $r_2 = \mathrm{lab}_\rho(x)$ ($\mathrm{lab}_\rho \in \mathrm{MSOL}_1(\Sigma, \Gamma)$), and $r_3 = {\downarrow_\gamma^*} \cdot \mathrm{lab}_\rho(x)$, with the relations $R_G(\|r_2\|) = \{(u_4, u_4)\}$, and $R_G(\|r_3\|) = \{(u_1, u_4), (u_2, u_4), (u_4, u_4)\}$, respectively. □

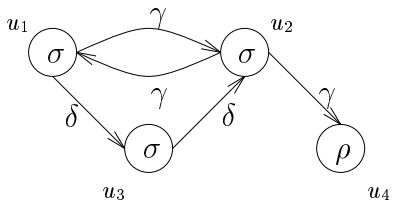

Figure 3.1: Graph G

We can view regular path languages as automata walking through a graph. We will use this view in Section 3.5 and Chapter 4. For an automaton A over a set of directives, and a graph G, an element (u, q) of $V_G \times Q$ is a *configuration* of the automaton. It signifies that A is at node u in state q. A *start configuration* is a configuration (u, q_0), and a *final configuration* is a configuration (u, q) with $q \in F$.

Definition 3.5. *Let Σ, Γ be alphabets of node and edge labels, respectively, Π a regular path language over (Σ, Γ), and $A = (Q, D(\Pi), \delta, q_0, F)$ a finite state (string) automaton with $\|A\| = \Pi$. Given a graph $G \in \mathrm{GR}(\Sigma, \Gamma)$, we define the following binary relations over $V_G \times Q$. For every $d \in D(\Pi)$, $u, u' \in V_G$ and $q, q' \in Q$,*

$$(u, q) \overset{d}{\twoheadrightarrow}_{A,G} (u', q') \quad \textit{iff} \quad (u, u') \in R_G(d) \textit{ and } (q, d, q') \in \delta.$$

If we leave out the label on the relation, we mean that configuration (u', q') can be reached from (u, q) through any single directive:

$$(u, q) \twoheadrightarrow_{A,G} (u', q') \quad \textit{iff} \quad \exists d \in D(\Pi) : (u, q) \overset{d}{\twoheadrightarrow}_{A,G} (u', q').$$

Last, for every $\pi = d_1 \cdots d_n \in D(\Pi)^*$, $u, u' \in V_G$ and $q, q' \in Q$,

$$(u, q) \overset{\pi}{\twoheadrightarrow}_{A,G} (u', q') \quad \textit{iff} \quad (u, q) \overset{d_1}{\twoheadrightarrow}_{A,G}; \cdots ; \overset{d_n}{\twoheadrightarrow}_{A,G} (u', q').$$

It follows from this definition that

$$(u, q) \overset{\pi}{\twoheadrightarrow}_{A,G} (u', q') \text{ iff } (u, u') \in R_G(\pi) \text{ and } (q, \pi, q') \in \delta,$$

and

$$(u,q) \twoheadrightarrow^*_{A,G} (u',q') \text{ iff } \exists \pi \in D(\Pi)^* : (u,q) \overset{\pi}{\twoheadrightarrow}_{A,G} (u',q').$$

A sequence $(u_0,q_0) \overset{d_1}{\twoheadrightarrow}_{A,G} (u_1,q_1) \overset{d_2}{\twoheadrightarrow}_{A,G} \cdots \overset{d_n}{\twoheadrightarrow}_{A,G} (u_n,q_n)$ with $q_n \in F$, is called a *walk* of A on G (*from* u_0 *to* u_n). Please note that a walk always starts in a start configuration and ends in a final configuration.

Let A be a deterministic automaton. In general, $\overset{d}{\twoheadrightarrow}_A$ is not a function for any d. If we confine ourselves to trees, however, $\overset{d}{\twoheadrightarrow}_A$ *is* a (partial) function, because no node has two outgoing or two incoming edges with the same label. In graphs and trees alike, $\twoheadrightarrow_A$ is in general not deterministic, because the symbols of the automaton are interpreted as actions, so it is not always clear what transition to take from a given state.

Lemma 3.6. *For every automaton* $A = (Q, D(\Pi), \delta, q_0, F)$*, and graph* G*:*

$$\begin{aligned}(u,u') \in R_G(\|A\|) \quad &\textit{iff} \quad \exists q_f \in F, \pi \in D(\Pi)^* : (u,q_0) \overset{\pi}{\twoheadrightarrow}_{A,G} (u',q_f),\\ &\textit{iff} \quad \exists q_f \in F : (u,q_0) \twoheadrightarrow^*_{A,G} (u',q_f).\end{aligned}$$

This follows immediately from the definitions.

Envision, if you wish, an automaton A_G over $D(\Pi)$, derived from G, that has the nodes of G as states, and a transition relation δ_G with $(u, \downarrow_\gamma, v) \in \delta_G$ if $u \overset{\gamma}{\to} v$, $(u, \uparrow_\gamma, v) \in \delta_G$ if $v \overset{\gamma}{\to} u$, and $(u, \psi(x), u) \in \delta_G$ if $(G,u) \models \psi(x)$. All states of A_G are both start and final states. Now, the ternary relation $\cdot \overset{\cdot}{\twoheadrightarrow}_A \cdot$ is the transition relation of the usual product automaton of A_G and A.

3.2 Regular Graph-Node Relations are MSO-definable

First we will compare the strength of regular path languages and MSO formulas on graphs. In this section, we show that for any regular path language there is an equivalent MSO formula. In the next section, we will show that the opposite doe not hold.

We claim that RP-NR $\subseteq$ MSO-NR. First we show how a regular path language can be converted in an equivalent MSO formula, and then we prove the correctness of the construction.

Construction

Let Π be a regular path language over (Σ, Γ) and r a regular expression for Π, i.e., $\|r\| = \Pi$. We construct a formula $\phi_r(x,y) \in \mathrm{MSOL}_2(\Sigma,\Gamma)$, such that $L_{\mathrm{GR}}(\phi_r(x,y)) = L_{\mathrm{GR}}(\|r\|)$. We do so using induction on the structure of r. Let r_1, r_2 be two regular expressions over $D_{\Sigma,\Gamma}$;

then

$$\begin{aligned}
\phi_{\varepsilon}(x,y) &= (x=y)\\
\phi_{\emptyset}(x,y) &= \text{false}\\
\phi_{\downarrow_\gamma}(x,y) &= \text{edg}_\gamma(x,y)\\
\phi_{\uparrow_\gamma}(x,y) &= \text{edg}_\gamma(y,x)\\
\phi_{\psi(z)}(x,y) &= \psi(x) \wedge (x=y)\\
\phi_{r_1 \cdot r_2}(x,y) &= \exists z(\phi_{r_1}(x,z) \wedge \phi_{r_2}(z,y))\\
\phi_{r_1+r_2}(x,y) &= \phi_{r_1}(x,y) \vee \phi_{r_2}(x,y)\\
\phi_{r_1^*}(x,y) &= \phi^*_{r_1}(x,y)
\end{aligned}$$

Proof of Correctness

We will prove that for any regular expression r, $L_{\text{GR}}(\|r\|) = L_{\text{GR}}(\phi_r(x,y))$. That is, for any regular expression r, graph $G \in \text{GR}(\Sigma,\Gamma)$ and $v,w \in V_G$, $(v,w) \in R_G(\|r\|)$ iff $(G,v,w) \models \phi_r(x,y)$.

First note that for all $\Pi_1, \Pi_2 \subseteq D^*_{\Sigma,\Gamma}$, $R_G(\Pi_1 \cdot \Pi_2) = R_G(\Pi_1); R_G(\Pi_2)$, this is proven as follows.

$$\begin{aligned}
R_G(\Pi_1 \cdot \Pi_2) &= R_G(\{\pi_1 \cdot \pi_2 \mid \pi_1 \in \Pi_1, \pi_2 \in \Pi_2\})\\
&= \bigcup\{R_G(\pi_1 \cdot \pi_2) \mid \pi_1 \in \Pi_1, \pi_2 \in \Pi_2\}\\
&= \bigcup\{R_G(\pi_1); R_G(\pi_2) \mid \pi_1 \in \Pi_1, \pi_2 \in \Pi_2\}\\
&= \bigcup\{R_G(\pi_1) \mid \pi_1 \in \Pi_1\}; \bigcup\{R_G(\pi_2) \mid \pi_2 \in \Pi_2\}\\
&= R_G(\Pi_1); R_G(\Pi_2)
\end{aligned}$$

This implies that $R_G(\Pi^i) = R_G(\Pi)^i$ and $R_G(\Pi^*) = R_G(\Pi)^*$. Recall also from Section 1.7, that $R_G(\phi) = \{(v,w) \mid (G,v,w) \models \phi(x,y)\}$.

Now, we give the proof based on the structure of r:

$r = \varepsilon$

$$\begin{aligned}
(v,w) \in R_G(\varepsilon) &\iff (v,w) \in \text{id}_{V_G}\\
&\iff v = w\\
&\iff (G,v,w) \models (x=y)\\
&\iff (G,v,w) \models \phi_\varepsilon(x,y)
\end{aligned}$$

$r = \emptyset$

$$\begin{aligned}
(v,w) \in R_G(\emptyset) &\iff (v,w) \in \emptyset\\
&\iff (G,v,w) \models \text{false}\\
&\iff (G,v,w) \models \phi_\emptyset(x,y)
\end{aligned}$$

$r = \psi(z)$

$$\begin{aligned}(v,w) \in R_G(\psi(z)) &\iff (G,v) \models \psi(x) \text{ and } v = w \\ &\iff (G,v) \models \psi(x) \text{ and } (G,v,w) \models (x = y) \\ &\iff (G,v,w) \models \psi(x) \wedge (x = y) \\ &\iff (G,v,w) \models \phi_{\psi(z)}(x,y)\end{aligned}$$

$r = \downarrow_\gamma$

$$\begin{aligned}(v,w) \in R_G(\downarrow_\gamma) &\iff v \xrightarrow{\gamma}_G w \\ &\iff (G,v,w) \models \mathrm{edg}_\gamma(x,y) \\ &\iff (G,v,w) \models \phi_{\downarrow_\gamma}(x,y)\end{aligned}$$

$r = \uparrow_\gamma$ This case is analogous to $r = \downarrow_\gamma$.

$r = r_1 \cdot r_2$ In the following formulas, we will also write the variable assignment as a function, to avoid ambiguity (see Section 1.7).

$$\begin{aligned}(v,w) \in R_G(\|r_1 \cdot r_2\|) = R_G(\|r_1\| \cdot \|r_2\|) = R_G(\|r_1\|); R_G(\|r_2\|) &\iff \\ \text{there is a node } u : (v,u) \in R_G(\|r_1\|) \text{ and } (u,w) \in R_G(\|r_2\|) &\iff \text{[induction]} \\ \text{there is a node } u : (G,v,u) \models \phi_{r_1}(x,y) \text{ and } (G,u,w) \models \phi_{r_2}(x,y) &\iff \\ \text{there is a node } u : (G,[x \mapsto v, y \mapsto w, z \mapsto u]) \models \phi_{r_1}(x,z) \wedge \phi_{r_2}(z,y) &\iff \\ (G,[x \mapsto v, y \mapsto w]) \models \exists z : \phi_{r_1}(x,z) \wedge \phi_{r_2}(z,y) &\iff \\ (G,v,w) \models \phi_{r_1 \cdot r_2}(x,y)\end{aligned}$$

$r = r_1 + r_2$

$$\begin{aligned}(v,w) \in R_G(\|r_1 + r_2\|) = R_G(\|r_1\| \cup \|r_2\|) = R_G(\|r_1\|) \cup R_G(\|r_2\|) &\iff \\ (v,w) \in R_G(\|r_1\|) \text{ or } (v,w) \in R_G(\|r_2\|) &\iff \text{[induction]} \\ (G,v,w) \models \phi_{r_1}(x,y) \text{ or } (G,v,w) \models \phi_{r_2}(x,y) &\iff \\ (G,v,w) \models \phi_{r_1}(x,y) \vee \phi_{r_2}(x,y) &\iff \\ (G,v,w) \models \phi_{r_1 + r_2}(x,y)\end{aligned}$$

$r = r_1^*$ Note that the induction hypothesis is equivalent to $R_G(\|r_1\|) = R_G(\phi_{r_1})$.

$$\begin{aligned}(v,w) \in R_G(\|r_1^*\|) = R_G(\|r_1\|^*) = R_G(\|r_1\|)^* &\iff \text{[induction]} \\ (v,w) \in R_G(\phi_{r_1})^* &\iff \text{[Lemma 1.47]} \\ (v,w) \in R_G(\phi_{r_1}^*) &\iff \text{[Definition of } \phi_{r^*}\text{]} \\ (v,w) \in R_G(\phi_{r_1^*}) &\iff \text{[Definition of } R_G(\phi)\text{]} \\ (G,v,w) \models \phi_{r_1^*}(x,y)\end{aligned}$$

Thus, we conclude that for any regular path language Π, there exists an MSO formula $\phi(x, y)$, such that $L_{\mathrm{GR}}(\Pi) = L_{\mathrm{GR}}(\phi(x, y))$, and this gives the following inclusion.

Proposition 3.7.

$$\text{RP-NR} \subseteq \text{MSO-NR}.$$

Example 3.8. Consider Example 3.4 with the graph in Figure 3.1. The MSO formulas corresponding to the languages of $r_1 = \downarrow_\gamma^*$, $r_2 = \mathrm{lab}_\rho(x)$ and $r_3 = \downarrow_\gamma^* \cdot \mathrm{lab}_\rho(x)$ are

$$\begin{aligned}
\phi_{r_1}(x, y) &= \mathrm{edg}_\gamma^*(x, y) \\
&= \forall X((x \in X \wedge \mathrm{closed}_{\mathrm{edg}_\gamma(x,y)}(X)) \rightarrow y \in X), \\
&= \mathrm{path}_\gamma(x, y) \\
\phi_{r_2}(x, y) &= \mathrm{lab}_\rho(x) \wedge x = y,
\end{aligned}$$

and

$$\begin{aligned}
\phi_{r_3}(x, y) &= \exists z(\phi_{r_1}(x, z) \wedge \phi_{r_2}(z, y)) \\
&= \exists z(\mathrm{path}_\gamma(x, z) \wedge \mathrm{lab}_\rho(z) \wedge z = y) \\
&= \mathrm{path}_\gamma(x, y) \wedge \mathrm{lab}_\rho(y),
\end{aligned}$$

respectively. □

3.3 MSO Definable Graph-Node Relations are not Always Regular

In this section we show that there are MSO definable graph-node relations for which no path language (regular or not) can be found. In MSO-NR we can define a relation that holds between nodes u and v exactly when they are in different connected components of the graph, i.e., there is *no* connection between them. Take, for example,

$$\phi(x, y) = \neg\, \mathrm{upath}(x, y).$$

In RP-NR such a relation is clearly not feasible, since there is no way to walk from u to v. Accordingly, MSO-NR $\not\subseteq$ RP-NR, and we can conclude the following proper inclusion.

Theorem 3.9.

$$\text{RP-NR} \subset \text{MSO-NR}.$$

Even for connected graphs there are graph-node relations that can be expressed by an MSO formula, but not by a path language.

For any alphabets Σ, Γ, let $\mathrm{CG}(\Sigma, \Gamma)$ be the set of all connected graphs over (Σ, Γ).

Proposition 3.10. *Let Σ, Γ be alphabets. There is an* MSO *formula $\phi(x,y) \in \mathrm{MSOL}(\Sigma,\Gamma)$ such that there is no path language Π over (Σ,Γ), with $L_{\mathrm{GR}}(\Pi) \upharpoonright \mathrm{CG}(\Sigma,\Gamma) = L_{\mathrm{GR}}(\phi) \upharpoonright \mathrm{CG}(\Sigma,\Gamma)$.*

Proof. We prove the proposition for the subclass of acyclic, connected graphs in which all incident edges of a node have different labels, since this is a relatively difficult subclass. One MSO formula that cannot be simulated by a path language is

$$\phi(x,y) = (x \neq y),$$

with semantics $L_{\mathrm{GR}}(\phi(x,y)) \upharpoonright \mathrm{CG}(\Sigma,\Gamma) = \{(G,u,v) \mid G \in \mathrm{CG}(\Sigma,\Gamma), u,v \in V_G, u \neq v\}$.

We will prove this is not possible, by introducing a set of graphs that have a circular structure, and are all very much alike, apart from the number of nodes. The path language will not be able to tell all of the graphs apart, since it can only use finitely many MSO formulas. Then, we find two indistinguishable graphs, such that any path that circles the first graph exactly halfway, fully circles the other graph. This leads to a contradiction, since any language that describes the relation $u \neq v$ has to have a path circling the first graph halfway, but cannot have a path that circles the second graph fully.

Consider graphs over $(\Sigma,\Gamma) = (\{\sigma\},\{a,b\})$. We prove the proposition by contradiction. Suppose there is a path language Π over (Σ,Γ), with

$$L_{\mathrm{GR}}(\Pi) \upharpoonright \mathrm{CG}(\Sigma,\Gamma) = \{(G,u,v) \mid G \in \mathrm{CG}(\Sigma,\Gamma), u,v \in V_G, u \neq v\}.$$

Only finitely many unary MSO formulas appear in Π, say $\psi_1(x),\ldots,\psi_m(x)$. Every node u of a graph G has a *type*, $\mathrm{type}_G(u) = (b_1,\ldots,b_m) \in \mathbb{B}^m$, with $b_i \Leftrightarrow ((G,u) \models \psi_i(x))$. There are 2^m different types of nodes.

We now define the following set of graphs over (Σ,Γ). For any even n,

$$G_n = (V_n, E_n, \mathrm{nlab}_n, \mathrm{elab}_n),$$

with

$$\begin{aligned} V_n &= \{u_i \mid 0 \leq i \leq n-1\},\\ E_n &= \{(u_i,u_j) \mid i \text{ even and } j - i \equiv \mp 1 \bmod n\},\\ \mathrm{nlab}_n &= \lambda u.\sigma\\ \mathrm{elab}_n &= \lambda(u_i,u_j).\begin{cases} a & \text{if } j-i \equiv 1 \bmod n,\\ b & \text{if } j-i \equiv -1 \bmod n. \end{cases} \end{aligned}$$

See Figure 3.2 for an example.

Clearly, for a fixed graph G_n, all $u_i \in G_n$ with even i have the same type, and all $u_i \in G_n$ with odd i have the same type. This is because of the automorphisms of G_n: $f_l(u_i) = u_{i+l \bmod n}$ is an automorphism for even l. The *type* of G_n is $\mathrm{type}(G_n) = (\mathrm{type}_{G_n}(u_0), \mathrm{type}_{G_n}(u_1))$. There are 2^{2m} different types of graphs.

For every $\pi \in \Pi$, let $\mathrm{nr}(\pi) = \#_{\downarrow a}(\pi) + \#_{\uparrow b}(\pi) - (\#_{\downarrow b}(\pi) + \#_{\uparrow a}(\pi))$. This is the net number of 'counter-clockwise' steps π takes in a graph of the above form.

The following hold.

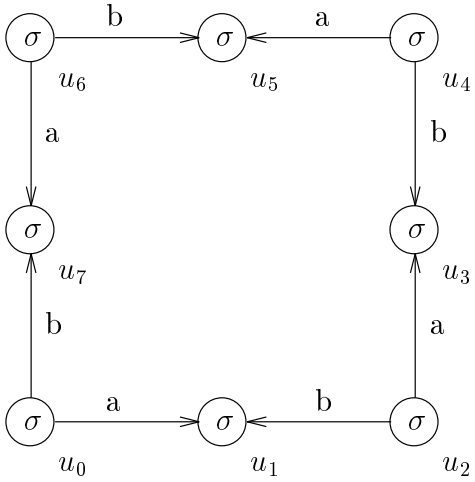

Figure 3.2: G_8

1. For any graph G_n, and $\pi \in \Pi$, if $(u_0, u_j) \in R_{G_n}(\pi)$, then $\mathrm{nr}(\pi) \bmod n = j$.

2. For any two graphs G_n and $G_{n'}$ of the same type, and any $\pi \in \Pi$, if $\mathrm{nr}(\pi) = i$ and $(u_0, u_{i \bmod n'}) \in R_{G_{n'}}(\pi)$, then also $(u_0, u_{i \bmod n}) \in R_{G_n}(\pi)$. This can be proven by induction on the length of π, and basically depends on the fact that π cannot discern 'in' which graph it is, judging by the information of the nodes, since they have the same type and the same incident edges in both graphs.

We now consider the sequence of graphs $G_2, G_4, G_8, \ldots$. Since this is an infinite sequence, there are graphs G_n, $G_{n'}$, with the same type, and $n' = n \cdot 2^k$ for some $k \geq 1$.

Since $u_0 \neq u_{n'/2}$ in $G_{n'}$, there is a $\pi \in \Pi$ such that $(u_0, u_{n'/2}) \in R_{G_{n'}}(\pi)$, by assumption. So, by (1), $\mathrm{nr}(\pi) \bmod n' = n'/2$. But now, since $(n'/2 + l \cdot n') \bmod n = 0$ for $l \in \mathbb{N}$, (2) implies $(u_0, u_0) \in R_{G_n}(\pi)$, a contradiction. □

Note that in the above proof, we do not demand that Π be a *regular* path language. There is not any path language over a finite alphabet that defines the above relation.

It remains an open problem whether (regular) path languages and MSO have the same strength on rooted directed acyclic graphs (i.e., trees with shared subexpressions).

3.4 MSO Definable and Regular Tree-Node Relations

In this section we confine ourselves to trees. First we will redefine the concepts that were defined for graphs in Section 3.1. Then we will prove that a node relation on trees defined by an MSO formula can be recognized by a regular path language.

3.4.1 Definitions

Let Σ be a operator alphabet. A *tree-node relation* over Σ is a subset of $\{(t, u, v) \mid t \in T_\Sigma$ and $u, v \in V_t\}$. According to Definition 1.44, an MSO formula $\phi(x, y) \in \mathrm{MSOL}_2(\Sigma)$ defines the tree-node relation

$$L(\phi(x, y)) = \{(t, v, w) \mid t \in T_\Sigma,\ v, w \in V_t, \text{ and } (t, v, w) \models \phi(x, y)\}.$$

A path language over $(\Sigma, \mathrm{rks}(\Sigma))$ is also called a path language over Σ. A path language Π over Σ defines a tree-node relation

$$L(\Pi) = \{(t, v, w)) \mid t \in T_\Sigma \text{ and } (v, w) \in R_t(\Pi)\}.$$

As with graphs, this relation is called regular if Π is a regular path language.

Definition 3.11. *The set of* MSO *definable tree-node relations is denoted* MSO-TNR. *The set of regular tree-node relations is denoted* RP-TNR.

We want to prove that MSO-TNR = RP-TNR. Clearly, RP-TNR $\subseteq$ MSO-TNR follows from Proposition 3.7. In the following subsection, we prove MSO-TNR $\subseteq$ RP-TNR.

3.4.2 MSO Definable Tree-Node Relations are Regular

Let Σ be an operator alphabet and let $\phi(x, y) \in \mathrm{MSOL}_2(\Sigma)$. We will prove that there exists a path language Π over Σ with $L(\Pi) = L(\phi(x, y))$. This path language has a special form: the strings in the language all describe paths in which no edge ever occurs more than once, that is, the shortest path is always taken. Compare this with [Oos89].

First we construct a tree automaton $\mathcal{A}$ over $\Sigma \cup (\Sigma \times B_2)$, that recognizes $\mathrm{mark}(t, u, v)$ exactly if $(t, u, v) \models \phi(x, y)$ (recall that $B_2 = \{(0, 1), (1, 0), (1, 1)\}$). The path language then simulates the tree automaton on the path from u to v, using MSO formulas (or rather attributes of an attribute grammar) to get information on the behavior of the automaton on the rest of the tree.

Construction of $\mathcal{A}$ is easy. Lemma 2.18 proves that from $\phi(x, y)$ we can construct a closed formula ψ over $\Sigma \cup (\Sigma \times B_2)$, such that $\mathrm{mark}(t, u, v) \models \psi$ iff $(t, u, v) \models \phi(x, y)$. Then, because the MSO definable and regular tree languages are the same (Proposition 1.45), there is a deterministic tree automaton $\mathcal{A} = (Q, F)$ over $\Sigma \cup (\Sigma \times B_2)$, with $\mathrm{val}_Q(\mathrm{mark}(t, u, v)) \in F$ iff $(t, u, v) \models \phi(x, y)$.

The Attribute Grammar

In order to gather information about parts of the tree outside the shortest path, we construct from $\mathcal{A}$ an attribute grammar G over Σ with finite semantic domains. The AG has synthesized attribute α, with semantic domain Q, and inherited attribute β, with semantic

domain $\mathcal{P}(Q)$. This AG is like the one in Chapter 2, but it does not have the δ attribute. The attribute grammar has the following rules for all $k \in \text{rks}(\Sigma)$ and $\sigma \in \Sigma_k$:

$$\begin{aligned}\langle \alpha, 0\rangle &= \sigma_Q(\langle \alpha, 1\rangle, \ldots, \langle \alpha, k\rangle),\\ \langle \beta, i\rangle &= \{q \in Q \mid \sigma_Q(\langle \alpha, 1\rangle, \ldots, \underset{i}{q}, \ldots, \langle \alpha, k\rangle) \in \langle \beta, 0\rangle\} \qquad \text{for } 1 \leq i \leq k,\end{aligned}$$

and it has one root rule,

$$\langle \beta, 0\rangle = F.$$

For any $t \in T_\Sigma$ and $v \in V_t$, the meaning of the attributes is

$$\langle \alpha, u\rangle = \text{val}_Q(\text{sub}_t(u)), \text{ and} \tag{3.1}$$

$$\langle \beta, u\rangle = \{q \in Q \mid \text{val}_Q(\text{ctx}_t(u))[\xi \mapsto q] \in F\}, \tag{3.2}$$

as was proven in Lemma 2.9 and Lemma 2.10.

The Regular Path Language

In a regular path language we are allowed to use MSO formulas with one free node variable. From Section 2.4 we recall that the value of a boolean attribute in an attribute grammar can always be calculated by an MSO formula with one free node variable. This means that, for convenience, we can check boolean attributes instead of MSO formulas. We do this by describing the boolean attribute between square brackets, using attributes α and β of G. Such a description is meant to denote the corresponding MSO formula. For example, when it says $[\sigma_Q(\langle \alpha, 1\rangle, \ldots, \underset{i}{q}, \ldots, \langle \alpha, k\rangle) = q']$ (where $q, q' \in Q$ and $i \in \text{rks}(\Sigma)$) in a regular path string, we mean "check the MSO formula corresponding to the (synthesized) boolean attribute $\text{att}_{q,q',i}$", which has the following rule, for all $k \in \text{rks}(\Sigma)$ and $\sigma \in \Sigma_k$:

$$\langle \text{att}_{q,q',i}, 0\rangle = \Big(\sigma_Q(\langle \alpha, 1\rangle, \ldots, \underset{i}{q}, \ldots, \langle \alpha, k\rangle) = q'\Big).$$

The right hand side of the rule is assumed to be 'false' if $i > k$.

For the other "assertions" in square brackets we can construct boolean attributes in a similar manner. The right-hand sides of the rules for the attributes consist exactly of the text between the square brackets. Again, a right hand side is assumed to be 'false' if one of its components is undefined.

The language Π that recognizes a pair of nodes (u, v) of a tree t iff $\text{mark}(t, u, v) \in L(\mathcal{A})$, consists of four groups of strings; $\Pi = \Pi_1 \cup \Pi_2 \cup \Pi_3 \cup \Pi_4$. See Figure 3.3 for the four possible cases.

1. For the case that $u = v$ we have one string consisting of one MSO formula:

$$\Pi_1 = \{[\langle \sigma, 1, 1\rangle_Q(\langle \alpha, 1\rangle, \ldots, \langle \alpha, k\rangle) \in \langle \beta, 0\rangle]\}.$$

   This formula checks if the automaton would recognize the tree if both marks were on the current node, assuming there are no marks on any other nodes in the tree.

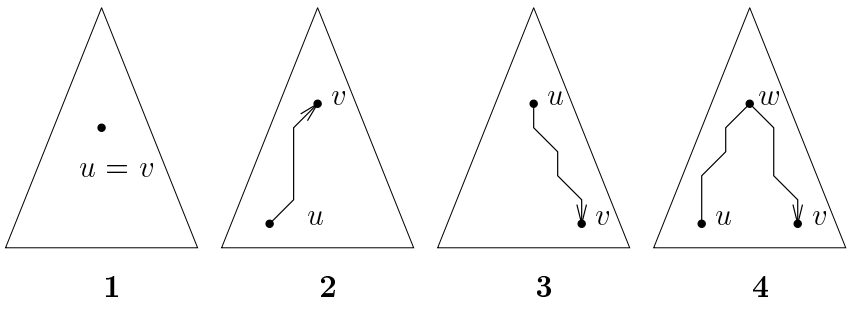

Figure 3.3: The four different groups

2. The following set of strings accounts for the cases that u is a proper descendant of v, or, more precisely, $u = v \cdot i_1 \cdot \cdots \cdot i_n$:

$$\begin{aligned}\Pi_2 = \{ \quad & [\langle\sigma,1,0\rangle_Q(\langle\alpha,1\rangle,\ldots,\langle\alpha,k\rangle) = q_n]\cdot \\ \uparrow_{i_n} \cdot \ & [\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{i_n}{q_n},\ldots,\langle\alpha,k\rangle) = q_{n-1}]\cdot \\ \vdots \\ \uparrow_{i_2} \cdot \ & [\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{i_2}{q_2},\ldots,\langle\alpha,k\rangle) = q_1]\cdot \\ \uparrow_{i_1} \cdot \ & [\langle\sigma,0,1\rangle_Q(\langle\alpha,1\rangle,\ldots,\underset{i_1}{q_1},\ldots,\langle\alpha,k\rangle) \in \langle\beta,0\rangle] \mid \\ n \in \mathbb{N}_+, i_1,\ldots,i_n \in \mathrm{rks}(\Sigma) \text{ and } & q_1,\ldots,q_n \in Q\}.\end{aligned}$$

A string in this group checks to see if the automaton accepts the tree with marks placed on u and v, and nowhere else. The states encode a potential run of the automaton, and the MSO formulas check if this run is correct.

3. This group of strings accounts for the case that u is a proper ancestor of v, more precisely, $v = u \cdot j_1 \cdot \cdots \cdot j_n$:

$$\begin{aligned}\Pi_3 = \{ \quad & [\langle\sigma,1,0\rangle_Q(\langle\alpha,1\rangle,\ldots,\underset{j_1}{p_1},\ldots,\langle\alpha,k\rangle) \in \langle\beta,0\rangle]\cdot \ \downarrow_{j_1} \cdot \\ & [\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{j_2}{p_2},\ldots,\langle\alpha,k\rangle) = p_1]\cdot \ \downarrow_{j_2} \cdot \\ & \qquad\qquad \vdots \\ & [\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{j_m}{p_m},\ldots,\langle\alpha,k\rangle) = p_{m-1}]\cdot \ \downarrow_{j_m} \cdot \\ & [\langle\sigma,0,1\rangle_Q(\langle\alpha,1\rangle,\ldots,\langle\alpha,k\rangle) = p_m] \mid \\ m \in \mathbb{N}_+, j_1,\ldots,j_m \in \mathrm{rks}(\Sigma), \text{ and } & p_1,\ldots,p_m \in Q\}.\end{aligned}$$

Each string in this group checks a potential run of the automaton, like in the last group. The order in which the states are checked is top-down, rather than bottom-up, but this is not relevant.

4. The last group of strings accounts for the case that neither u is an ancestor of v, nor v of u. More precisely, there is a w, such that $u = w \cdot i_1 \cdot \cdots \cdot i_n$ and $v = w \cdot j_1 \cdot \cdots \cdot j_m$,

and $i_1 \neq j_1$. Note that w is the least common ancestor of u and v.

$$\begin{aligned}
\Pi_4 = \{ \quad & [\langle\sigma,1,0\rangle_Q(\langle\alpha,1\rangle,\ldots,\langle\alpha,k\rangle) = q_n]\cdot \\
\uparrow_{i_n} \cdot \quad & [\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{i_n}{q_n},\ldots,\langle\alpha,k\rangle) = q_{n-1}]\cdot \\
& \vdots \\
\uparrow_{i_2} \cdot \quad & [\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{i_2}{q_2},\ldots,\langle\alpha,k\rangle) = q_1]\cdot \\
\uparrow_{i_1} \cdot \quad & [\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{i_1}{q_1},\ldots,\underset{j_1}{p_1},\ldots,\langle\alpha,k\rangle) \in \langle\beta,0\rangle]\cdot\downarrow_{j_1}\cdot \\
& [\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{j_2}{p_2},\ldots,\langle\alpha,k\rangle) = p_1]\cdot\downarrow_{j_2}\cdot \\
& \vdots \\
& [\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{j_m}{p_m},\ldots,\langle\alpha,k\rangle) = p_{m-1}]\cdot\downarrow_{j_m}\cdot \\
& [\langle\sigma,0,1\rangle_Q(\langle\alpha,1\rangle,\ldots,\langle\alpha,k\rangle) = p_m] \mid \\
m,n \in \mathbb{N}_+, i_1,\ldots,i_n,j_1,\ldots,j_m \in \mathrm{rks}(\Sigma), & i_1 \neq j_1, \text{ and } q_1,\ldots,q_n,p_1,\ldots,p_m \in Q\}.
\end{aligned}$$

Strings in this last group have a form that combines that of the two groups before. Again, the states are encoded in the formulas and the path language checks to see if the run is correct, walking from u to v.

We now give a right-linear grammar for Π to show that it is regular. The grammar has a nonterminal S, and nonterminals U_q and D_q for every $q \in Q$. The start symbol for the grammar is S, and the productions are the following.

stop immediately: ($u = v$)

$$S \to [\langle\sigma,1,1\rangle_Q(\langle\alpha,1\rangle,\ldots,\langle\alpha,k\rangle) \in \langle\beta,0\rangle]$$

start moving up:

$$S \to [\langle\sigma,1,0\rangle_Q(\langle\alpha,1\rangle,\ldots,\langle\alpha,k\rangle) = q]\cdot U_q \qquad \forall q \in Q$$

start moving down: (u is a proper ancestor of v)

$$S \to [\langle\sigma,1,0\rangle_Q(\langle\alpha,1\rangle,\ldots,\underset{j}{p},\ldots,\langle\alpha,k\rangle) \in \langle\beta,0\rangle]\cdot\downarrow_j \cdot D_p \qquad \forall p \in Q \text{ and } j \in \mathrm{rks}(\Sigma)$$

move another step up:

$$U_q \to \uparrow_i \cdot[\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{i}{q},\ldots,\langle\alpha,k\rangle) = q']\cdot U_{q'} \qquad \forall q,q' \in Q \text{ and } i \in \mathrm{rks}(\Sigma)$$

turn around: (u is not an ancestor of v or vice-versa)

$$U_q \to \uparrow_i \cdot[\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{i}{q},\ldots,\underset{j}{p},\ldots,\langle\alpha,k\rangle) \in \langle\beta,0\rangle]\cdot\downarrow_j \cdot D_p$$
$$\forall p,q \in Q, i,j \in \mathrm{rks}(\Sigma), \text{ with } i \neq j$$

move another step down:

$$D_p \to [\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{j}{p'},\ldots,\langle\alpha,k\rangle)=p]\cdot\downarrow_j\cdot D_{p'} \qquad \forall p,p'\in Q, j\in \mathrm{rks}(\Sigma)$$

stop moving up: (v is a proper ancestor of u)

$$U_q \to \uparrow_i \cdot[\langle\sigma,0,1\rangle_Q(\alpha_1,\ldots,\underset{i}{q},\ldots,\alpha_k)\in\langle\beta,0\rangle] \qquad \forall q\in Q$$

stop moving down:

$$D_p \to [\langle\sigma,0,1\rangle_Q(\langle\alpha,1\rangle,\ldots,\langle\alpha,k\rangle)=p] \qquad \forall p\in Q$$

Proof of Correctness

Before we proceed with the proof, we present two lemmata. The first is a generalization of Lemma 2.9.

Lemma 3.12. *For any tree t over Σ and nodes u,v,w of t, if u and v are not descendants of w, then* $\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(w))=\langle\alpha,w\rangle$.

Proof. The assumption that u and v are not descendants of w is equivalent to the statement $\mathrm{sub}_{\mathrm{mark}(t,u,v)}(w)=\mathrm{sub}_t(w)$. So, since $\langle\alpha,w\rangle=\mathrm{val}_Q(\mathrm{sub}_t(w))$ by Lemma 2.9, $\langle\alpha,w\rangle=\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(w))$. □

The second lemma is a generalization of Lemma 2.10 and Lemma 2.11.

Lemma 3.13. *For any tree t over Σ and nodes u,v,w of t, if u and v are descendants of w, then* $\mathrm{val}_Q(\mathrm{mark}(t,u,v))\in F \Leftrightarrow \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(w))\in\langle\beta,w\rangle$.

Proof. The assumption that u and v are descendants of w, is equivalent to $\mathrm{ctx}_{\mathrm{mark}(t,u,v)}(w)=\mathrm{ctx}_t(w)$. Therefore,

$$\begin{array}{lcl}
\mathrm{val}_Q(\mathrm{mark}(t,u,v))\in F & \Longleftrightarrow & \\
\mathrm{val}_Q\Big(\mathrm{ctx}_{\mathrm{mark}(t,u,v)}(w)[\xi\mapsto \mathrm{sub}_{\mathrm{mark}(t,u,v)}(w)]\Big)\in F & \Longleftrightarrow & \text{[Proposition 1.18]}\\
\mathrm{val}_Q(\mathrm{ctx}_{\mathrm{mark}(t,u,v)}(w))[\xi\mapsto \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(w))]\in F & \Longleftrightarrow & \\
\mathrm{val}_Q(\mathrm{ctx}_t(w))[\xi\mapsto \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(w))]\in F & \Longleftrightarrow & \text{[Lemma 2.10]}\\
\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(w))\in\langle\beta,w\rangle & &
\end{array}$$

□

To prove that Π does what we want it to do, we have to prove that for any tree $t\in T_\Sigma$ and $u,v\in V_t$, $(t,u,v)\in L(\Pi)$ iff $(t,u,v)\in L(\phi(x,y))$. To that extent it suffices to prove $(t,u,v)\in L(\Pi)$ iff $\mathrm{mark}(t,u,v)\in L(\mathcal{A})$, or equivalently,

$$(u,v)\in R_t(\Pi) \quad \text{iff} \quad \mathrm{val}_Q(\mathrm{mark}(t,u,v))\in F.$$

We distinguish four cases, analogous to the cases used before: first: $u = v$, second: u is a proper descendant of v, third: u is a proper ancestor of v, and fourth: a third node is the least common ancestor of u and v. What the path language does is simulate the tree automaton. For those parts of the tree on which it can assume no marks are placed, it can use MSO formulas (or attributes) to calculate what value the tree automaton gives. For the path between u and v, it keeps track of the state of the automaton. We check here that it does that correctly.

1. Let $u = v$ and $\pi = [\langle \sigma, 1, 1\rangle_Q(\langle \alpha, 1\rangle, \ldots, \langle \alpha, k\rangle) \in \langle \beta, 0\rangle]$. Then,

$$\mathrm{val}_Q(\mathrm{mark}(t, u, v)) \in F$$

$\Longleftrightarrow$ [Lemma 3.13]

$$\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u)) \in \langle \beta, u\rangle$$

$\Longleftrightarrow$ [$u = v$]

$$\langle \mathrm{nlab}_t(u), 1, 1\rangle_Q\Big(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u \cdot 1) \cdots \mathrm{sub}_{\mathrm{mark}(t,u,v)}(u \cdot \mathrm{rk}(u))\Big) \in \langle \beta, u\rangle$$

$\Longleftrightarrow$ [Lemma 3.12]

$$\langle \mathrm{nlab}_t(u), 1, 1\rangle_Q(\langle \alpha, u \cdot 1\rangle, \ldots, \langle \alpha, u \cdot \mathrm{rk}(u)\rangle) \in \langle \beta, u\rangle$$

$\Longleftrightarrow$ [by definition of π]

$$(u, u) \in R_t(\pi)$$

$\Longleftrightarrow$

$$(u, u) \in R_t(\Pi)$$

We will clarify the last step. First, $(u, u) \in R_t(\pi)$ implies $(u, u) \in R_t(\Pi)$, because $\pi \in \Pi$. Second, if $(u, u) \in R_t(\Pi)$, then $(u, u) \in R_t(\pi)$, because we can not use a string $\pi' \in \Pi \setminus \Pi_1$. This can easily be seen if we consider the other possibilities. Suppose $\pi' \in \Pi_2$, then $(u, v) \in R_t(\pi')$ implies that u is a proper descendant of v. This is not the case. It is not possible that $\pi' \in \Pi_3$, for a similar reason. Last, $\pi' \notin \Pi_4$, because $(u, v) \in R_t(\Pi_4)$ implies that the least common ancestor of u and v is not equal to u or v.

2. u is a proper descendant of v. Suppose $u = v \cdot i_1 \cdot \cdots \cdot i_n$. Let $u_0 = v$, $u_l = u_{l-1} \cdot i_l$ and $q_l = \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u_l))$ for $1 \leq l \leq n$. Now let

$$\begin{aligned}
\pi = \quad & [\langle \sigma, 1, 0\rangle_Q(\langle \alpha, 1\rangle, \ldots, \langle \alpha, k\rangle) = q_n]\cdot \\
\uparrow_{i_n} \cdot \quad & [\sigma_Q(\langle \alpha, 1\rangle, \ldots, \underset{i_n}{q_n}, \ldots, \langle \alpha, k\rangle) = q_{n-1}]\cdot \\
& \vdots \\
\uparrow_{i_2} \cdot \quad & [\sigma_Q(\langle \alpha, 1\rangle, \ldots, \underset{i_2}{q_2}, \ldots, \langle \alpha, k\rangle) = q_1]\cdot \\
\uparrow_{i_1} \cdot \quad & [\langle \sigma, 0, 1\rangle_Q(\langle \alpha, 1\rangle, \ldots, \underset{i_1}{q_1}, \ldots, \langle \alpha, k\rangle) \in \langle \beta, 0\rangle].
\end{aligned}$$

• First, assume that $\mathrm{val}_Q(\mathrm{mark}(t,u,v)) \in F$. We prove all the assertions in π, for $u_n, \ldots, u_1, u_0$ respectively, because that proves $(u,v) \in R_t(\pi)$ and thus $(u,v) \in R_t(\Pi)$, since π is an element of Π. The first assertion is clearly true for $u_n = u$, since, by Lemma 3.12,

$$\langle \mathrm{nlab}(u), 1, 0\rangle_Q(\langle \alpha, u \cdot 1\rangle, \ldots, \langle \alpha, u \cdot \mathrm{rk}(u)\rangle) = \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u)) = q_n.$$

The second assertion up to the second last assertion are also true, since for all $l \geq 2$,

$$\mathrm{nlab}(u_{l-1})_Q(\langle \alpha, u_{l-1} \cdot 1\rangle, \ldots, \underset{i_l}{q_l}, \ldots, \langle \alpha, u_{l-1} \cdot \mathrm{rk}(u_{l-1})\rangle) =$$

$$\mathrm{nlab}(u_{l-1})_Q(\langle \alpha, u_{l-1} \cdot 1\rangle, \ldots, \underset{i_l}{\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u_l))}, \ldots, \langle \alpha, u_{l-1} \cdot \mathrm{rk}(u_{l-1})\rangle) =$$

$$\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u_{l-1})) = q_{l-1}.$$

For the last assertion, $\mathrm{val}_Q(\mathrm{mark}(t,u,v)) \in F$, combined with Lemma 3.13, implies $\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(v)) \in \langle \beta, v\rangle$, and thus

$$\langle \mathrm{nlab}(v), 0, 1\rangle_Q(\langle \alpha, v \cdot 1\rangle, \ldots, \underset{i_1}{q_1}, \ldots, \langle \alpha, v \cdot \mathrm{rk}(v)\rangle) \in \langle \beta, v\rangle.$$

• Second, suppose $(u,v) \in R_t(\Pi)$, so we can walk from u to v following the instructions in Π. We will prove that the tree automaton succeeds on $\mathrm{mark}(t,u,v)$.

If $(u,v) \in R_t(\Pi)$, then there is a $\pi' \in \Pi$, such that $(u,v) \in R_t(\pi')$. Clearly, $\pi' \in \Pi_2$, with $i'_1 = i_1, \ldots, i'_n = i_n$, but possibly different $q'_1, \ldots, q'_n$. What are the values of $q'_1, \ldots, q'_n$?

Since $\langle \mathrm{nlab}(u), 1, 0\rangle_Q(\langle \alpha, u \cdot 1\rangle, \ldots, \langle \alpha, u \cdot \mathrm{rk}(u)\rangle) = \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u))$, the first assertion of π' implies $q'_n = \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u_n)) = q_n$. Furthermore, if $q'_l = q_l$, then it follows from the assertions in π' that

$$\begin{aligned} q'_{l-1} &= \mathrm{nlab}(u_{l-1})_Q(\langle \alpha, u_{l-1} \cdot 1\rangle, \ldots, \underset{i_l}{q'_l}, \ldots, \langle \alpha, u_{l-1} \cdot \mathrm{rk}(u_{l-1})\rangle) \\ &= \mathrm{nlab}(u_{l-1})_Q(\langle \alpha, u_{l-1} \cdot 1\rangle, \ldots, \underset{i_l}{q_l}, \ldots, \langle \alpha, u_{l-1} \cdot \mathrm{rk}(u_{l-1})\rangle) \\ &= \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u_{l-1})) \\ &= q_{l-1}. \end{aligned}$$

So, by induction, $q'_l = \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u_l)) = q_l$ for all $l \in [1,n]$. Hence, $\pi' = \pi$.

Because the last assertion in π is true, it follows that

$$\langle \mathrm{nlab}(v), 0, 1\rangle_Q(\langle \alpha, v \cdot 1\rangle, \ldots, \underset{i_1}{\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u_1))}, \ldots, \langle \alpha, v \cdot \mathrm{rk}(v)\rangle) \in \langle \beta, v\rangle,$$

in other words,

$$\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(v)) \in \langle \beta, v\rangle.$$

Because of Lemma 3.13, this implies that

$$\mathrm{val}_Q(\mathrm{mark}(t,u,v)) \in F.$$

3. u is a proper ancestor of v. Suppose $v = u \cdot j_1 \cdot \cdots \cdot j_m$. Let $v_0 = u$, $v_l = v_{l-1} \cdot j_l$, and $p_l = \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(v_l))$, for $1 \leq l \leq m$. Now, let

$$\begin{aligned}\pi = &[\langle\sigma,1,0\rangle_Q(\langle\alpha,1\rangle,\ldots,\underset{j_1}{p_1},\ldots,\langle\alpha,k\rangle) \in \langle\beta,0\rangle]\cdot\downarrow_{j_1}\cdot\\ &[\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{j_2}{p_2},\ldots,\langle\alpha,k\rangle) = p_1]\cdot\downarrow_{j_2}\cdot\\ &\vdots\\ &[\sigma_Q(\langle\alpha,1\rangle,\ldots,\underset{j_m}{p_m},\ldots,\langle\alpha,k\rangle) = p_{m-1}]\cdot\downarrow_{j_m}\cdot\\ &[\langle\sigma,0,1\rangle_Q(\langle\alpha,1\rangle,\ldots,\langle\alpha,k\rangle) = p_m],\end{aligned}$$

• First, suppose that $\mathrm{val}_Q(\mathrm{mark}(t,u,v)) \in F$. We will prove $(u,v) \in R_t(\pi)$. Since $\mathrm{val}_Q(\mathrm{mark}(t,u,v)) \in F$, it follows that $\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u)) \in \langle\beta,u\rangle$, so the first assertion of π is true. For the second to the second last assertion, note that for all $1 \leq l \leq m-1$

$$\begin{gathered}\mathrm{nlab}(v_l)_Q(\langle\alpha,v_l\cdot 1\rangle,\ldots,\underset{j_{l+1}}{p_{l+1}},\ldots\langle\alpha,v_l\cdot\mathrm{rk}(v_l)\rangle) =\\ \mathrm{nlab}(v_l)_Q(\langle\alpha,v_l\cdot 1\rangle,\ldots,\underset{j_{l+1}}{\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(v_l\cdot j_{l+1}))},\ldots,\langle\alpha,v_l\cdot\mathrm{rk}(v_l)\rangle) =\\ \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(v_l)) = p_l,\end{gathered}$$

so these assertions are true as well. The last assertion is true, since

$$\langle\mathrm{nlab}(v),0,1\rangle_Q(\langle\alpha,v\cdot 1\rangle,\ldots,\langle\alpha,\mathrm{rk}(v)\rangle) = \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(v)).$$

So, clearly, $(u,v) \in R_t(\pi)$, and since $\pi \in \Pi$ it follows that $(u,v) \in R_t(\Pi)$.

• Second, we assume that $(u,v) \in R_t(\Pi)$ and we prove $\mathrm{val}_Q(\mathrm{mark}(t,u,v)) \in F$. The proof is very similar to that of the previous case; first we observe that there is a $\pi' \in \Pi$ with $(u,v) \in R_t(\pi')$. Now clearly $\pi' \in \Pi_3$, with $j'_1 = j_1,\ldots,j'_m = j_m$, but possibly different $p'_1,\ldots,p'_m$. From the last assertion it follows that $p'_1 = p_1$, and by the other assertions, it follows with induction that $p'_l = p_l$ for all $1 \leq l \leq m$. Therefore, $\pi' = \pi$. Now we can conclude from the first assertion that

$$\langle\mathrm{nlab}(u),1,0\rangle_Q\Big(\langle\alpha,u\cdot 1\rangle,\ldots,\underset{j_1}{\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(v_1))},\ldots,\langle\alpha,u\cdot\mathrm{rk}(u)\rangle\Big) \in \langle\beta,u\rangle,$$

and thus that $\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u)) \in \langle\beta,u\rangle$, and finally, by Lemma 3.13, that $\mathrm{val}_Q(\mathrm{mark}(t,u,v)) \in F$.

4. For the fourth and last case, assume there is a node w, that is the least common proper ancestor of u and v. More precisely, let $u = w \cdot i_1 \cdot \cdots \cdot i_n$, and $v = w \cdot j_1 \cdot \cdots \cdot j_m$, with $i_1 \neq j_1$. Let $u_0 = w$, $u_l = u_{l-1} \cdot i_l$ and $q_l = \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(u_l))$ for $1 \leq l \leq n$. Let $v_0 = w$, $v_l = v_{l-1} \cdot j_l$ and $p_l = \mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(v_l))$ for $1 \leq l \leq m$. Now, let

$$
\begin{array}{lll}
\pi = & [\langle \sigma, 1, 0\rangle_Q(\langle \alpha, 1\rangle, \ldots, \langle \alpha, k\rangle) = q_n]\cdot & \text{(i)} \\
\uparrow_{i_n} \cdot & [\sigma_Q(\langle \alpha, 1\rangle, \ldots, \underset{i_n}{q_n}, \ldots, \langle \alpha, k\rangle) = q_{n-1}]\cdot & \text{(ii)} \\
 & \vdots & \text{(ii)} \\
\uparrow_{i_2} \cdot & [\sigma_Q(\langle \alpha, 1\rangle, \ldots, \underset{i_2}{q_2}, \ldots, \langle \alpha, k\rangle) = q_1]\cdot & \text{(ii)} \\
\uparrow_{i_1} \cdot & [\sigma_Q(\langle \alpha, 1\rangle, \ldots, \underset{i_1}{q_1}, \ldots, \underset{j_1}{p_1}, \ldots, \langle \alpha, k\rangle) \in \langle \beta, 0\rangle]\cdot \downarrow_{j_1} \cdot & \text{(iii)} \\
 & [\sigma_Q(\langle \alpha, 1\rangle, \ldots, \underset{j_2}{p_2}, \ldots, \langle \alpha, k\rangle) = p_1]\cdot \downarrow_{j_2} \cdot & \text{(iv)} \\
 & \vdots & \text{(iv)} \\
 & [\sigma_Q(\langle \alpha, 1\rangle, \ldots, \underset{j_m}{p_m}, \ldots, \langle \alpha, k\rangle) = p_{m-1}]\cdot \downarrow_{j_m} \cdot & \text{(iv)} \\
 & [\langle \sigma, 0, 1\rangle_Q(\langle \alpha, 1\rangle, \ldots, \langle \alpha, k\rangle) = p_m] & \text{(v)}
\end{array}
$$

• First, we assume $\mathrm{val}_Q(\mathrm{mark}(t,u,v)) \in F$, and we prove that $(u,v) \in R_t(\pi)$. The assertions marked (i) or (ii) are true; the proof is analogous to that of case 2. The assertions marked (iv) or (v) are also true; the proof is analogous to that of the case 3. Only the assertion marked (iii) remains to be proven. Well, $\mathrm{val}_Q(\mathrm{mark}(t,u,v)) \in F$ implies $\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(w)) \in \langle \beta, w\rangle$, and since

$$\mathrm{sub}_{\mathrm{mark}(t,u,v)}(w) = \mathrm{nlab}(w)_Q(\langle \alpha, w \cdot 1\rangle, \ldots, \underset{i_1}{q_1}, \ldots, \underset{j_1}{p_1}, \ldots, \langle \alpha, w \cdot \mathrm{rk}(w)\rangle)$$

this assertion is also true. Now it follows that $(u,v) \in R_t(\pi)$, and, because $\pi \in \Pi$, $(u,v) \in R_t(\Pi)$.

• Second, we assume $(u,v) \in R_t(\Pi)$ and we prove $\mathrm{val}_Q(\mathrm{mark}(t,u,v)) \in F$. If $(u,v) \in R_t(\Pi)$, then there is a $\pi' \in \Pi$ such that $(u,v) \in R_t(\pi')$. Now $\pi' \in \Pi_4$, and from the reasoning in cases 2 and 3, it is clear that the q_i and p_j are as given, for all i and j. This leads us to conclude that

$$\mathrm{nlab}(w)_Q(\langle \alpha, w \cdot 1\rangle, \ldots, \underset{i_1}{q_1}, \ldots, \underset{j_1}{p_1}, \ldots, \langle \alpha, w \cdot \mathrm{rk}(w)\rangle) \in \langle \beta, w\rangle,$$

and so $\mathrm{val}_Q(\mathrm{sub}_{\mathrm{mark}(t,u,v)}(w)) \in \langle \beta, w\rangle$ and $\mathrm{val}_Q(\mathrm{mark}(t,u,v)) \in F$.

Lemma 3.14. *Let Σ be an operator alphabet. For every $\phi(x,y) \in \mathrm{MSOL}_2(\Sigma)$ there is a regular path language Π over Σ, such that for every tree $t \in T_\Sigma$ and nodes $u, v \in V_t$, $(u,v) \in R_t(\Pi)$ iff $(t,u,v) \models \phi(x,y)$. Moreover, for all $t \in T_\Sigma$, and $u, v \in V_t$ with $(t,u,v) \models \phi(x,y)$, there is a unique string $\pi \in \Pi$ such that $(u,v) \in R_t(\pi)$.*

If A is a deterministic finite state string automaton recognizing Π, and $(u,v) \in R_t(\|A\|)$, then there is a unique walk $(u, q_0) \overset{d_1}{\twoheadrightarrow}_{A,t} \ldots \overset{d_n}{\twoheadrightarrow}_{A,t} (v, q_f)$ with $q_f \in F$.

Proof. The first part of the lemma is proven above. The second part then easily follows from Lemma 3.6, the fact that $\pi = d_1 \cdots d_n$ is unique, and the fact that $\overset{d}{\twoheadrightarrow}_{A,t}$ is a (partial) function for any d. □

It follows that MSO-TNR is a subset of RP-TNR.

Lemma 3.15.

$$\text{MSO-TNR} \subseteq \text{RP-TNR}$$

3.4.3 MSO and Regular Tree-Node Relations are the Same

From Proposition 3.7 and the last lemma, we obtain the following equality.

Theorem 3.16.

$$\text{MSO-TNR} = \text{RP-TNR}\,.$$

This has the practical consequence that for any MSO formula $\phi(x, y)$ with two free node variables, there is a path language Π, such that for any tree t and nodes u and v of t,

$$(t, u, v) \models \phi(x, y) \Leftrightarrow (u, v) \in R_t(\Pi),$$

and vice-versa.

Remark 3.17. From here on, we can, and will, assume that all regular path languages are of the (shortest-path) form presented in Subsection 3.4.2.

3.5 Comparison with Routing Languages

The path languages defined above are syntactically richer than the routing languages of [KS93]. They are also semantically richer: there are tree-node relations that can be defined by a path language, but not by a routing language.

Definition 3.18. *A* routing language *over* (Σ, Γ) *is a string language over the following constrained set of directives.*

$$D^r_{\Sigma,\Gamma} = \bigcup_{\gamma \in \Gamma} \{\downarrow_\gamma, \uparrow_\gamma\} \cup \{\text{root}(x)\} \cup \{\text{leaf}(x)\} \cup \{\text{lab}_\sigma(x) \mid \sigma \in \Sigma\},$$

where $\text{root}(x)$ *and* $\text{leaf}(x)$ *are the MSO formulas with one free node variable defined in Section 1.7.*

Routing languages are restricted path languages, so without further ado, we can speak about the relation and tree-node language defined by a routing language.

Routing languages are used by Klarlund and Schwartzbach to define recursive data structures. Recursive data strutures have an intrinsic tree structure (that can be defined by, e.g., a tree automaton). Path languages are used to define extra pointers in the tree structure, for instance to produce the data structure of circulary linked lists or root-linked binary trees. If t is an instance of a data structure (i.e., a tree), a tuple (u, v) in the relation defined by the routing language signifies a pointer from node u to node v. Klarlund and Schwartzbach show that many useful data structures can be defined using functional routing languages (i.e., routing languages Π for which $R_t(\Pi)$ is functional for every t). They also prove that the pointers defined by such routing languages can be computed efficiently. We show here that there are structures that can be defined by path languages, but not by routing languages. In Section 3.6 we will show that path languages can be evaluated just as efficiently as routing languages.

Proposition 3.19. *There is a (functional) regular path language Π, such that there is no regular routing language Π' with $L(\Pi') = L(\Pi)$.*

Proof. We consider binary trees with red and black leaves, i.e., leaves with label 'red' and leaves with label 'black', respectively. Let $\Sigma = \Sigma_0 \cup \Sigma_2$, with $\Sigma_0 = \{\text{red}, \text{black}\}$, and $\Sigma_2 = \{\text{internal}\}$. We define a path language Π that connects the leafs of a tree. If there is exactly one red leaf, all leaves have a pointer to that leaf. If there is no red leaf, or if there is more than one, all leaves are linked in left-to-right circular order.

To be able to construct the corresponding path language, we define two abbreviations: ALL stands for the regular path expression $\downarrow_1 + \downarrow_2 + \uparrow_1 + \uparrow_2$, and STEP abbreviates $\uparrow_2^* \cdot(\uparrow_1 \cdot \downarrow_2 + \text{root}(x))\cdot \downarrow_1^*$. Expression ALL allows to move in any direction, while STEP describes the language that moves from one leaf to the next in left-to-right circular order. We also define the MSO formula $\text{orl} = \exists!x : \text{lab}_{\text{red}}(x)$, that is true iff there is exactly one red leaf.

Now consider the path language

$$\Pi = \|((\text{leaf}(x) \wedge \text{orl}) \cdot \text{ALL}^* \cdot \text{lab}_{\text{red}}(x)) + ((\text{leaf}(x) \wedge \neg\text{orl}) \cdot \text{STEP} \cdot \text{leaf}(x))\|.$$

If a tree t has exactly one red leaf v, then $(u, v) \in R_t(\Pi)$ iff u is a leaf. Otherwise, $(u, v) \in R_t(\Pi)$ iff u and v are both leaves, and v follows u in left-to-right circular order. Note that $R_t(\Pi)$ is a partial function for every $t \in T_\Sigma$.

This tree-node relation cannot be defined by a regular routing language. We prove so by contradiction.

Suppose there is a regular routing language $\Pi' \subseteq (D^{\text{r}}_{\Sigma,\text{rks}(\Sigma)})^*$, with $L(\Pi') = L(\Pi)$. Let $A' = (Q, D(\Pi'), \delta, q_0, F)$ be a finite state (string) automaton with $\|A'\| = \Pi'$. Now consider the trees t and t' in Figure 3.4. Name the leaves in both trees u_1 through $u_{\#Q+1}$ in left-to-right order. In t, all leaves are black. In t', leaves u_1 through $u_{\#Q}$ are black and leaf $u_{\#Q+1}$ is red. In the following, let $\text{succ}(k) = (k + 1) \bmod (\#Q + 1)$, giving the successor of a node number in left-to-right circular order.

Since t has no red leaves, for every $k \in [1, \#Q + 1]$, there is an $f_k \in F$, such that $(u_k, q_0) \twoheadrightarrow^*_{A,t} (u_{\text{succ}(k)}, f_k)$ (see Lemma 3.6). On the other hand, since t' has exactly one red

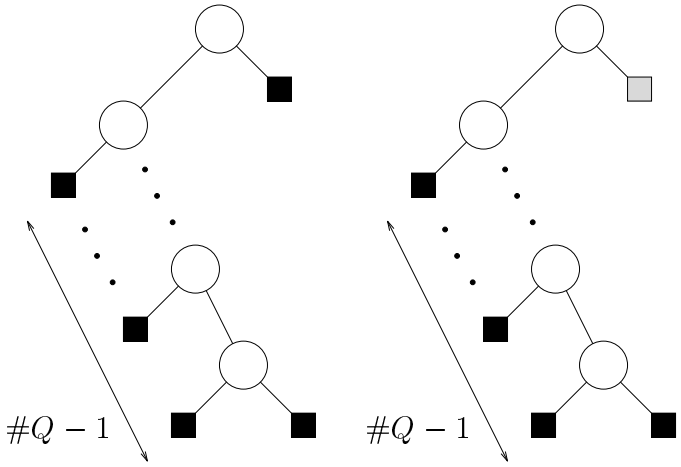

Figure 3.4: To the left: tree t, with only black leaves; to the right: tree t', with one red leaf

leaf, for all $k \neq \#Q$, there is no $f \in F : (u_k, q_0) \twoheadrightarrow^*_{A,t'} (u_{\text{succ}(k)}, f)$. This implies that for all k there is a $q_k \in Q$ such that $(u_k, q_0) \twoheadrightarrow^*_{A,t} (u_{\#Q+1}, q_k) \twoheadrightarrow^*_{A,t} (u_{\text{succ}(k)}, f_k)$. This is easily proven: any walk by A in t that does not go through $u_{\#Q+1}$ can also be done in t' (and the walk starting in $u_{\#Q}$ goes through $u_{\#Q+1}$ anyhow).

Concentrating solely on tree t now, this means that for all $k \in [1, \#Q + 1]$, there is a $q_k \in Q$, with

$$(u_k, q_0) \twoheadrightarrow^*_{A,t} (u_{\#Q+1}, q_k) \twoheadrightarrow^*_{A,t} (u_{\text{succ}(k)}, f_k).$$

But since there are $\#Q + 1$ possibilities for k, and only $\#Q$ states, there have to be a k and $k' \neq k \in [1, \#Q + 1]$ such that $q_k = q_{k'}$, and

$$(u_k, q_0) \twoheadrightarrow^*_{A,t} (u_{\#Q+1}, q_k) \twoheadrightarrow^*_{A,t} (u_{\text{succ}(k)}, f_k)$$

and

$$(u_{k'}, q_0) \twoheadrightarrow^*_{A,t} (u_{\#Q+1}, q_{k'}) \twoheadrightarrow^*_{A,t} (u_{\text{succ}(k')}, f_{k'}).$$

But since $q_k = q_{k'}$, this implies

$$(u_{k'}, q_0) \twoheadrightarrow^*_{A,t} (u_{\#Q+1}, q_k) \twoheadrightarrow^*_{A,t} (u_{\text{succ}(k)}, f_k)$$

and thus $(u_{k'}, u_{\text{succ}(k)}) \in R_t(\Pi')$ with $k \neq k'$, a contradiction. □

3.6 Complexity

Klarlund and Schwartzbach proved that for any regular routing language Π and tree t, $R_t(\Pi)$ can be computed in linear time, if it is a partial function. Although path languages

are stronger than the routing languages, we can evaluate them in the same order of time. This gives us a linear time method of computing $R_t(\phi(x,y))$ for given $\phi(x,y)$, if that relation is a partial function. Compare this to Theorem 2.20, where we stated that $R_t(\phi(x,y))$ can be computed in $O(n^2)$ time in any case.

Lemma 3.20. *Let Σ be an operator alphabet, $\phi(x,y) \in \mathrm{MSOL}_2(\Sigma)$, and $t \in T_\Sigma$. If $R_t(\phi(x,y)) = \{(u,v) \mid (t,u,v) \models \phi(x,y)\}$ is a partial function, then $R_t(\phi(x,y))$ can be computed in time linear to the size of the tree.*

Proof. The main part of this proof is adapted from [KS93].

Let t be a tree over Σ. To be able to compute $R_t(\phi(x,y))$ in linear time, we first transform $\phi(x,y)$ into an equivalent regular path language Π, as was shown in Section 3.4. We can compute and store $\{u \mid (t,u) \models \psi(x)\}$ in linear time, for all $\psi(x)$ that occur in Π (see Theorem 2.20). For Π we construct a finite state (string) automaton $A = (Q, \Sigma, \delta, q_0, F)$, with $\|A\| = \Pi$.

In the next algorithm, we build a table T. The table is indexed with the configurations of the automaton A, walking on t (see Lemma 3.5). For every configuration $(u,q) \in V_t \times Q$, $T[u,q] = u' \in V_t$, where u' is a node with $(u,q) \twoheadrightarrow^*_{A,t} (u', q_f)$, for some $q_f \in F$, if such a node exists. If it does not, $T[u,q] = \textsc{nil}$. Then, by the functionality of $R_t(\phi)$, for any node u, $T[u,q_0] = u'$ iff $(u,u') \in R_t(\Pi) = R_t(\phi(x,y))$, and hence

$$R_t(\phi) = \{(u,u') \mid T[u,q_0] = u', u' \neq \textsc{nil}\}.$$

type
  Entry $= V_t \times Q$;
var
  T : **array** [Entry] **of** V_t;
  (u,q), (u',q') : Entry;
  L : **queue of** Entry;
begin
  Init(L);
  for all $(u,q) \in V_t \times Q$ **do**
    $T[u,q] :=$ NIL
  od ;
  for all $(u,q) \in V_t \times F$ **do**
    $T[u,q] := u$;
    $L \Leftarrow (u,q)$
  od ;
  while $\neg$ IsEmpty(L) **do**
    $(u',q') \Leftarrow L$;
    for all $(u,q) \in V_t \times Q$ with $T[u,q] =$ NIL and $(u,q) \twoheadrightarrow_{A,t} (u',q')$ **do**
      $T[u,q] := T[u',q']$;
      $L \Leftarrow (u,q)$;

od
od
end

In this algorithm, Init initializes the queue, $L \Leftarrow (u, q)$ puts (u, q) on the end of queue L, $(u, q) \Leftarrow L$ removes the first element from the queue and assigns it to (u, q), and IsEmpty(L) returns true if L is empty.

This algorithm runs in time linear to the size of the tree, as was proved in [KS93]. The crux in this proof is the following. Consider the last for loop. Because the path language is fixed, in the automaton A, any state q' has a fixed number of incoming arrows. For every arrow, the source q and label d are fixed, and, since the automaton walks on a tree, there is at most one u such that $(u, q) \xrightarrow{d}_{A,t} (u', q')$. This means that in the last for loop, only a constant number of entries (u, q) have to be considered, given (u', q'). It should be noted that every such u can be computed in constant time. For the directives $\uparrow_i$ and $\downarrow_i$ this is clear, and for the directives ψ it is true because $\{u \mid (t, u) \models \psi(x)\}$ has been precomputed for all ψ.

The correctness of the algorithm should be clear: in the second for loop, the table is filled in correctly (with respect to the intended contents of the table) for all final configurations. From there on, the automaton is followed back on its walk, and every possible previous configuration is filled in correctly. Since every configuration that has a walk leading to a final configuration is eventually reached, the algorithm is correct. □

More than Two Free Variables

Theorem 2.20 states we can find $R_t(\phi)$ in $O(n^k)$ time if ϕ has k free node variables. We can speed up calculation of $R_t(\phi)$ for formulas with more than one free node variable, if one of the variables depends on (some of) the others.

First we will formally define dependencies. We speak of a dependency in a relation when the value of one of the elements of a tuple in the relation is fully determined by the value of some of the others.

Definition 3.21. *Let R be a k-ary relation, $i \in [1, k]$, and $D \subseteq [1, k]$. We say that i* (functionally) depends on D (in R) *if for all $((a_1, \ldots, a_k), (a'_1, \ldots, a'_k)) \in R$ with $a_d = a'_d$ for all $d \in D$, a_i is equal to a'_i.*

Note that for any relation R, i depends on $\{i\}$, and if i depends on D, then for any $D' \supseteq D$, i depends on D'.

Recall that $R_t(\phi(x_1, \ldots, x_k)) = \{(u_1, \ldots, u_k) \mid (t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k)\}$ for a tree $t \in T_\Sigma$.

Theorem 3.22. *Let Σ be an operator alphabet, $k \geq 2$, $\phi(x_1, \ldots, x_k) \in \mathrm{MSOL}_k(\Sigma)$ and $t \in T_\Sigma$. If there is an $i \in [1, k]$, and a $D \subseteq [1, k]$ with $i \notin D$, such that i depends on D in $R_t(\phi(x_1, \ldots, x_k))$, then $R_t(\phi(x_1, \ldots, x_k))$ can be computed in $O(n^{k-1})$ time.*

Proof. The case of $k = 2$ is proven in the last lemma.

For the other cases, without loss of generality, we can assume $i = k$ and $D = [1, k-1]$. What we will do is the following. For every possibility for the first $k-2$ arguments, we build a tree with $k-2$ marks at the appropriate places, and we have a formula checking the last two arguments, in which the last argument depends on the previous one.

We proceed like we did in Section 2.5, using Lemma 2.18. We transform $\phi(x_1, \ldots, x_k)$ into an MSO formula $\psi(x_{k-1}, x_k)$ over $\Sigma \cup (\Sigma \times B_{k-2})$ with two free node variables, such that

$$(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k) \iff (\text{mark}(t, u_1, \ldots, u_{k-2}), u_{k-1}, u_k) \models \psi(x_{k-1}, x_k).$$

For all tuples $(u_1, \ldots, u_{k-1}, u_k)$ and $(u_1, \ldots, u_{k-1}, u'_k)$ in the relation $R_t(\phi(x_1, \ldots, x_k))$, by the dependency, u_k is equal to u'_k. Hence, if

$$(\text{mark}(t, u_1, \ldots, u_{k-2}), u_{k-1}, u_k) \models \psi(x_{k-1}, x_k),$$

and

$$(\text{mark}(t, u_1, \ldots, u_{k-2}), u_{k-1}, u'_k) \models \psi(x_{k-1}, x_k),$$

then $u_k = u'_k$. So, $R_{\text{mark}(t,u_1,\ldots,u_{k-2})}(\psi(x_{k-1}, x_k))$ is a partial function (its second argument depends on its first).

For any sequence of nodes $u_1, \ldots, u_{k-2} \in V_t$, we can find the corresponding marked tree $\text{mark}(t, u_1, \ldots, u_{k-2})$ in $O(n)$ time. Because u_k depends on u_{k-1}, we can then find all pairs (u_{k-1}, u_k) such that $(\text{mark}(t, u_1, \ldots, u_{k-2}), u_{k-1}, u_k) \models \psi(x_{k-1}, x_k)$ in $O(n)$ time, as shown in Lemma 3.20. There are $O(n^{k-2})$ possible combinations for $u_1, \ldots, u_{k-2}$, so the time needed to find all $u_1, \ldots, u_k$ such that $(\text{mark}(t, u_1, \ldots, u_{k-2}), u_{k-1}, u_k) \models \psi(x_{k-1}, u_k)$ is $O(n^{k-1})$. □

Dependency is Decidable

For the above theorem to be of any use, we have to be able to find out whether a dependency exists. Given a formula $\phi(x, y) \in \text{MSOL}_2(\Sigma)$, it is decidable whether $R_t(\phi)$ is a partial function for all $t \in T_\Sigma$. Consider the closed formula

$$\text{func}_\phi = \forall x(\exists! y : \phi(x, y) \vee \neg \exists y : \phi(x, y)).$$

Now $R_t(\phi)$ is a partial function for all $t \in T_\Sigma$ iff func_ϕ is a tautology. This is decidable by Proposition 1.46.

In general, given a formula $\phi(x_1, \ldots, x_k) \in \text{MSOL}_k(\Sigma)$, for any i and D it is decidable whether for all t, i functionally depends on D in $R_t(\phi)$. Here we have the formula

$$\text{func}_\phi = \forall x_1, \ldots, x_k, \forall x'_1, \ldots, x'_k \Big((\phi(x_1, \ldots, x_k) \wedge \phi(x'_1, \ldots, x'_k) \wedge \forall d \in D(x_d = x'_d)) \rightarrow x_i = x'_i \Big),$$

and i depends on D in $R_t(\phi)$ for all $t \in T_\Sigma$ iff func_ϕ is a tautology.

Chapter 4

Tree Transducers

In this chapter we consider functions from trees over one alphabet to trees over another: tree transductions. More precisely, we look at tree transducers, i.e., specific methods of defining tree transductions. We consider both tree MSO tree transducers and attributed tree transducers, a form of attribute grammars. We show how attributed tree transducers can be modified in such a way that they define the same class of transductions as MSO tree transducers do.

In the last section we will go into the complexity of computing an MSO definable tree transduction.

4.1 MSO Transducers

MSO graph transducers define their output graph in terms of MSO formulas on the input graph. See for example [Eng91, Cou92, Cou94], for a somewhat more general approach. In the next two definitions we define syntax and semantics of MSO graph transducers.

Definition 4.1. *An* MSO graph transducer *from* $\mathrm{GR}(\Sigma_1, \Gamma_1)$ *to* $\mathrm{GR}(\Sigma_2, \Gamma_2)$ *is a quadruple*

$$T = (C, \phi, \Psi, X),$$

where

1. *C is a finite set, the* copy set,
2. *ϕ is a closed* MSO *formula over (Σ_1, Γ_1), the* domain formula,
3. *$\Psi = \{\psi_{\sigma_2,c}(x)\}_{\sigma_2 \in \Sigma_2, c \in C}$, where the $\psi_{\sigma_1,c}(x) \in \mathrm{MSOL}_2(\Sigma_1, \Gamma_1)$ are the* node formulas, *and*
4. *$X = \{\chi_{\gamma_2,c,c'}(x,y)\}_{\gamma_2 \in \Gamma_2, c,c' \in C}$, where the $\chi_{\gamma_2,c,c'}(x,y) \in \mathrm{MSOL}_2(\Sigma_1, \Gamma_1)$ are the* edge formulas.

The copy number *of T is $\#C$.*

Definition 4.2. *An* MSO *graph transducer* $T = (C, \phi, \Psi, X)$ *from* $\mathrm{GR}(\Sigma_1, \Gamma_1)$ *to* $\mathrm{GR}(\Sigma_2, \Gamma_2)$ *defines a partial function* $\mathcal{T}_{gr} : \mathrm{GR}(\Sigma_1, \Gamma_1) \rightharpoonup \mathrm{GR}(\Sigma_2, \Gamma_2)$*, with* $\mathcal{T}_{gr}(G_1) = G_2$ *iff*

1. $G_1 \models \phi$,
2. $V_{G_2} = \{(v, c) \mid v \in V_{G_1}, c \in C \text{ and } \exists! \sigma_2 \in \Sigma_2 : (G_1, v) \models \psi_{\sigma_2, c}(x)\}$,
3. $E_{G_2} = \{((v, c), (v', c')) \mid (v, c), (v', c') \in V_{G_2}, \text{ and } \exists! \gamma_2 \in \Gamma_2 : (G_1, v, v') \models \chi_{\gamma_2, c, c'}(x, y)\}$,
4. $\mathrm{nlab}_{G_2} = \{((v, c), \sigma_2) \mid (v, c) \in V_{G_2} \text{ and } (G_1, v) \models \psi_{\sigma_2, c}(x)\}$, *and*
5. $\mathrm{elab}_{G_2} = \left\{\left(((v, c), (v', c')), \gamma_2\right) \mid ((v, c), (v', c')) \in E_{G_2} \text{ and } (G_1, v, v') \models \chi_{\gamma_2, c, c'}(x, y)\right\}$.

Note that $\mathrm{dom}(\mathcal{T}_{\mathrm{gr}}) = \{G_1 \in \mathrm{GR}(\Sigma_1, \Gamma_1) \mid G_1 \models \phi\}$.

Remark 4.3. For the sake of convenience, we can assume without loss of generality that for an MSO graph transducer T from $\mathrm{GR}(\Sigma_1, \Gamma_1)$ to $\mathrm{GR}(\Sigma_2, \Gamma_2)$, and for all $G_1 \in \mathrm{GR}(\Sigma_1, \Gamma_1)$, the following hold.

1. For fixed c, the $\psi_{\sigma_2, c}(x)$ are mutually exclusive, i.e., for any $v \in V_{G_1}$, there is not more than one $\sigma_2 \in \Sigma_2$ such that $(G_1, v) \models \psi_{\sigma_2, c}(x)$ (which implies that we can replace $\exists! \sigma_2$ by $\exists \sigma_2$ in case 2 of the last definition).
2. For fixed c, c', the $\chi_{\gamma_2, c, c'}(x, y)$ are mutually exclusive, i.e., for any $v, v' \in V_{G_1}$, there is not more than one $\gamma_2 \in \Gamma_2$, such that $(G_1, v, v') \models \chi_{\gamma_2, c, c'}(x, y)$ (which implies that we can replace $\exists! \gamma_2$ by $\exists \gamma_2$ in case 3 of the last definition).
3. An edge formula $\chi_{\gamma_2, c, c'}(x, y)$ only holds for nodes $v, v' \in V_{G_1}$ if $(u, c), (v, c')$ are in the output graph. In other words, if $\mathcal{T}_{\mathrm{gr}}(G_1) = G_2$, then $\forall \gamma_2 \in \Gamma_2, c, c' \in C, v, v' \in V_{G_1} : ((G_1, v, v') \models \chi_{\gamma_2, c, c'}(x, y)$ implies $(v, c), (v', c') \in V_{G_2})$.
4. The edge and node formulas are only true if the domain formula is satisfied. That means that for all σ_2, c, v: if $(G_1, v) \models \psi_{\sigma_2, c}(x)$, then $G_1 \models \phi$, and likewise, for all γ_2, c, c', v, v': if $(G_1, v, v') \models \chi_{\gamma_2, c, c'}(x, y)$, then $G_1 \models \phi$.

These conditions are all easily achieved. For example, the third condition is achieved by changing every formula $\chi_{\gamma_2, c, c'}(x, y)$ into $\chi'_{\gamma_2, c, c'}(x, y) = \exists! \sigma_2 : \psi_{\sigma_2, c}(x) \wedge \chi_{j, c, c'}(x, y) \wedge \exists! \sigma_2 : \psi_{\sigma_2, c}(y)$. The other conditions are achieved in a likewise simple manner. The given requirements allow us to check fewer conditions in the remainder of the chapter.

Definition 4.4. *Let* Σ *and* Δ *be operator alphabets. An* MSO tree transducer *from* Σ *to* Δ *is an* MSO *graph transducer* T *from* $(\Sigma, \mathrm{rks}(\Sigma))$ *to* $(\Delta, \mathrm{rks}(\Delta))$ *for which* $\mathcal{T}_{gr}(t) \in T_\Delta$ *for all* $t \in T_\Sigma \cap \mathrm{dom}(\mathcal{T}_{gr})$*. It defines the partial function* $\mathcal{T} : T_\Sigma \rightharpoonup T_\Delta$*, where* $\mathcal{T} = \mathcal{T}_{gr} \restriction T_\Sigma$*. The set of all* MSO *definable tree transductions is denoted* MSOTT.

Example 4.5. We give an MSO tree transducer from Σ to Δ, where $\Sigma = \Sigma_0 \cup \Sigma_2$, with $\Sigma_0 = \{\#\}$, and $\Sigma_2 = \{\sigma\}$, and $\Delta = \Delta_0 \cup \Delta_1 \cup \Delta_2$, with $\Delta_0 = \Sigma_0$, $\Delta_2 = \Sigma_2$, and $\Delta_1 = \{*\}$. The transducer transforms a tree by inserting a $*$ on each of its edges. See Figure 4.1 for an example. The transducer is

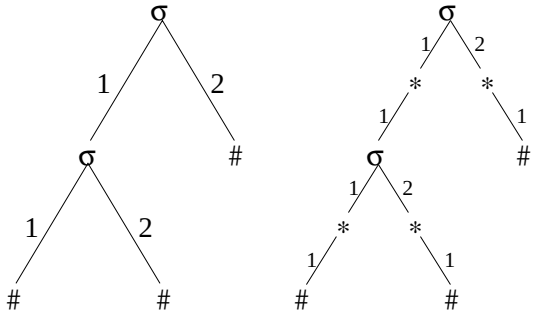

Figure 4.1: Example of a tree and its transduction

$$T = (C, \phi, \Psi, X),$$

with

- The copy set C is $\{0, 1\}$. A node $\langle v, 0\rangle$ of $\mathcal{T}(t)$ is a direct copy of a node v of t (with the same label), and a node $\langle v, 1\rangle$ has label $*$, and has node $\langle v, 0\rangle$ as its child. Two copies are made of every node except the root.
- The domain formula $\phi = \text{true}$, there are no input restrictions.
- The node formulas are as follows.

$$\begin{array}{lll} \psi_{\delta,0}(x) & = \text{lab}_\delta(x) & \text{for all } \delta \in \Sigma, \\ \psi_{*,0}(x) & = \text{false}, & \\ \psi_{\delta,1}(x) & = \text{false}, & \text{for all } \delta \in \Sigma, \text{ and} \\ \psi_{*,1}(x) & = \neg\, \text{root}(x). & \end{array}$$

- And the edge formulas are

$$\begin{array}{lll} \chi_{j,0,0}(x,y) & = \text{false} & \text{for all } j \in \{1,2\}, \\ \chi_{j,1,1}(x,y) & = \text{false} & \text{for all } j \in \{1,2\}, \\ \chi_{j,0,1}(x,y) & = \text{edg}_j(x,y) & \text{for all } j \in \{1,2\}, \\ \chi_{1,1,0}(x,y) & = x = y, & \text{and} \\ \chi_{2,1,0}(x,y) & = \text{false}\,. & \end{array}$$

□

We give one more example, adapted from [FV95].

Example 4.6. Let T be an MSO tree transducer from Σ to Δ. The operator alphabet Σ is $\{\sigma, *, \#\}$, with $\mathrm{rk}_\Sigma(\sigma) = 2$, and $\mathrm{rk}_\Sigma(\#) = \mathrm{rk}_\Sigma(*) = 0$. The operator alphabet Δ is $\{1, 2, *, \#\}$, with $\mathrm{rk}_\Delta(1) = \mathrm{rk}_\Delta(2) = 1$, and $\mathrm{rk}_\Delta(*) = \mathrm{rk}_\Delta(\#) = 0$. If the input tree t contains exactly one leaf labelled $*$, the transducer transforms it into a tree over Δ, which codes the path leading from the root of t to the leaf labelled $*$ in the obvious way. Otherwise, the output is $\#$. For example, $\mathcal{T}(\sigma(\sigma(\#, \sigma(\#, *)), \#)) = 1(2(2(*)))$, and $\mathcal{T}(\sigma(*, \sigma(\#, *))) = \#$.

$$T = (\{c\}, \phi, \{\psi_{1,c}, \psi_{2,c}, \psi_{*,c}, \psi_{\#,c}\}, \{\chi_{1,c,c}\}),$$

where

$$\begin{aligned}
\phi &= \mathrm{true} \\
\psi_{1,c}(x) &= \exists! y : \mathrm{lab}_*(y) \wedge \exists y\Big(\mathrm{edg}_1(x, y) \wedge \exists z(\mathrm{path}(y, z) \wedge \mathrm{lab}_*(z))\Big), \\
\psi_{2,c}(x) &= \exists! y : \mathrm{lab}_*(y) \wedge \exists y\Big(\mathrm{edg}_2(x, y) \wedge \exists z(\mathrm{path}(y, z) \wedge \mathrm{lab}_*(z))\Big), \\
\psi_{*,c}(x) &= \exists! y : \mathrm{lab}_*(y) \wedge \mathrm{lab}_*(x), \\
\psi_{\#,c}(x) &= \neg\exists! y : \mathrm{lab}_*(y) \wedge \mathrm{root}(x), \\
\chi_{1,c,c}(x, y) &= \mathrm{edg}_1(x, y) \vee \mathrm{edg}_2(x, y).
\end{aligned}$$

For the last formula, please note that an edge in the output tree is only drawn if both nodes it is incident to exist, so this need not be checked explicitly. However, this rule violates the third condition of Remark 4.3. Without altering the transduction, we can comply with the third condition by changing the edge formula into

$$\chi_{1,c,c}(x, y) = \exists \delta \in \Delta, c \in C : \psi_{\delta,c}(x) \wedge (\mathrm{edg}_1(x, y) \vee \mathrm{edg}_2(x, y)) \wedge \exists \delta \in \Delta, c \in C : \psi_{\delta,c}(y).$$

□

4.2 Attributed Tree Transducers

An attributed tree transducer defines a function from trees over one alphabet to trees over another, like MSO tree transducers do. They are based on attribute grammars, with some extra constraints, see [Fül81].

We will here develop a kind of attributed tree transducers that is equivalent to MSO tree transducers. The first type we consider is that of plain attributed tree transducers, based on unconditional attribute grammars with trees over Δ as the only semantic domain. The semantic rules are limited to involve substitution only.

Definition 4.7. *Let Σ, Δ be operator alphabets. A* plain attributed tree transducer (att, for short) *from Σ to Δ is an unconditional, (weakly) non-circular attribute grammar $(\Sigma, \Omega, (S, I, W), R, \alpha_{\mathrm{mean}})$ with the following additional properties.*

- $\Omega = \{T_\Delta\}$.
- *Every semantic rule is of the form*

$$\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle),$$

where for all $t_1, \ldots, t_k \in T_\Delta$,

$$f(t_1, \ldots, t_k) = r'[\xi_1 \mapsto t_1, \ldots, \xi_k \mapsto t_k],$$

for some linear, non-deleting $r' \in T_\Delta(\{\xi_1, \ldots, \xi_k\})$.

The set of all transductions that can be defined by a plain attributed tree transducer is denoted ATT.

From here on, in a semantic rule, we will abbreviate $f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle)$ by r, where $r = r'[\xi_1 \mapsto \langle \alpha_1, i_1 \rangle, \ldots, \xi_k \mapsto \langle \alpha_k, i_k \rangle]$. Note that, since the $\langle \alpha_j, i_j \rangle$ are mutually distinct (see Definition 1.27), r is a linear term in $T_\Delta(A \times \mathrm{rks}(\Sigma))$.

We require non-deleting terms, because that implies that weak non-circularity coincides with the classical concept of circularity (and hence is decidable). Linearity of the terms used in the rules is a convenient technical detail. It can always be achieved by duplicating attributes, although this will not preserve the WSUR (this will become important later on).

We give an example of a rule that is not linear, and we then transform it to two linear rules. If the output alphabet is Δ, with $\delta \in \Delta_2$, a typical rule would be

$$\langle \alpha, 0 \rangle = f(\langle \alpha, 1 \rangle), \text{ with } f(t) = \delta(\xi, \xi)[\xi \mapsto t_1].$$

This rule would be abbreviated $\langle \alpha, 0 \rangle = \delta(\langle \alpha, 1 \rangle, \langle \alpha, 1 \rangle)$ (note that the abbreviation of a rule is a linear term over $T_\Delta(A \times \mathrm{rks}(\Sigma))$ iff the term used in the rule is linear). We can make it linear by adding an attribute α', with rule

$$\langle \alpha', 0 \rangle = f'(\langle \alpha, 1 \rangle), \text{ where } f'(t) = \xi[\xi \mapsto t], \text{ for all } t,$$

and changing the rule for α to

$$\langle \alpha, 0 \rangle = f''(\langle \alpha', 0 \rangle, \langle \alpha, 1 \rangle), \text{ with } f''(t_1, t_2) = \delta(\xi_1, \xi_2)[\xi_1 \mapsto t_1, \xi_2 \mapsto t_2].$$

Note that the WSUR is not preserved ($\langle \alpha, 1 \rangle$ is used twice).

Remark 4.8. By Definition 1.34, an att G induces a (total) function $\mathcal{G}$ from T_Σ to T_Δ: $\mathcal{G}(t) = \mathrm{dec}_{G,t}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t) \rangle)$.

We are looking for a sort of attributed tree transducer that is equivalent to MSO tree transducers. Plain attributed tree transducers will not do, not even for transductions that are total functions, although this is not a trivial observation. Example 4.6 on page 67

is from [FV95][1], where it is proven that it is not a transduction computable by a plain attributed tree transducer.

To overcome this problem, we generalize plain attributed tree transducers to attributed tree transducers with flags. Flags are attributes with a finite semantic domain. An att with flags is a 2-phase attribute grammar: one can always evaluate the flags first, and the tree attributes second. The operations on tree attributes are also limited: only substitution can be used, and flags can be tested.

Definition 4.9. *Let Σ and Δ be operator alphabets. An* attributed tree transducer with flags (fatt) *from Σ to Δ is an attribute grammar over Σ with the following constraints.*

- *Each set in Ω is either finite or equal to T_Δ. Attributes with semantic domain T_Δ are called* tree attributes. *The set of all these tree attributes is denoted B (boom, Baum = tree). An attribute with finite semantic domain is called a* flag*; the set of all flags is denoted F.*

- *The attribute α_{mean} is a tree attribute.*

- *Every semantic rule has one of the following forms:*

  - *it is either a* flag rule,

    $$\langle \mathrm{fl}_0, i_0 \rangle = f(\langle \mathrm{fl}_1, i_1 \rangle, \ldots, \langle \mathrm{fl}_k, i_k \rangle),$$

    with $\mathrm{fl}_i \in F$, and $f : W(\mathrm{fl}_1) \times \cdots \times W(\mathrm{fl}_k) \to W(\mathrm{fl}_0)$,

  - *or a* tree rule,

    $$\langle \beta, i \rangle = \begin{cases} f_1(\langle \beta_{11}, i_{11} \rangle, \ldots, \langle \beta_{1k_1}, i_{1k_1} \rangle) & \text{if } \langle \mathrm{fl}, 0 \rangle = e_1 \\ \vdots & \vdots \\ f_N(\langle \beta_{N1}, i_{N1} \rangle, \ldots, \langle \beta_{Nk_N}, i_{Nk_N} \rangle) & \text{if } \langle \mathrm{fl}, 0 \rangle = e_N \end{cases}$$

    where $\beta, \beta_{jj'} \in B$, $\mathrm{fl} \in F$, and $W(\mathrm{fl}) = \{e_1, \ldots, e_N\}$, and as in Definition 4.7, for all j and all $t_1, \ldots, t_{k_j}$, $f_j(t_1, \ldots, t_{k_j}) = r'_j[\xi_1 \mapsto t_1, \ldots, \xi_{k_j} \mapsto t_{k_j}]$, for some linear, non-deleting $r'_j \in T_\Delta(\{\xi_1, \ldots, \xi_{k_j}\})$. We will abbreviate the functions in the way we did in Definition 4.7.

- *The semantic conditions pertain only to the flags, i.e., they are of the form*

  $$f(\langle \mathrm{fl}_1, i_1 \rangle, \ldots, \langle \mathrm{fl}_k, i_k \rangle),$$

  with $\mathrm{fl}_1, \ldots, \mathrm{fl}_k$ flags and f a function from the appropriate semantic domains to $\mathbb{B}$.

[1] The example is not exactly the one given in [FV95], but if there is an att that calculates this transduction, it can easily be rewritten in their notation. Removing all rules with righthand side # then gives an att that would calculate their transduction.

- *The attribute grammar is weakly non-circular with respect to* (F, B) *(see Definition 1.40).*

The set of all transductions definable by an attributed tree transducer with flags is denoted FATT.

The att rules of the form $\langle \beta, i \rangle = r$, that were introduced in Definition 4.7 for plain atts, are easily mimicked by fatts. Intuitively, this is done by adding a flag fl, with a singleton semantic domain, say {true}. Now, the rule can be written

$$\langle \beta, i \rangle = \Big\{ r \quad \text{if } \langle \text{fl}, 0 \rangle = \text{true}$$

The simple technical details are omitted. Thus, we will allow plain att rules in atts with flags.

On one hand, plain atts are weaker then MSO tree transducers, as was just discussed. On the other hand, plain atts and atts with flags are stronger than MSO tree transducers, since they can produce an output with size exponential in the size of the input, whereas an MSO tree transducer can enlarge the input by a constant factor only (the copy number). For this reason we restrict the fatts by adding the weak single use requirement (Definition 1.35), for the second phase.

Definition 4.10. *Let* Σ, Γ *be operator alphabets. A* plain WSUR attributed tree transducer (satt) *is an* WSUR *plain att. An* WSUR attributed tree transducer with flags (sfatt) *is an fatt that is phase-2* WSUR *with respect to* (F, B)*. The sets of all transductions definable by satts and sfatts are denoted* SATT *and* SFATT *respectively.*

Next, we give an example of a WSUR attributed tree transducer with flags.

Example 4.11. The transduction in Example 4.6 can be computed by the WSUR att with flags $(\Sigma, \Omega, (S, I, W), R, C, \beta)$, defined as follows.

The semantic domains are $\Omega = \{T_\Delta, \{0, 1, \text{many}\}\}$. We have two synthesized attributes, fl and β, with $W(\text{fl}) = \{0, 1, \text{many}\}$ and $W(\beta) = T_\Delta$. Thus, $F = \{\text{fl}\}$, and $B = \{\beta\}$. Intuitively, $\langle \text{fl}, u \rangle$ is the number of occurrences of $*$ in $\text{sub}_t(u)$, where 'many' means 'more than one'.

In $R(\sigma)$ we have the rules

$$\langle \beta, 0 \rangle = \begin{cases} 1(\langle \beta, 1 \rangle) & \text{if } \langle \text{fl}, 1 \rangle = 1 \text{ and } \langle \text{fl}, 2 \rangle = 0, \\ 2(\langle \beta, 2 \rangle) & \text{if } \langle \text{fl}, 1 \rangle = 0 \text{ and } \langle \text{fl}, 2 \rangle = 1, \\ \# & \text{otherwise,} \end{cases}$$

and

$$\langle \text{fl}, 0 \rangle = \langle \text{fl}, 1 \rangle + \langle \text{fl}, 2 \rangle,$$

where addition is to be interpreted in the obvious way ($1 + 1 =$ many, and many plus anything equals many). The rules in $R(\#)$ are $\langle \beta, 0 \rangle = \#$ and $\langle \mathrm{fl}, 0 \rangle = 0$. The rules in $R(*)$ are $\langle \beta, 0 \rangle = *$ and $\langle \mathrm{fl}, 0 \rangle = 1$.

Note that the tree rules in $R(\sigma)$ does not follow the appropriate format (it may only test on one flag). We will use this looser form throughout the remainder of the chapter, since the right format can easily be achieved by adding an extra flag.

We will illustrate this in this example. The proper format is achieved by adding a synthesized flag fl', with semantic domain $W(\mathrm{fl}') = \{1, 2, 3\}$, and the following rule in $R(\sigma)$:

$$\langle \mathrm{fl}', 0 \rangle = \begin{cases} 1 & \text{if } \langle \mathrm{fl}, 1 \rangle = 1 \text{ and } \langle \mathrm{fl}, 2 \rangle = 0, \\ 2 & \text{if } \langle \mathrm{fl}, 1 \rangle = 0 \text{ and } \langle \mathrm{fl}, 2 \rangle = 1, \\ 3 & \text{otherwise.} \end{cases}$$

Apart from that, the only tree rule in $R(\sigma)$ is changed into

$$\langle \beta, 0 \rangle = \begin{cases} 1(\langle \beta, 1 \rangle) & \text{if } \langle \mathrm{fl}', 0 \rangle = 1, \\ 2(\langle \beta, 2 \rangle) & \text{if } \langle \mathrm{fl}', 0 \rangle = 2, \\ \# & \text{if } \langle \mathrm{fl}', 0 \rangle = 3. \end{cases}$$

□

4.3 For every WSUR attributed tree transducer there is an MSO transducer

We show here that every attributed tree transduction defined by an sfatt is an MSO tree transduction.

4.3.1 Some Assumptions

Without loss of generality, we will assume the following for an sfatt.

- For every tree rule

$$\langle \beta, i \rangle = \begin{cases} r_1 & \text{if } \langle \mathrm{fl}, 0 \rangle = e_1 \\ \vdots & \vdots \\ r_N & \text{if } \langle \mathrm{fl}, 0 \rangle = e_N \end{cases}$$

  and all $j \in [1, N]$, not more than one node of r_j is labelled with a label from Δ.

  We can obtain this form by introducing some new tree attributes. We introduce an attribute for every subexpression of r_j, except for the subexpressions that are attributes. As a technical convenience, we view the expressions as trees and assume all trees in all rules of the attribute grammar to be mutually disjoint. For every

$j \in [1, N]$, and for every node v of r_j which has a label $\delta \in \Delta$, we add a synthesized tree attribute $r_{j,v}$. Let $k = \mathrm{rk}(\delta)$. This attribute has the following rule at every $\sigma \in \Sigma$:

$$\langle r_{j,v}, 0\rangle = \delta(w_{v\cdot 1}, \ldots, w_{v\cdot k}),$$

where, for any node u of r_j, w_u is either the attribute giving the subexpression rooted at u, or the attribute that is the label of u:

$$w_u = \begin{cases} \langle r_{j,u}, 0\rangle & \text{if } \mathrm{nlab}_{r_j}(u) \in \Delta, \\ \mathrm{nlab}_{r_j}(u) & \text{otherwise } (\mathrm{nlab}_{r_j}(u) \text{ is an attribute}). \end{cases}$$

Then, in the original rule, r_j is replaced by $\langle r_{j,\mathrm{root}(r_j)}, 0\rangle$, if it is not an attribute, and it stays the same, otherwise.

This construction preserves the WSUR, because an attribute $\langle r_{j,v\cdot m}, u\rangle$ is only used once, to define $\langle r_{j,v}, u\rangle$.

- The semantic conditions are of the form

  $$\langle \mathrm{fl}, 0\rangle = \mathrm{true}$$

  for some $\mathrm{fl} \in F$ with $W(\mathrm{fl}) = \mathbb{B}$.

  This is easily achieved by adding an extra flag for every existing semantic condition, which is true iff the corresponding condition is true. The original condition can then be replaced by a test on this flag.

Copy Rules

Tree rules pose a special problem when one of the cases in the rule merely copies the value of another attribute. A tree rule

$$\langle \beta, i\rangle = \begin{cases} r_1 & \text{if } \langle \mathrm{fl}, 0\rangle = e_1 \\ \vdots & \vdots \\ r_N & \text{if } \langle \mathrm{fl}, 0\rangle = e_N \end{cases}$$

is a *copy rule* if there is a j, such that $r_j \in B \times [0, \mathrm{rks}(\Sigma)]$ (the rule merely copies a value when $\langle \mathrm{fl}, 0\rangle = e_j$).

4.3.2 Plain Attributed Tree Transducers

For the sake of clarity we will first prove that *plain* (i.e., flag-free) WSUR attributed tree transductions are always MSO transductions. We assume that the transducer has no copy rules, we will find a solution for copy rules in Subsection 4.3.4.

The output tree of such an attributed tree transduction is very similar to the dependency graph of the input tree. More precisely, it consists of a part of the dependency graph

with all edges reversed, and labels added. This part is the subgraph of the dependency graph induced by all nodes from which there is a path to $\langle \alpha_{\text{mean}}, \text{root}(t) \rangle$. Every node label is the (single!) label used in the corresponding rule, and edge labels are added appropriately. The single use requirement (and the non-circularity of the AG) makes sure that this part of the dependency graph is indeed a tree.

Example 4.12. We give a simple example of a plain WSUR attributed tree transducer: $T = (\Sigma, \Omega, (S, I, W), R, C, \beta)$. Given a tree t, it reproduces the monadic tree that constitutes the path from the root of t to its leftmost leaf. The input alphabet is the same as in Example 4.6: $\Sigma_2 = \{\sigma\}$, and $\Sigma_0 = \{*, \#\}$. The output alphabet is $\Delta = \Delta_1 \cup \Delta_0$, with $\Delta_1 = \{\sigma\}$, and $\Delta_0 = \{*, \#\}$. The only semantic domain is T_Δ, there is only one synthesized attribute β, which is the meaning attribute, and there are no inherited attributes. The rules are simple: the single rule in $R(\sigma)$ is $\langle \beta, 0 \rangle = \sigma \langle \beta, 1 \rangle$, $R(*)$ holds the single rule $\langle \beta, 0 \rangle = *$ and $R(\#)$ holds the single rule $\langle \beta, 0 \rangle = \#$.

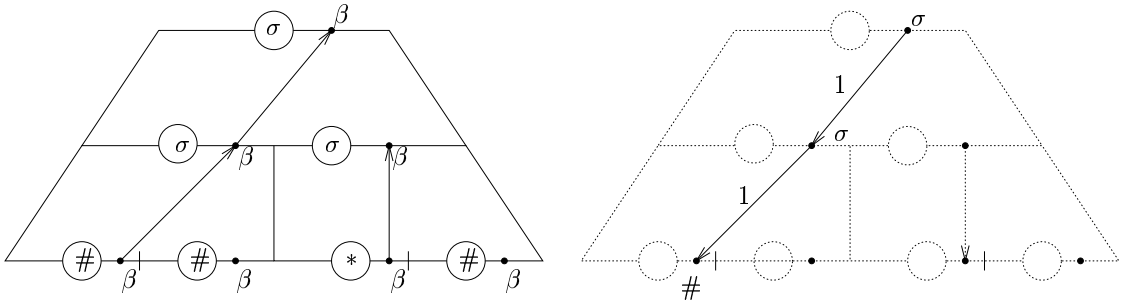

Figure 4.2: Dependency graph and output

In Figure 4.2, for the tree $\sigma\sigma\#\#\sigma * \#$, the dependency graph is given to the left, and the output tree $\sigma\sigma\#$ to the right (ignore the dotted lines). □

Let $G = (\Sigma, \{T_\Delta\}, (S, I, W), R, \alpha_{\text{mean}})$ be a plain WSUR attributed tree transducer without copy rules, from Σ to Δ. The corresponding MSO transducer is

$$T = (B, \phi, \{\psi_{\delta,\beta}\}_{\delta \in \Delta, \beta \in B}, \{\chi_{j,\beta,\beta'}\}_{j \in \text{rks}(\Delta), \beta, \beta' \in B}),$$

where

- The copy set is B, the set of (tree) attributes of G. Thanks to the single use restriction, not more than $\#B$ copies of any node can be made by G.
- We have no semantic conditions, so any tree is allowed and $\phi = \text{true}$.
- The edge formulas check for a dependency. An edge formula $\chi_{j,\beta,\beta'}(x, y)$ checks the semantic rules to see if there is a semantic instruction that defines $\langle \beta, x \rangle$ in terms of

$\langle \beta', y \rangle$. For all $j \in \mathrm{rks}(\Delta)$, $\beta, \beta' \in B$, the edge formulas are

$$\begin{aligned}\chi_{j,\beta,\beta'}(x,y) = \exists i, i' \in [0, \mathrm{rk}(\Sigma)], \exists \sigma \in \Sigma, \exists z \\ (x = z \cdot i \wedge y = z \cdot i' \wedge \mathrm{lab}_\sigma(z) \wedge \exists \delta \in \Delta : \langle \beta, i \rangle = \delta(\cdots, \underset{j}{\langle \beta', i' \rangle}, \cdots) \in R(\sigma)) \vee \\ (\mathrm{root}(x) \wedge x = y \wedge \exists \delta \in \Delta : \langle \beta, 0 \rangle = \delta(\cdots, \underset{j}{\langle \beta', 0 \rangle}, \cdots) \in R_{\mathrm{root}})\end{aligned}$$

- The node formulas determine the right label, and check that the attribute really does contribute to the transduction. For all $\delta \in \Delta$, $\beta \in B$, they are

$$\psi_{\delta,\beta}(x) = \text{right-lab}_{\delta,\beta}(x) \wedge \text{connected-to-}\alpha\text{-mean}_\beta(x),$$

where $\text{right-lab}_{\delta,\beta}(x)$ checks if there is a semantic instruction that assigns $\delta(\cdots)$ to $\langle \beta, x \rangle$:

$$\begin{aligned}\text{right-lab}_{\delta,\beta}(x) = \exists i \in [0, \mathrm{rk}(\Sigma)], \exists \sigma \in \Sigma, \exists z (x = z \cdot i \wedge \mathrm{lab}_\sigma(z) \wedge \langle \beta, i \rangle = \delta(\cdots) \in R(\sigma)) \\ \vee (\mathrm{root}(x) \wedge \langle \beta, 0 \rangle = \delta(\cdots) \in R_{\mathrm{root}}),\end{aligned}$$

and $\text{connected-to-}\alpha\text{-mean}_\beta(x)$ checks if the meaning attribute of the root depends on attribute β of x, as follows (where $B = \{\beta_1, \ldots, \beta_L\}$):

$$\begin{aligned}\text{connected-to-}\alpha\text{-mean}_\beta(z) = \exists X_{\beta_1} \ldots X_{\beta_L} \Big(\forall l \in [1, L], \forall y \in X_{\beta_l} \Big((y = z \wedge \beta_l = \beta) \vee \\ (\exists l', \exists y' \in X_{\beta_{l'}}, \exists j : \chi_{j,\beta_l,\beta_{l'}}(y, y')) \Big) \wedge (\forall x (\mathrm{root}(x) \to x \in X_{\alpha_{\mathrm{mean}}})) \Big).\end{aligned}$$

To understand the latter formula, note that in an acyclic graph, $\mathrm{path}(x, z)$ is equivalent to

$$\mathrm{acpath}(x, z) = \exists X \Big(\forall y \in X (y = z \vee \exists y' \in X : \mathrm{edg}(y, y')) \wedge x \in X \Big).$$

If we translate this formula from the output tree of the transduction to the input tree, using the usual method of [Cou91], we obtain the formula

$$\begin{aligned}\mathrm{conn}_{\alpha,\beta}(x, z) = \exists X_{\beta_1} \ldots X_{\beta_L} \Big(\forall l \in [1, L], \forall y \in X_{\beta_l} \Big((y = z \wedge \beta_l = \beta) \vee \\ (\exists l', \exists y' \in X_{\beta_{l'}}, \exists j : \chi_{j,\beta_l,\beta_\alpha}(y, y')) \Big) \wedge x \in X_\alpha \Big)\end{aligned}$$

We now obtain $\text{connected-to-}\alpha\text{-mean}_\beta(z)$ by filling in $\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t) \rangle$ as starting point, i.e.,

$$(t, v) \models \text{connected-to-}\alpha\text{-mean}_\beta(z) \iff (t, \mathrm{root}(t), v) \models \mathrm{conn}_{\alpha_{\mathrm{mean}},\beta}(x, z).$$

Proof of Correctness

We will show that the above way to construct an MSO transducer from a WSUR plain attributed tree transducer without copy rules is indeed correct. Please keep in mind that tree rules have exactly one operator.

We first define the semantic graph of an att G on a tree t. The semantic graph of a tree t is the same as the dependency graph $\mathrm{WD}_G(t)$ of t, except that it has all the edges reversed, and is labelled. The nodes are labelled with the single operator $\delta \in \Delta$ used in the semantic instruction associated with the node, and the edges are labelled according to the order of the attributes in the instruction. Also, it is close to the transduction of t: as will be proven below, the transduction of t is the subtree of the semantic graph rooted in $\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t) \rangle$.

Definition 4.13. *The* semantic graph $S_G(t)$ *of a plain* WSUR *attributed tree transducer* G *from* Σ *to* Δ *on a tree* $t \in T_\Sigma$ *is a forest over* Δ *(see Definition 1.22).*

$$S_G(t) = (V, E, \mathrm{nlab}, \mathrm{elab}),$$

where

$$\begin{aligned}
V &= A(t), \\
E &= \Big\{ ((\alpha, u), (\alpha', u')) \mid \exists \delta \in \Delta : (\alpha, u) = \delta(\cdots, (\alpha', u'), \cdots) \in R(t) \Big\} \\
\mathrm{nlab} &= \Big\{ ((\alpha, u), \delta) \mid (\alpha, u) = \delta(\cdots) \in R(t) \Big\}, \text{ and} \\
\mathrm{elab} &= \Big\{ \Big(((\alpha, u), (\alpha', u')), j \Big) \mid \exists \delta \in \Delta : (\alpha, u) = \delta(\cdots, \underset{j}{(\alpha', u')}, \cdots) \in R(t) \Big\}.
\end{aligned}$$

Lemma 4.14. *For every plain* WSUR *attributed tree transducer* G *without copy rules there is an* MSO *tree transducer* T *with* $\mathcal{T} = \mathcal{G}$.

Proof. Since the attribute grammar is non-circular and WSUR, the semantic graph is acyclic and every node has at most one incoming edge, so $S_G(t)$ is a forest over Δ. We now prove that $\mathcal{G}(t) = \mathrm{sub}_{S_G(t)}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t) \rangle) = \mathcal{T}(t)$, where the MSO transducer T is obtained from G in the manner described above, see Figure 4.3.

We will do this in two parts: first we prove $\mathcal{G}(t) = \mathrm{sub}_{S_G(t)}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t) \rangle)$, and second $\mathcal{T}(t) = \mathrm{sub}_{S_G(t)}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t) \rangle)$.

1. $\mathcal{G}(t) = \mathrm{sub}_{S_G(t)}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t) \rangle)$. Since t has a unique decoration (see Proposition 1.33), $\mathrm{dec}_{G,t}$, it suffices to show that $\mathrm{sub}_{S_G(t)}$ is a decoration. When this is established, it follows that $\mathrm{dec}_{G,t} = \mathrm{sub}_{G,t}$, and hence that $\mathcal{G}(t) = \mathrm{dec}_{G,t}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t) \rangle) = \mathrm{sub}_{G,t}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t) \rangle)$.

   According to Definition 1.30, a function is a decoration if all semantic instructions in $R(t)$ are obeyed. If there is a semantic instruction

$$\langle \alpha_0, u_0 \rangle = \delta(\langle \alpha_1, u_1 \rangle, \ldots, \langle \alpha_k, u_k \rangle),$$

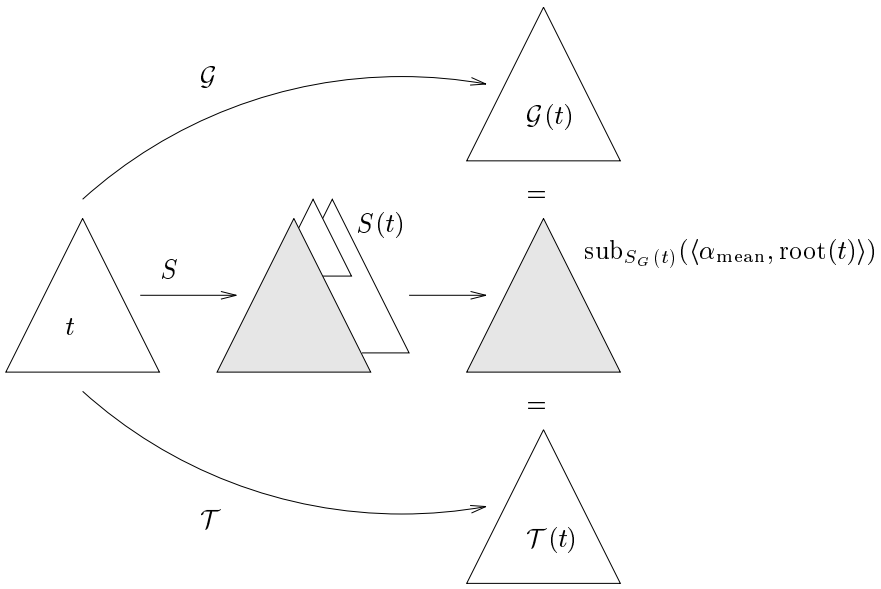

Figure 4.3: How the transductions relate to the semantic graph

then $\mathrm{nlab}_{S_G(t)}(\langle \alpha_0, u_0\rangle) = \delta$, and for all $j \in [1,k] : \langle \alpha_0, u_0\rangle \xrightarrow{j}_{S_G(t)} \langle \alpha_j, u_j\rangle$, and hence, since $S_G(t)$ is a forest over Δ,

$$\mathrm{sub}_{S_G(t)}(\langle \alpha_0, u_0\rangle) = \delta(\mathrm{sub}_{S_G(t)}(\langle \alpha_1, u_1\rangle), \ldots, \mathrm{sub}_{S_G(t)}(\langle \alpha_k, u_k\rangle)),$$

i.e., $\mathrm{sub}_{S_G(t)}$ obeys the semantic instructions.

2. $\mathcal{T}(t) = \mathrm{sub}_{S_G(t)}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t)\rangle)$. Note that the equality only holds on the level of abstract graphs: we identify the nodes $\langle v, \beta\rangle$ of $\mathcal{T}(t)$ with the nodes $\langle \beta, v\rangle$ in $\mathrm{sub}_{S_G(t)}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t)\rangle)$.

   We will show that if we drop the connected-to-α-$\mathrm{mean}_\beta(x)$ from the node formulas, the transduction of t is equal to $S_G(t)$. Because $(t,u) \models$ connected-to-α-$\mathrm{mean}_\beta(x)$ if and only if $(S_G(t), \langle \alpha_{\mathrm{mean}}, \mathrm{root}(t)\rangle, \langle \beta, u\rangle) \models \mathrm{acpath}(x,y)$, it then follows that $\mathcal{T}(t) = \mathrm{sub}_{S_G(t)}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t)\rangle)$.

   Let T' be T, disregarding the connected-to-α-mean formulas:

$$T' = (B, \mathrm{true}, \{\text{right-lab}_{\delta,\beta}\}_{\delta\in\Delta,\beta\in B}, \{\chi_{j,\beta,\beta'}\}_{j\in \mathrm{rks}(\Delta),\beta,\beta'\in B}).$$

   It is obvious that for any $t \in T_\Sigma$, $\mathcal{T}'(t) = S_G(t)$, from the way the semantic instructions are derived from the semantic rules (Definition 1.29). More exactly, $(t,u) \models \text{right-lab}_{\delta,\beta}(x)$ iff there is a semantic instruction $(\beta, u) = \delta(\cdots) \in R(t)$, and $(t,u,v) \models \chi_{j,\beta,\beta'}(x,y)$ iff there is a semantic instruction $(\beta, u) = \delta(\cdots, \underset{j}{(\beta', v)}, \cdots) \in R(t)$ for certain $\delta \in \Delta$. Hence, $\mathcal{T}'(t) = S_G(t)$.

□

4.3.3 Attributed Tree Transducers with Flags

We now show that any WSUR fatt is an MSO transduction. We still assume that the transducer has no copy rules; we postpone handling copy rules to Subsection 4.3.4.

By Lemma 2.15, for each flag $\mathrm{fl} \in F$ and each $e \in W(\mathrm{fl})$, we can make an MSO formula $\omega_{\mathrm{fl}=e}(x)$, with $\forall t \in T_\Sigma, \forall u \in V_t : (t,u) \models \omega_{\mathrm{fl}=e}(x)$ iff $\langle \mathrm{fl}, u\rangle_t = e$. From here on we simply write $\mathrm{fl}(x) = e$ for $\omega_{\mathrm{fl}=e}(x)$.

Let G be a WSUR attributed tree transducer from Σ to Δ without copy rules. The corresponding MSO transducer is

$$T = (B, \phi, \{\psi_{\delta,\beta} \mid \delta \in \Delta, \beta \in B\}, \{\chi_{j,\beta,\beta'} \mid j \in \mathrm{rks}(\Delta), \beta, \beta' \in B\}),$$

where

- The copy set B is the set of tree attributes.
- The domain formula checks whether all semantic conditions hold:

$$\phi = \forall x, \forall \sigma \in \Sigma, \forall\, \mathrm{fl} \in F((\mathrm{lab}_\sigma(x) \wedge \text{“}C(\sigma) \text{ is } \langle \mathrm{fl}, 0\rangle = \mathrm{true}\text{”}) \to \mathrm{fl}(x) = \mathrm{true})$$

- Again, the edge formulas check the dependencies. Because of the case structure of the rules, the edge formulas are a bit more complicated here. For all $j \in \mathrm{rks}(\Delta), \beta, \beta' \in B$, they are:

$$\begin{aligned}
\chi_{j,\beta,\beta'}(x,y) = \exists i, i' \in [0, \mathrm{rk}(\Sigma)], \exists \sigma \in \Sigma, \exists z, \exists\, \mathrm{fl} \in F, \exists e \in W(\mathrm{fl}) \\
\Big(x = z \cdot i \wedge y = z \cdot i' \wedge \mathrm{lab}_\sigma(z) \wedge \mathrm{fl}(z) = e \wedge \\
\exists \delta \in \Delta : \langle \beta, i\rangle = \left\{ \begin{array}{ll} \vdots & \vdots \\ \delta(\cdots, \underset{j}{\langle \beta', i'\rangle}, \cdots) & \text{if } \langle \mathrm{fl}, 0\rangle = e \\ \vdots & \vdots \end{array} \right\} \in R(\sigma)\Big) \vee \\
\exists\, \mathrm{fl} \in F, \exists e \in W(\mathrm{fl})\Big(\mathrm{root}(x) \wedge x = y \wedge \mathrm{fl}(x) = e \wedge \\
\exists \delta \in \Delta : \langle \beta, 0\rangle = \left\{ \begin{array}{ll} \vdots & \vdots \\ \delta(\cdots, \underset{j}{\langle \beta', 0\rangle}, \cdots) & \text{if } \langle \mathrm{fl}, 0\rangle = e \\ \vdots & \vdots \end{array} \right\} \in R_{\mathrm{root}}\Big).
\end{aligned}$$

- The node formulas for all $\delta \in \Delta, \beta \in B$ are

$$\psi_{\delta,\beta}(x) = \text{right-lab}_{\delta,\beta}(x) \wedge \text{connected-to-}\alpha\text{-mean}_\beta(x),$$

where

$$\text{right-lab}_{\delta,\beta}(x) = \exists i \in [0, \text{rk}(\Sigma)], \exists \sigma \in \Sigma, \exists z, \exists\, \text{fl} \in F, \exists e \in W(\text{fl})$$
$$\Big(x = z \cdot i \wedge \text{lab}_\sigma(z) \wedge \text{fl}(z) = e \wedge$$
$$\langle \beta, i \rangle = \left\{ \begin{array}{ll} \vdots & \vdots \\ \delta(\cdots) & \text{if } \langle \text{fl}, 0 \rangle = e \\ \vdots & \vdots \end{array} \right\} \in R(\sigma)\Big) \vee$$

$$\exists\, \text{fl} \in F, \exists e \in W(\text{fl})\Big(\text{root}(x) \wedge \text{fl}(x) = e \wedge \langle \beta, 0 \rangle = \left\{ \begin{array}{ll} \vdots & \vdots \\ \delta(\cdots) & \text{if } \langle \text{fl}, 0 \rangle = e \\ \vdots & \vdots \end{array} \right\} \in R_{\text{root}}\Big),$$

and connected-to is defined like before:

$$\text{connected-to-}\alpha\text{-mean}_\beta(z) = \exists X_{\beta_1} \ldots X_{\beta_L} \Big(\forall l \in [1, L], \forall y \in X_{\beta_l} \Big((y = z \wedge \beta_l = \beta) \vee$$
$$(\exists l', \exists y' \in X_{\beta_{l'}}, \exists j : \chi_{j,\beta_l,\beta_{l'}}(y, y'))\Big) \wedge (\forall x(\text{root}(x) \to x \in X_{\alpha_{\text{mean}}}))\Big).$$

Proof of Correctness

The proof for WSUR fatts G without copy rules is much the same as for plain attributed tree transducers without copy rules. Again, we define a semantic graph. The semantic graph of a tree t assumes the phase-1 decoration $\text{dec}^1_{G,t}$ of t to be known. The graph is a labelled version of the phase-2 weak-dependency graph $\text{WD}^2_G(t)$ of t, with the edges reversed. The node labels are the ones appropriate considering the phase-1 decoration, and the edges are labelled in order.

Definition 4.15. *The semantic graph $S_G(t)$ of a* WSUR *attributed tree transducer with flags G from Σ to Δ on a tree $t \in T_\Sigma$ is a forest over Δ.*

$$S_G(t) = (V, E, \text{nlab}, \text{elab}),$$

where

$$V = B \times V_t,$$
$$E = \Big\{((\beta, u \cdot i), (\beta', u \cdot i')) \mid \exists\, \text{fl} \in F, \exists e \in W(\text{fl})(\text{dec}^1(\langle \text{fl}, u \rangle) = e \text{ and}$$
$$\exists \delta \in \Delta : \langle \beta, u \cdot i \rangle = \left\{ \begin{array}{ll} \vdots & \vdots \\ \delta(\cdots, \langle \beta', u \cdot i' \rangle, \cdots) & \text{if } \langle \text{fl}, u \rangle = e \\ \vdots & \vdots \end{array} \right\} \in R(t))\Big\}$$
$$\text{nlab} = \Big\{((\beta, u \cdot i), \delta) \mid \exists\, \text{fl} \in F, \exists e \in W(\text{fl})(\text{dec}^1(\langle \text{fl}, u \rangle) = e \text{ and}$$
$$\langle \beta, u \cdot i \rangle = \left\{ \begin{array}{ll} \vdots & \vdots \\ \delta(\cdots) & \text{if } \langle \text{fl}, u \rangle = e \\ \vdots & \vdots \end{array} \right\} \in R(t))\Big\}, \text{ and}$$

$$\mathrm{elab} = \Big\{\Big(((\beta, u\cdot i),(\beta, u\cdot i')), j\Big) \mid \exists\, \mathrm{fl} \in F, \exists e \in W(\mathrm{fl})(\mathrm{dec}^1(\langle \mathrm{fl}, u\rangle) = e \text{ and}$$
$$\exists \delta \in \Delta\langle \beta, u\cdot i\rangle = \left\{ \begin{array}{ll} \vdots & \vdots \\ \delta(\cdots, \langle \beta', \underset{j}{u\cdot i'}\rangle, \cdots) & \text{if } \langle \mathrm{fl}, u\rangle = e \\ \vdots & \vdots \end{array} \right\} \in R(t))\Big\}.$$

The semantic graph $S_G(t)$ is equal to the phase-2 dependency graph $\mathrm{WD}^2(t)$, with the exception of the labels and the direction of the edges. This should be clear from the following two facts.

1. The nodes are the same: $A_2(t) = B \times V_t$.
2. The edges (apart from their direction) are also the same. Consider a semantic instruction
$$\langle \beta, u\cdot i\rangle = \begin{cases} f_1(\langle \beta_{11}, u\cdot i_{11}\rangle, \ldots, \langle \beta_{1k_1}, u\cdot i_{1k_1}\rangle) & \text{if } \langle \mathrm{fl}, u\rangle = e_1, \\ \vdots & \vdots \\ f_N(\langle \beta_{N1}, u\cdot i_{N1}\rangle, \ldots, \langle \beta_{Nk_N}, u\cdot i_{Nk_N}\rangle) & \text{if } \langle \mathrm{fl}, u\rangle = e_N \end{cases}$$
in $R(t)$, and a phase-1 decoration that has $\mathrm{dec}^1_{G,t}(\langle \mathrm{fl}, u\rangle) = e_j$. From this instruction we obtain a phase-2 instruction in $R^2(t)$ that is equivalent to
$$\langle \beta, u\cdot i\rangle = f_j(\langle \beta_{j1}, u\cdot i_{j1}\rangle, \ldots, \langle \beta_{jk_j}, u\cdot i_{jk_j}\rangle),$$
keeping only the dependencies from the sub-instruction for $\langle \mathrm{fl}, u\rangle = e_j$. For the edges of $S_G(t)$, we consider all tree rules, i.e., exactly those rules that give rise to a phase-2 instruction in $R^2(t)$, and we take into account the correct dependencies, by considering the phase-1 decoration.

Note that the semantic graph is indeed a forest, since the attributed tree transducer is weakly non-circular and WSUR with respect to (F, B).

Lemma 4.16. *For every* WSUR *fatt G without copy rules there is an* MSO *tree transducer T with $\mathcal{T} = \mathcal{G}$.*

Proof. Let T be the MSO tree transducer derived for T in the manner described above. Phase 2 of G is unconditional. Clearly, $t \models \phi$ iff $\mathrm{dec}^1_{G,t}$ satisfies the semantic tests, so $\mathrm{dom}(\mathcal{T}) = \mathrm{dom}(\mathcal{G})$.

Assume $t \in T_\Sigma$ with $t \models \phi$. We prove $\mathcal{G}(t) = \mathrm{sub}_{S_G(t)}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t)\rangle) = \mathcal{T}(t)$, in the same way as before.

1. $\mathcal{G}(t) = \mathrm{sub}_{S_G(t)}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t)\rangle)$. The decomposition in phases of G is (F, B). Since α_{mean} is a tree attribute, and thus in phase 2, we only have to show that $\mathrm{sub}_{S_G(t)}$ is a phase-2 decoration, i.e., it satisfies all rules in $R^2(t)$ (see Lemma 1.41). We consider

$\mathrm{dec}^1_{G,t}$ to be known. It is clear how the semantic instructions are divided over the phases: the phase-1 instructions define the flags, and the phase-2 instructions define the tree attributes.

We consider the semantic instructions in $R(t)$ that give rise to a phase-2 instruction in $R^2(t)$. Let

$$\langle \beta_0, u \cdot i_0 \rangle = \begin{cases} \vdots & \vdots \\ \delta(\langle \beta_1, u \cdot i_1 \rangle, \ldots, \langle \beta_k, u \cdot i_k \rangle) & \text{if } \langle \mathrm{fl}, u \rangle = e \\ \vdots & \vdots \end{cases}$$

be a semantic instruction in $R(t)$, and let $\mathrm{dec}^1(\langle \mathrm{fl}, u \rangle) = e$. Then, $\mathrm{nlab}_{S_G(t)}(\langle \beta_0, u \cdot i_0 \rangle) = \delta$, and for all $j \in [1,k] : \langle \beta_0, u \cdot i_0 \rangle \xrightarrow{j}_{S_G(t)} \langle \beta_j, u \cdot i_j \rangle$, and therefore $\mathrm{sub}_{S_G(t)}(\langle \beta_0, u \cdot i_0 \rangle) = \delta(\mathrm{sub}_{S_G(t)}(\langle \beta_1, u \cdot i_1 \rangle), \ldots, \mathrm{sub}_{S_G(t)}(\langle \beta_k, u \cdot i_k \rangle))$, and hence, since $S_G(t)$ is a forest over Δ, $\mathrm{sub}_{S_G(t)}$ obeys the phase-2 semantic instruction corresponding to the above instruction.

2. $\mathcal{T}(t) = \mathrm{sub}_{S_G(t)}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t) \rangle)$. This follows the same line of reasoning as case 2 for plain attributed tree transducers.

□

4.3.4 Copy Rules

The process described above is not suitable for attributed tree transducers with copy rules. We now show how to construct an MSO transducer for an att with copy rules.

Let G be an sfatt with copy rules.

- First we change the attributed tree transducer. Assume $* \notin \Delta$. By $\Delta \cup \{*\}$ we mean the operator alphabet with the same rank function as Δ has, and $\mathrm{rk}_{\Delta \cup \{*\}}(*) = 1$. Let G' be the sfatt from Σ to $\Delta \cup \{*\}$ obtained by replacing every case $\langle \beta, i \rangle$, of every right-hand side of every rule in G by $*(\langle \beta, i \rangle)$. For example, a rule

$$\langle \beta, 0 \rangle = \begin{cases} \delta(\langle \beta, 1 \rangle, \langle \beta', 2 \rangle) & \text{if } \mathrm{fl} = 1 \\ \langle \beta, 1 \rangle & \text{if } \mathrm{fl} = 2, \end{cases}$$

  is a copy rule because of the second case. It is transformed to

$$\langle \beta, 0 \rangle = \begin{cases} \delta(\langle \beta, 1 \rangle, \langle \beta', 2 \rangle) & \text{if } \mathrm{fl} = 1 \\ *(\langle \beta, 1 \rangle) & \text{if } \mathrm{fl} = 2, \end{cases}$$

  Please note that the output of G is equal to that of G', with the $*$s "cut out".

- Because G' has no copy rules, we can build an MSO transducer T' with $\mathcal{T}' = \mathcal{G}'$.

- We build an MSO transducer T'' from $\Delta \cup \{*\}$ to Δ, that cuts all nodes labelled $*$ from a tree. The transduction has copy number 1, we leave the subscripted c out.

$$T'' = (\{c\}, \text{true}, \{\psi_\delta\}_{\delta\in\Delta}, \{\chi_j\}_{j\in \text{rks}(\Delta)}),$$

  where for every $\delta \in \Delta$,

$$\psi_\delta(x) = \text{lab}_\delta(x),$$

  so we copy exactly those nodes that do not have label $*$. For the edge formulas we give regular path expressions. By the path expressions we mean, of course, the corresponding MSO formulas (Theorem 3.16). For all $j \in \text{rks}(\Delta)$,

$$\chi_j(x, y) = \downarrow_j \cdot(\text{lab}_*(z)\cdot \downarrow_1)^*.$$

  Thus, an edge is drawn between two nodes whenever they are connected through nodes labelled $*$ alone.

- Note that $\mathcal{T}';\mathcal{T}'' = \mathcal{G}$, and that MSO transductions are closed under composition [Cou91]. This implies that there is an MSO transducer T, with $\mathcal{T} = \mathcal{G}$.

We have now handled all cases, and can conclude that for any WSUR attributed tree transducer with flags and copy rules, there is an equivalent MSO tree transducer.

Lemma 4.17. *For every* WSUR *attributed tree transducer with flags G there is an* MSO *tree transducer T, such that $\mathcal{T} = \mathcal{G}$, or*

$$\text{SFATT} \subseteq \text{MSOTT}\,.$$

4.4 For every MSO tree transducer there is an WSUR tree transducer

We will prove that for every MSO tree transducer there is an equivalent WSUR attributed tree transducer with flags. First we give the intuitive idea, then, in four subsections, we prove some basic properties, we present the formal construction, we give an example, and finally we prove the correctness of the construction.

Let $T = (C, \phi, \Psi, X)$ be an MSO tree transducer from Σ to Δ. We will build a WSUR fatt G in the following manner. Let $t \in \text{dom}(\mathcal{T})$, and $t' = \mathcal{T}(t)$.

With every node u, we have tree attributes β_c for every $c \in C$. If (u, c) is a node of t', attribute $\langle\beta_c, u\rangle$ will hold the value $\text{sub}_{t'}(u, c)$.

A difficult point is that if (v, c') is a child of (u, c) in t', v does not have to be near u in t. We use copy rules to transport the data through the tree, but we have to know the proper route. If (v, c') is the jth child of (u, c) in t', then there is an edge formula $\chi_{j,c,c'}(x, y)$ that is satisfied by (t, u, v). There is also a path language corresponding to $\chi_{j,c,c'}(x, y)$ and an automaton $A_{j,c,c'}$ that recognizes its language. This automaton has a unique walk

$$(u, q_0) \twoheadrightarrow_t \ldots \twoheadrightarrow_t (v, q_f) \quad \text{for some } q_f \in F_{A_{j,c,c'}}\ ,$$

along the shortest path from u to v. We will transport the data backwards from $\langle \beta_{c'}, v \rangle$ to $\langle \beta_c, u \rangle$ using attributes $\beta_{(j,c,c'),q}$ (cf. the proof of Lemma 3.20). An attribute $\langle \beta_{(j,c,c'),q}, w \rangle$ will hold the value $\mathrm{sub}_{t'}(\langle v, c' \rangle)$ if (w, q) is on the walk of the automaton. The dependencies will run parallel to the walk of the automaton: $\langle \beta_{(j,c,c'),q}, w \rangle$ depends on $\langle \beta_{(j,c,c'),q'}, w' \rangle$ iff $(w, q) \twoheadrightarrow (w', q')$ is a part of the walk of the automaton. A problem here is that $\langle \beta_{(j,c,c'),q}, w \rangle$ may depend both on attributes of children of w and on attributes of the parent of w. Yet, we have to choose $\langle \beta_{(j,c,c'),q}, w \rangle$ to be either synthesized or inherited. This problem is circumvented by making $\beta_{(j,c,c'),q}$ synthesized, and adding an extra inherited attribute $\beta_{(j,c,c'),q,*}$.

The last thing is to make the value of $\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t) \rangle$ equal to t'. If (u_r, c_r) is the root of t', then the value of $\langle \beta_{c_r}, u_r \rangle$ (which is t') is transported to $\mathrm{root}(t)$ by copy rules. This is done by synthesized attribute β_{root}.

4.4.1 Some Basic Properties

Let $T = (C, \phi, \Psi, X)$ be an MSO tree transducer from Σ to Δ. Please note that in this section, we will make use of the four assumptions given in Remark 4.3. These assumptions are equivalent to the following.

- if $(t, u) \models \psi_{\delta,c}(x)$, then $t \models \phi$, and $(u, c) \in V_{\mathcal{T}(t)}$, and
- if $(t, u, v) \models \chi_{j,c,c'}(x, y)$, then $t \models \phi$, and $((u, c), (v, c')) \in E_{\mathcal{T}(t)}$.

We have three lemmata.

Lemma 4.18. *For all $t \in T_\Sigma$ with $t \models \phi$, all $(u, c) \in V_{\mathcal{T}(t)}$ with label δ, and all $j \in \mathrm{rks}(\Delta)$,*

$$\exists!(v, c') \in V_{\mathcal{T}(t)}((t, u, v) \models \chi_{j,c,c'}(x, y)) \quad \text{if } j \leq \mathrm{rk}(\delta),$$

and

$$\neg\exists(v, c') \in V_{\mathcal{T}(t)}((t, u, v) \models \chi_{j,c,c'}(x, y)) \quad \text{if } j > \mathrm{rk}(\delta).$$

This is the *unique destination property (udp)*, cf. [KS93]. It follows from the fact that $\mathcal{T}(t)$ is a tree, so a node w has exactly one outgoing edge labelled j for all $j \in [1, \mathrm{rk}(w)]$, and the fact that the edge formulas are mutually exclusive. The latter condition is needed, because otherwise two different edge formulas could be true, resulting in no edge being drawn in the output.

Lemma 4.19. *For all $t \in T_\Sigma$ with $t \models \phi$, and all $(v, c') \in V_{\mathcal{T}(t)}$, there is at most one $(u, c) \in V_{\mathcal{T}(t)}$, such that there is a $j \in \mathrm{rks}(\Delta)$ with $(t, u, v) \models \chi_{j,c,c'}(x, y)$.*

This is the *unique source property (usp)*. Again, it depends on the edge formulas being mutually exclusive, and the fact that $\mathcal{T}(t)$ is a tree, so no node has two incoming edges.

According to Definition 1.43, for every j, c, c' and every tree $t \in T_\Sigma$, $\chi_{j,c,c'}(x,y)$ defines the relation $R_t(\chi_{j,c,c'}(x,y)) = \{(u,v) \mid (t,u,v) \models \chi_{j,c,c'}(x,y)\}$. As a special case of the unique destination property (using the assumptions in Remark 4.3), $R_t(\chi_{j,c,c'}(x,y))$ is a partial function on V_t. Because of the unique source property, this partial function is injective.

In order to simulate the edge formulas of the transducer, we need path languages. For every $\chi_e(x,y)$ (with $e = j,c,c'$), we build a regular path language Π_e over $D_{\Sigma,\mathrm{rks}(\Sigma)}$ (Theorem 3.16). For the regular language we construct a deterministic finite state (string) automaton $A_e = (Q_e, D(\Pi_e), \delta_e, q_{e,0}, F_e)$ accepting Π_e. These automata play an important role in the following. We will also denote an automaton A_e simply by $e = (j,c,c') \in \mathrm{rks}(\Delta) \times C \times C$, writing, for example $\twoheadrightarrow_e$, instead of $\twoheadrightarrow_{A_e}$.

The walks of these automata are very restricted. This is of major importance to the rules of the attribute grammar we are to present. The next lemma describes the *unique walk property (uwp)*.

Lemma 4.20. *Let $t \in T_\Sigma$ and $e \in \mathrm{rks}(\Delta) \times C \times C$. For every $(w,q) \in V_t \times Q_e$, if there are $u, v \in V_t$ such that*

$$(u, q_{e,0}) \twoheadrightarrow^*_{e,t} (w,q) \twoheadrightarrow^*_{e,t} (v, q_f), \quad \textit{for some } q_f \in F_e,$$

then there are unique $u_0, \ldots, u_n \in V_t$, unique $q_0, \ldots, q_n \in Q_e$, and unique $d_1, \ldots, d_n \in D(\Pi_e)$ such that $q_0 = q_{e,0}$, $q_n \in F_e$,

$$(u_0, q_0) \overset{d_1}{\twoheadrightarrow}_{e,t} (u_1, q_1) \overset{d_2}{\twoheadrightarrow}_{e,t} \ldots \overset{d_n}{\twoheadrightarrow}_{e,t} (u_n, q_n),$$

and $(w,q) = (u_i, q_i)$ for some $i \in [0,n]$.

Proof. Remember that Lemma 3.14 states that if A is a deterministic finite state string automaton corresponding to a regular path language, and $(u,v) \in R_t(\|A\|)$, there is a unique walk $(u,q_0) \overset{d_1}{\twoheadrightarrow}_{A,t} \ldots \overset{d_n}{\twoheadrightarrow}_{A,t} (v, q_f)$ with $q_f \in F_A$.

Because $R_t(\|A_e\|) = R_t(\chi_e(x,y))$ is an injective partial function, for every node u there is at most one node v such that $(u, q_{e,0}) \twoheadrightarrow^*_e (v, q_f)$ for some $q_f \in F_e$, and vice-versa, for every node v there is at most one node u such that $(u, q_{e,0}) \twoheadrightarrow^*_e (v, q_f)$ for some $q_f \in F_e$. This implies that for every configuration (w,q) of A_e for which there are nodes u, v such that $(u, q_{e,0}) \twoheadrightarrow^*_e (w,q) \twoheadrightarrow^*_e (v, q_f)$ for some $q_f \in F_e$, u and v are uniquely determined. This implies that the entire walk is determined. □

Note that this result means that all walks of a given automaton A_e are disjoint.

4.4.2 Construction

Let $T = (C, \phi, \Psi, X)$ be an MSO tree transducer from Σ to Δ. We now build an sfatt $G = (\Sigma, \Omega, (S, I, W), R, C, \alpha_{\mathrm{mean}})$ from Σ to $\Delta \cup \{\bot\}$ that simulates T, using the automata A_e defined above. Here, $\Delta \cup \{\bot\}$ has the same rank function as Δ, but $\mathrm{rk}_{\Delta\cup\{\bot\}}(\bot) = 0$. We make sure that for all t in the domain of $\mathcal{G}$, $\mathcal{G}(t) \in T_\Delta$. We proceed in this way, because we want to have an extra element to signify that the value of a tree attribute is 'undefined'.

The Input Alphabet and the Semantic Domains

The input alphabet Σ is already given, and the set of semantic domains Ω is equal to $\{\mathbb{B}, [0, \mathrm{rk}(\Sigma)], T_{\Delta\cup\{\perp\}}\}$.

The Attributes

We first describe the attributes, with their intended values. The semantic rules will be given later.

With every tree attribute we have a boolean flag, 'def', that is 'true' iff the attribute has a proper value. The value of the attribute is set to $\perp$, whenever the corresponding def flag is false. The latter step is not necessary, since the tree attribute is not used if its corresponding def flag is false, but it makes for convenient reading.

The value of the def flags, and indeed the value of all flags used, can be computed by MSO formulas. Recall that the attributed tree transducer has decomposition in phases (F, B), where the first phase constitutes an attribute grammar with finite semantic domains in its own right. This implies that Theorem 2.14 gives a method to compute their values. Hence, we omit details like whether the flags are inherited or synthesized.

First we give these boolean **def flags**, then the corresponding tree attributes. Along with the attributes, we give their intended value, for every $t \in T_\Sigma$ and $u \in V_t$, denoting $\mathcal{T}(t)$ by t' whenever $t \models \phi$.

- def_c. For all $c \in C$.
  $\langle \mathrm{def}_c, u\rangle$ = true iff $\exists \delta : (t, u) \models \psi_{\delta,c}(x)$; that is, iff $(u, c) \in V_{t'}$.

- $\mathrm{def}_{e,q}$. For all $e = (j, c, c') \in \mathrm{rks}(\Delta) \times C \times C$ and $q \in Q_e$.
  $\langle \mathrm{def}_{e,q}, u\rangle$ = true iff there are nodes v, w such that $(v, q_{e,0}) \twoheadrightarrow_e^* (u, q) \twoheadrightarrow_e^* (w, q_f)$ for some $q_f \in F_e$; i.e., if the configuration (u, q) is on a walk of the automaton A_e. Note that, if it exists, this walk is unique because of the unique walk property for (u, q), and hence, if v, w exist, they are also unique.

- $\mathrm{def}_{e,q,*}$. For all $e = (j, c, c') \in \mathrm{rks}(\Delta) \times C \times C$ and $q \in Q_e$.
  For all $l \in [1, \mathrm{rk}(u)]$, $\langle \mathrm{def}_{e,q,*}, u \cdot l\rangle$ = true iff $\exists v, w, \exists q' \in Q_e$ such that $(v, q_{e,0}) \twoheadrightarrow_e^* (u \cdot l, q') \overset{\uparrow_l}{\twoheadrightarrow}_e (u, q) \twoheadrightarrow_e^* (w, q_f)$, for some $q_f \in F_e$. If these v, w exist, they are unique. Moreover, $\langle \mathrm{def}_{e,q,*}, \mathrm{root}(t)\rangle$ = false. Note that for every u, there is at most one l such that $\langle \mathrm{def}_{e,q,*}, u \cdot l\rangle$ = true. This follows from the unique walk property for (u, q).

- $\mathrm{def}_{\mathrm{root}}$.
  $\langle \mathrm{def}_{\mathrm{root}}, u\rangle$ = true iff $t \models \phi$ and $\mathrm{root}(t') = (v, c)$ for some $c \in C$ and $v \in \mathrm{sub}_t(u)$.

Note that all defs are false if t does not satisfy ϕ.

Now the **tree attributes** themselves. Again, we give the intended values for every $t \in T_\Sigma$ and $u \in V_t$, denoting $\mathcal{T}(t)$ by t' whenever $t \models \phi$.

- β_c. Synthesized, for all $c \in C$.
  This attribute gives the subtree of t', rooted in (u, c), if (u, c) is a node of t'.

$$\langle \beta_c, u \rangle = \begin{cases} \mathrm{sub}_{t'}(u, c) & \text{if } \langle \mathrm{def}_c, u \rangle, \\ \bot & \text{otherwise.} \end{cases}$$

- $\beta_{e,q}$. Synthesized, for all $e = (j, c, c') \in \mathrm{rks}(\Delta) \times C \times C$, and $q \in Q_e$.
  This attribute is used to 'transport' the values of the $\beta_{c'}$ through the graph, according to the instructions of the automaton A_e. Please note that the direction in which the automaton walks through the tree is exactly opposite to the direction of the attribute dependencies.

$$\langle \beta_{e,q}, u \rangle = \begin{cases} \mathrm{sub}_{t'}(w, c') & \text{if } \langle \mathrm{def}_{e,q}, u \rangle, \text{ and } \exists q_f \in F_e : (u, q) \twoheadrightarrow_e^* (w, q_f), \text{ for } w \in V_t, \\ \bot & \text{otherwise.} \end{cases}$$

  Note that w is uniquely determined as is explained in the description of $\mathrm{def}_{e,q}$.

- $\beta_{e,q,*}$. Inherited, for all $e \in \mathrm{rks}(\Delta) \times C \times C$ and $q \in Q_e$.
  This attribute is used so the synthesized attribute $\beta_{e,q'}$ of a node $u \cdot l$ for some l can depend on the value of $\beta_{e,q}$ at u.

$$\langle \beta_{e,q,*}, u \cdot l \rangle = \begin{cases} \langle \beta_{e,q}, u \rangle & \text{if } \langle \mathrm{def}_{e,q,*}, u \cdot l \rangle \\ \bot & \text{otherwise} \end{cases}$$

- β_{root} is the meaning attribute. Synthesized.

$$\langle \beta_{\mathrm{root}}, u \rangle = \begin{cases} t' & \text{if } \langle \mathrm{def}_{\mathrm{root}}, u \rangle, \\ \bot & \text{otherwise.} \end{cases}$$

The fact that β_c is synthesized is rather arbitrary. As will be shown in the rules, $\langle \beta_c, u \rangle$ depends only on other attributes of u, not on attributes of the parent or children of u. For $\beta_{e,q}$, and $\beta_{e,q,*}$, the situation is a bit more complicated. Attribute $\langle \beta_{e,q}, u \rangle$ would be dependent both on attributes of the parent of u, as on attributes of the children of u. Since this is not possible with a single attribute, the synthesized attribute $\beta_{e,q}$ is joined by the inherited attribute $\beta_{e,q,*}$. See Figure 4.4 for an illustration of their relation (where the solid lines represent copy rules).

Now we describe the **remaining flags**. These all check relatively simple properties. Again, these are properties that can be computed by an MSO formula.
We list the flags together with their intended value for all $t \in T_\Sigma$, and $u \in V_t$, where $t' = \mathcal{T}(t)$ whenever $t \models \phi$.

- dom. $W(\mathrm{dom}) = \mathbb{B}$.
  $\langle \mathrm{dom}, u \rangle = \mathrm{true}$ iff $t \models \phi$, i.e., the tree satisfies the domain formula.

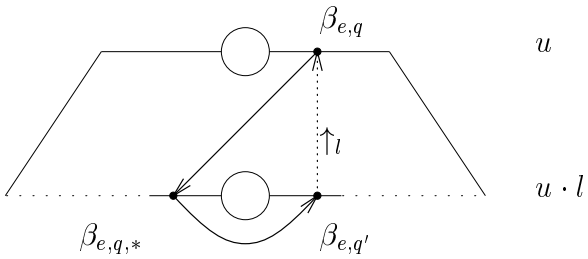

Figure 4.4: Step of the automaton (dotted line) and attribute dependencies (solid lines), when the automaton moves up

- $\psi_{\delta,c}$. $W(\psi_{\delta,c}) = \mathbb{B}$, for all $\delta \in \Delta, c \in C$.
  $\langle \psi_{\delta,c}, u \rangle$ = true iff $(t,u) \models \psi_{\delta,c}(x)$.
- ψ. $W(\psi) = \mathbb{B}$, for all MSO formulas $\psi(x) \in \mathrm{MSOL}_1(\Sigma) \cap D(\Pi_e)$ for some e.
  $\langle \psi, u \rangle$ = true iff $(t,u) \models \psi(x)$.
- chno (child number). $W(\mathrm{chno}) = [0, \mathrm{rk}(\Sigma)]$.
  $\langle \mathrm{chno}, u \rangle = l$ iff $(t,u) \models \exists y : \mathrm{edg}_l(y,x)$, but 0 if $u = \mathrm{root}(t)$.
- root_c. $W(\mathrm{root}_c) = \mathbb{B}$, for all $c \in C$.
  $\langle \mathrm{root}_c, u \rangle$ = true iff $\mathrm{root}(t') = (u,c)$, in other words, $\langle \mathrm{root}_c, u \rangle$ = true iff $(t,u) \models \exists \delta :$ $\psi_{\delta,c}(x) \wedge \neg \exists y, \exists c' \in C, \exists j : \chi_{j,c',c}(y,x)$.

The Semantic Rules

First we give the rules for the **tree attributes**.

In the following formula, let e_j stand for (j, c, c_j). For all $\sigma \in \Sigma, c \in C$,

$$\langle \beta_c, 0 \rangle = \begin{cases} \delta(\langle \beta_{e_1,q_{e_1,0}}, 0 \rangle, \ldots, \langle \beta_{e_k,q_{e_k,0}}, 0 \rangle) & \text{if } \langle \psi_{\delta,c}, 0 \rangle, \text{ and } \forall j \in [1,k] : \langle \mathrm{def}_{e_j,q_{e_j,0}}, 0 \rangle, \\ & \quad \text{for } \delta \in \Delta,\ k = \mathrm{rk}(\delta),\ c_1, \ldots, c_k \in C, \\ \bot & \text{if } \neg \langle \mathrm{def}_c, 0 \rangle \end{cases}$$

is a rule in $R(\sigma)$. Note that the first case of the above rule really consists of many cases, one for each $\delta \in \Delta$ and $c_1, \ldots, c_{\mathrm{rk}(\delta)} \in C$, abbreviated to one. We have to make sure that only one case holds at any time, or, in other words, that δ, k, and c_j for all j are uniquely determined. Assume $\langle \mathrm{def}_c, u \rangle$ = true. This implies $t \models \phi$, and (u,c) is a node of t'. Clearly, δ (and thus k) is uniquely determined, because the node formulas are mutually exclusive. It follows from the udp that for every $j \in [1, \mathrm{rk}(\delta)]$ there is a unique (w_j, c_j) such that $(t,u,w_j) \models \chi_{j,c,c_j}(x,y)$, i.e., $(u, q_{e_j,0}) \twoheadrightarrow^*_{e_j} (w_j, q_f)$ for some $q_f \in F_{e_j}$. Hence, for every j, there is a unique c_j such that $\langle \mathrm{def}_{e_j,q_{e_j,0}}, u \rangle$ = true.

Note also that if (w_j, c_j) is chosen such that $(t,u,w_j) \models \chi_{j,c,c_j}(x,y)$, then

$$\mathrm{sub}_{t'}(u,c) = \delta(\mathrm{sub}_{t'}(w_1,c_1), \ldots, \mathrm{sub}_{t'}(w_k,c_k)),$$

which proves the correctness of this rule with respect to the intended meaning of the attributes.

The attributes $\langle \beta_{e,q}, u \rangle$ transport the data through the tree, according to the steps of the automaton A_e. They do so by checking if the configuration (u, q) is indeed on a walk of A_e, which is the case if $\langle \mathrm{def}_{e,q}, u \rangle$ is true. If so, the rule follows one step of the automaton. It checks if a step is possible, and if the resulting configuration is also on the walk of the automaton. This will be the case for exactly one step, because of the unique walk property.

In the next rule, please add the condition $\langle \mathrm{def}_{e,q}, 0 \rangle$ to every case but the first. For all $\sigma \in \Sigma$, $e = (j, c, c') \in \mathrm{rks}(\Delta) \times C \times C$, and $q \in Q_e$,

$$\langle \beta_{e,q}, 0 \rangle = \begin{cases} \bot & \text{if } \neg\langle \mathrm{def}_{e,q}, 0 \rangle, \\ \langle \beta_{c'}, 0 \rangle & \text{if } q \in F_e \text{ and } \langle \mathrm{def}_{c'}, 0 \rangle, \\ \langle \beta_{e,q'}, l \rangle & \text{if } \delta_e(q, \downarrow_l) = q' \text{ and } \langle \mathrm{def}_{e,q'}, l \rangle, \text{ for } l \in [1, \mathrm{rk}(\sigma)] \text{ and } q' \in Q, \\ \langle \beta_{e,q',*}, 0 \rangle & \text{if } \exists l : l = \langle \mathrm{chno}, 0 \rangle, \delta_e(q, \uparrow_l) = q', \text{ and } \langle \mathrm{def}_{e,q',*}, 0 \rangle, \text{ for } q' \in Q, \\ \langle \beta_{e,q'}, 0 \rangle & \text{if } \exists \psi \in D(\Pi_e) \cap \mathrm{MSOL}_1(\Sigma) \colon \delta_e(q, \psi) = q', \langle \psi, 0 \rangle, \text{ and } \langle \mathrm{def}_{e,q'}, 0 \rangle, \\ & \quad \text{for } q' \in Q \end{cases}$$

is a rule in $R(\sigma)$. Again, each of the last three cases consists of many cases. There always is exactly one true case: consider a node u. If $\langle \mathrm{def}_{e,q}, u \rangle$ is true, then $\exists! v, w : (v, q_{e,0}) \twoheadrightarrow_e^* (u, q) \twoheadrightarrow_e^* (w, q_f)$, for some $q_f \in F_e$. From the unique walk property we can conclude that either $(u, q) = (w, q_f)$ (which corresponds to the second case), or $\exists! (u', q'), d \in D(\Pi_e) : (v, q_{e,0}) \twoheadrightarrow_e^* (u, q) \overset{d}{\twoheadrightarrow}_e (u', q') \twoheadrightarrow_e^* (w, q_f)$ (which corresponds to exactly one of the other cases).

From this, the correctness of the rule with respect to the intended meaning of the attributes should also be clear: in the second case, $\langle \beta_{e,q}, u \rangle = \langle \beta_{c'}, u \rangle = \mathrm{sub}_{t'}(u, c')$, and in the other cases $\langle \beta_{e,q}, u \rangle = \langle \beta_{e,q'}, u' \rangle = \mathrm{sub}_{t'}(w, c')$.

For all $\sigma \in \Sigma$, $l \in [1, \mathrm{rk}(\sigma)]$, $e \in \mathrm{rks}(\Delta) \times C \times C$, and $q \in Q_e$,

$$\langle \beta_{e,q,*}, l \rangle = \begin{cases} \langle \beta_{e,q}, 0 \rangle & \text{if } \langle \mathrm{def}_{e,q,*}, l \rangle, \\ \bot & \text{otherwise} \end{cases}$$

is a rule in $R(\sigma)$.

For all $\sigma \in \Sigma$,

$$\langle \beta_{\mathrm{root}}, 0 \rangle = \begin{cases} \langle \beta_c, 0 \rangle & \text{if } \langle \mathrm{root}_c, 0 \rangle, \text{ for } c \in C, \\ \langle \beta_{\mathrm{root}}, l \rangle & \text{if } \langle \mathrm{def}_{\mathrm{root}}, l \rangle, \text{ for } l \in [1, \mathrm{rk}(\sigma)], \\ \bot & \text{otherwise} \end{cases}$$

is a rule in $R(\sigma)$.

The only root rules we have are for every $e \in \mathrm{rks}(\Delta) \times C \times C$, and $q \in Q_e$,

$$\langle \beta_{e,q,*}, 0 \rangle = \bot$$

is in R_{root}.

We do not give rules for the **flags**. All the flags are specified by an MSO formula. Although MSO formulas have not been specified for $\text{def}_{e,q}$, $\text{def}_{e,q,*}$, and def_{root}, it is clear that they exist. For $\text{def}_{e,q}$, we can check whether $(u,q) \twoheadrightarrow_e^* (u',q')$ by a regular path language. This is the language defined by the automaton $(Q_e, D(\Pi_e), \delta_e, q, \{q'\})$ (which is A_e with start state q and final state q'). Thus, we can check whether $(u,q) \twoheadrightarrow_e^* (u',q')$ by an MSO formula. A formula for $\text{def}_{e,q,*}$ is easily constructed when formulas for $\text{def}_{e,q}$ are known. Last, $\langle \text{def}_{\text{root}}, u\rangle = \text{true}$ iff $\exists c \in C, \exists v : \langle \text{root}_c, v\rangle = \text{true}$ and $\text{path}(u,v)$, a property that can easily be expressed by an MSO formula.

Note that the method of deriving attribute rules from MSO formulas presented in Section 2.3 introduces more than one attribute (all of finite semantic domain), and hence, strictly speaking, we have not listed all attributes of the attributed tree transducer.

The Semantic Conditions

For all $\sigma \in \Sigma$, we have the semantic condition

$$C(\sigma) = (\langle \text{dom}, 0\rangle = \text{true}).$$

The Meaning Attribute

As mentioned before, β_{root} is the meaning attribute:

$$\alpha_{\text{mean}} = \beta_{\text{root}}.$$

This concludes the description of the attributed tree transducer.

4.4.3 Example

Example 4.21. We will show a simple MSO tree transducer and the corresponding attributed tree transducer. Let the input alphabet be $\Sigma = \Sigma_0 \cup \Sigma_2$, with $\Sigma_0 = \{\#, *\}$, and $\Sigma_2 = \{\$\}$. The output alphabet is $\Delta = \Delta_0 \cup \Delta_1$, with $\Delta_1 = \Sigma_0 = \{\#, *\}$, and $\Delta_0 = \{\#', *'\}$.

We present an MSO transducer T, such that for any tree $t \in T_\Sigma$, $\mathcal{T}(t)$ is equal to the yield of t as a monadic tree (a string can be seen as a monadic tree, with its first character as root, the second character as child of the first, etcetera, up to the last character, which is the leaf of the tree). We have to beware, because all labels in the monadic tree that constitutes the yield have rank 1, except for the last one, which has rank 0. We will therefore use elements of Δ_1 for all of the labels, except the last one, for which we will use a corresponding label from Δ_0.

First we present a few more MSO formulas over Σ. The next formula checks if x is the rightmost leaf of a (binary) tree

$$\text{rml}(x) = \exists y : (\text{root}(y) \wedge \text{path}_2(y,x)).$$

We also have a formula to check if x is the leftmost leaf of a tree

$$\mathrm{lml}(x) = \exists y : (\mathrm{root}(y) \wedge \mathrm{path}_1(y, x)).$$

Let $T = (\{c\}, \phi, \{\psi_{\delta,c}\}_{\delta\in\Sigma_0} \cup \{\psi_{\delta',c}\}_{\delta\in\Sigma_0}, \{\chi_{1,c,c}\})$, be the MSO tree transducer from Σ to Δ, with

$$\begin{aligned}
\phi &= \mathrm{true}, \\
\psi_{\delta,c}(x) &= \mathrm{lab}_\delta(x) \wedge \neg\, \mathrm{rml}(x) \quad \text{for } \delta \in \Sigma_0, \\
\psi_{\delta',c}(x) &= \mathrm{lab}_\delta(x) \wedge \mathrm{rml}(x) \quad \text{for } \delta \in \Sigma_0, \\
\chi_{1,c,c}(x, y) &= \exists z\, (\exists z_l(\mathrm{edg}_1(z, z_l) \wedge \mathrm{path}_2(z_l, x)) \wedge \exists z_r(\mathrm{edg}_2(z, z_r) \wedge \mathrm{path}_1(z_r, y)))\,.
\end{aligned}$$

For leaves x and y, the last formula checks if y directly follows x in the left-to-right order of leaves. Note that this transducer satisfies all constraints of Remark 4.3, except the third. We will fix this by replacing the one edge formula by

$$\begin{aligned}
\chi_{1,c,c}(x, y) = \mathrm{leaf}(x) \wedge \\
\exists z\, (\exists z_l(\mathrm{edg}_1(z, z_l) \wedge \mathrm{path}_2(z_l, x)) \wedge \exists z_r(\mathrm{edg}_2(z, z_r) \wedge \mathrm{path}_1(z_r, y))) \wedge \mathrm{leaf}(y).
\end{aligned}$$

We will now construct the unconditional sfatt

$$G = (\Sigma, \{\mathbb{B}, \{0, 1, 2\}, T_{\Delta\cup\bot}\}, (S, I, W), R, \beta_{\mathrm{root}}),$$

that corresponds to this MSO tree transducer. First of all, the path language that corresponds to $\chi_{1,c,c}(x, y)$ is

$$\Pi = \mathrm{leaf}(x)(\uparrow_2)^* \uparrow_1\downarrow_2 (\downarrow_1)^* \mathrm{leaf}(x).$$

The transition graph of the corresponding automaton is depicted in Figure 4.5, in which the states are numbered in Roman numerals. The start state is I, and the single final state is V.

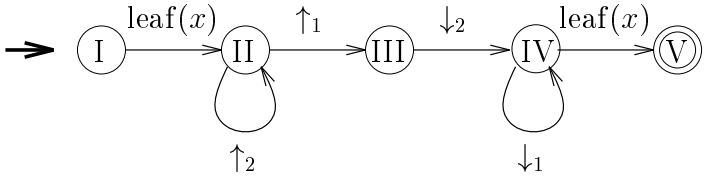

Figure 4.5: The automaton for $\chi_{1,c,c}(x, y)$

In the following, let $e = (1, c, c)$. For every node u of t, the values of the flags of G are

- $\langle \mathrm{def}_c, u\rangle$ = true iff u is a leaf.
- $\langle \mathrm{def}_{e,\mathrm{I}}, u\rangle$ = true iff u is a leaf, but not the rightmost one.

- $\langle \mathrm{def}_{e,\mathrm{II}}, u\rangle$ = true iff u is not on the path from the root to the rightmost leaf.
- $\langle \mathrm{def}_{e,\mathrm{III}}, u\rangle$ = true iff u is not a leaf.
- $\langle \mathrm{def}_{e,\mathrm{IV}}, u\rangle$ = true iff u is not on the path from the root to the leftmost leaf.
- $\langle \mathrm{def}_{e,\mathrm{V}}, u\rangle$ = true iff u is a leaf, but not the leftmost one.
- $\langle \mathrm{def}_{e,\mathrm{I},*}, u\rangle = \langle \mathrm{def}_{e,\mathrm{IV},*}, u\rangle = \langle \mathrm{def}_{e,\mathrm{V},*}, u\rangle$ = false.
- $\langle \mathrm{def}_{e,\mathrm{II},*}, u\rangle$ = true iff $u = u' \cdot 2$ for some u', and u is not on the path from the root to the rightmost leaf.
- $\langle \mathrm{def}_{e,\mathrm{III},*}, u\rangle$ = true iff $u = u' \cdot 1$ for some u'.
- $\langle \mathrm{def}_{\mathrm{root}}, u\rangle$ = true iff u is on the path from the root of t to its leftmost leaf.

Apart from these, we have the flags dom, ψ (for various formulas ψ), chno and root_c, which we will not discuss here.

As an example of how we derived the above values, we will show why $\langle \mathrm{def}_{e,\mathrm{III},*}, u\rangle$ = true iff $u = u' \cdot 1$ for some u', one of the more difficult cases. From the definition of $\mathrm{def}_{e,\mathrm{III},*}$, $\langle \mathrm{def}_{e,\mathrm{III},*}, u\rangle$ = true iff $\exists u' : u = u' \cdot l$ and $\exists v, w, q'$ such that

$$(v, q_{e,0}) \twoheadrightarrow_e^* (u' \cdot l, q') \overset{\uparrow_l}{\twoheadrightarrow}_e (u', \mathrm{III}) \twoheadrightarrow_e^* (w, q_f), \text{for some } q_f \in F_e.$$

If we look at the automaton, we see that $q' = \mathrm{II}$ and $\uparrow_l = \uparrow_1$, so this condition is equivalent to $\exists u' : u = u' \cdot 1$ and $\exists v, w$ such that

$$(v, q_{e,0}) \twoheadrightarrow_e^* (u' \cdot 1, \mathrm{II}) \overset{\uparrow_1}{\twoheadrightarrow}_e (u', \mathrm{III}) \twoheadrightarrow_e^* (w, q_f), \text{for some } q_f \in F_e.$$

Consider the automaton once again. Rewriting the above rule, by substituting regular expressions for parts of the automaton, we obtain the condition

$$\begin{gathered}\exists u' : u = u' \cdot 1,\\ \exists v : (v, u) \in R_t(\| \operatorname{leaf}(x)(\uparrow_2)^* \|), \text{ and}\\ \exists w : (u', w) \in R_t(\| \downarrow_2 (\downarrow_1)^* \operatorname{leaf}(x) \|).\end{gathered}$$

The second formula of the three is always true (any subtree rooted in a node u has a rightmost leaf v). The third one is equivalent to saying u' is not a leaf, which is implied by the first formula. So, $\langle \mathrm{def}_{e,\mathrm{III},*}, u\rangle$ = true iff u is a left child.

We now give the tree rules. For all $\sigma \in \Sigma$, the rule for β_c in $R(\sigma)$ is

$$\langle \beta_c, 0\rangle = \begin{cases} *(\langle \beta_{e,\mathrm{I}}, 0\rangle) & \text{if } \langle \mathrm{lab}_*(x) \wedge \neg\, \mathrm{rml}(x), 0\rangle,\\ \#(\langle \beta_{e,\mathrm{I}}, 0\rangle) & \text{if } \langle \mathrm{lab}_\#(x) \wedge \neg\, \mathrm{rml}(x), 0\rangle,\\ *' & \text{if } \langle \mathrm{lab}_*(x) \wedge \mathrm{rml}(x), 0\rangle,\\ \#' & \text{if } \langle \mathrm{lab}_\#(x) \wedge \mathrm{rml}(x), 0\rangle,\\ \bot & \text{if } \neg\langle \mathrm{def}_c, 0\rangle. \end{cases}$$

The following five rules are for the $\beta_{e,q}$. Still, e stands for $1, c, c$. For all $\sigma \in \Sigma$, $R(\sigma)$ contains the rule

$$\langle \beta_{e,\mathrm{I}}, 0 \rangle = \begin{cases} \bot & \text{if } \neg\langle \mathrm{def}_{e,\mathrm{I}}, 0 \rangle, \\ \langle \beta_{e,\mathrm{II}}, 0 \rangle & \text{if } \langle \mathrm{leaf}(x), 0 \rangle, \langle \mathrm{def}_{e,\mathrm{II}}, 0 \rangle, \text{ and } \langle \mathrm{def}_{e,\mathrm{I}}, 0 \rangle. \end{cases}$$

For all $\sigma \in \Sigma$, $R(\sigma)$ contains

$$\langle \beta_{e,\mathrm{II}}, 0 \rangle = \begin{cases} \bot & \text{if } \neg\langle \mathrm{def}_{e,\mathrm{II}}, 0 \rangle, \\ \langle \beta_{e,\mathrm{II},*}, 0 \rangle & \text{if } \langle \mathrm{chno}, 0 \rangle = 2, \langle \mathrm{def}_{e,\mathrm{II},*}, 0 \rangle, \text{ and } \langle \mathrm{def}_{e,\mathrm{II}}, 0 \rangle \\ \langle \beta_{e,\mathrm{III},*}, 0 \rangle & \text{if } \langle \mathrm{chno}, 0 \rangle = 1, \langle \mathrm{def}_{e,\mathrm{III},*}, 0 \rangle, \text{ and } \langle \mathrm{def}_{e,\mathrm{II}}, 0 \rangle. \end{cases}$$

The rules for state III are different for operators in Σ_0 and Σ_2. The sets $R(\#)$ and $R(*)$ contain the semantic rule

$$\langle \beta_{e,\mathrm{III}}, 0 \rangle = \bot,$$

and $R(\$)$ contains the semantic rule

$$\langle \beta_{e,\mathrm{III}}, 0 \rangle = \begin{cases} \bot & \text{if } \neg\langle \mathrm{def}_{e,\mathrm{III}}, 0 \rangle, \\ \langle \beta_{e,\mathrm{IV}}, 2 \rangle & \text{if } \langle \mathrm{def}_{e,\mathrm{IV}}, 2 \rangle \text{ and } \langle \mathrm{def}_{e,\mathrm{III}}, 0 \rangle. \end{cases}$$

Likewise for state IV. For all $\sigma \in \Sigma_0$, $R(\sigma)$ contains

$$\langle \beta_{e,\mathrm{IV}}, 0 \rangle = \begin{cases} \bot & \text{if } \neg\langle \mathrm{def}_{e,\mathrm{IV}}, 0 \rangle, \\ \langle \beta_{e,\mathrm{V}}, 0 \rangle & \text{if } \langle \mathrm{leaf}(x), 0 \rangle, \langle \mathrm{def}_{e,\mathrm{V}}, 0 \rangle, \text{ and } \langle \mathrm{def}_{e,\mathrm{IV}}, 0 \rangle, \end{cases}$$

and $R(\$)$ contains

$$\langle \beta_{e,\mathrm{IV}}, 0 \rangle = \begin{cases} \bot & \text{if } \neg\langle \mathrm{def}_{e,\mathrm{IV}}, 0 \rangle, \\ \langle \beta_{e,\mathrm{IV}}, 1 \rangle & \text{if } \langle \mathrm{def}_{e,\mathrm{IV}}, 1 \rangle \text{ and } \langle \mathrm{def}_{e,\mathrm{IV}}, 0 \rangle \\ \langle \beta_{e,\mathrm{V}}, 0 \rangle & \text{if } \langle \mathrm{leaf}(x), 0 \rangle, \langle \mathrm{def}_{e,\mathrm{V}}, 0 \rangle, \text{ and } \langle \mathrm{def}_{e,\mathrm{IV}}, 0 \rangle. \end{cases}$$

Last, for all $\sigma \in \Sigma$, $R(\sigma)$ contains

$$\langle \beta_{e,\mathrm{V}}, 0 \rangle = \begin{cases} \bot & \text{if } \neg\langle \mathrm{def}_{e,\mathrm{V}}, 0 \rangle, \\ \langle \beta_c, 0 \rangle & \text{if } \langle \mathrm{def}_c, 0 \rangle \text{ and } \langle \mathrm{def}_{e,\mathrm{V}}, 0 \rangle. \end{cases}$$

We have some more rules. First for the $\beta_{e,q,*}$, for $l \in \{1, 2\}$, $R(\$)$ contains

$$\langle \beta_{e,q,*}, l \rangle = \begin{cases} \langle \beta_{e,q}, 0 \rangle & \text{if } \langle \mathrm{def}_{e,q,*}, l \rangle, \\ \bot & \text{otherwise,} \end{cases}$$

and second for β_{root}, $R(\$)$ contains

$$\langle \beta_{\text{root}}, 0\rangle = \begin{cases} \langle \beta_c, 0\rangle & \text{if } \langle \text{lml}(x), 0\rangle, \\ \langle \beta_{\text{root}}, 1\rangle & \text{if } \langle \text{def}_{\text{root}}, 1\rangle, \\ \langle \beta_{\text{root}}, 2\rangle & \text{if } \langle \text{def}_{\text{root}}, 2\rangle, \\ \bot & \text{otherwise,} \end{cases}$$

and $R(\#)$ and $R()$ contain the semantic rule

$$\langle \beta_{\text{root}}, 0\rangle = \begin{cases} \langle \beta_c, 0\rangle & \text{if } \langle \text{lml}(x), 0\rangle, \\ \bot & \text{otherwise.} \end{cases}$$

The root rules are $\langle \beta_{e,q,*}, 0\rangle = \bot$ for every state q. The att is unconditional, as mentioned before, and the meaning attribute is β_{root}.

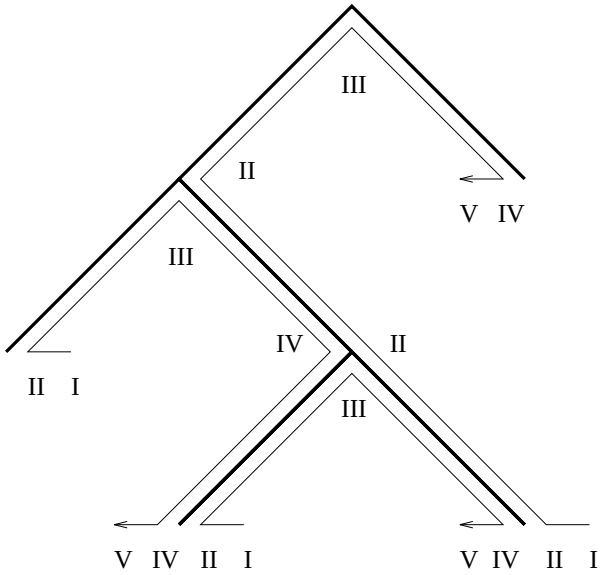

Figure 4.6: All walks of the automaton on tree t

To conclude the example, we consider how G acts on tree $t = \$\$\#\$*\#\#$, with output $\mathcal{T}(t) = \#*\#\#'$. Figure 4.6 gives all walks of the automaton on t.

Figure 4.7 gives the phase-2 (tree attributes) dependency graph. It shows the inherited attributes to the left of the node, and the synthesized attributes to the right. The attribute names have been abbreviated. Attribute β_{root} is denoted rt, β_c is denoted c, and the attributes $\beta_{e,q}$ and $\beta_{e,q,*}$ are both denoted q, for all q. The synthesized attributes $\beta_{e,\text{I},*}$, $\beta_{e,\text{IV},*}$, and $\beta_{e,\text{V},*}$ have been left out, because they are never used.

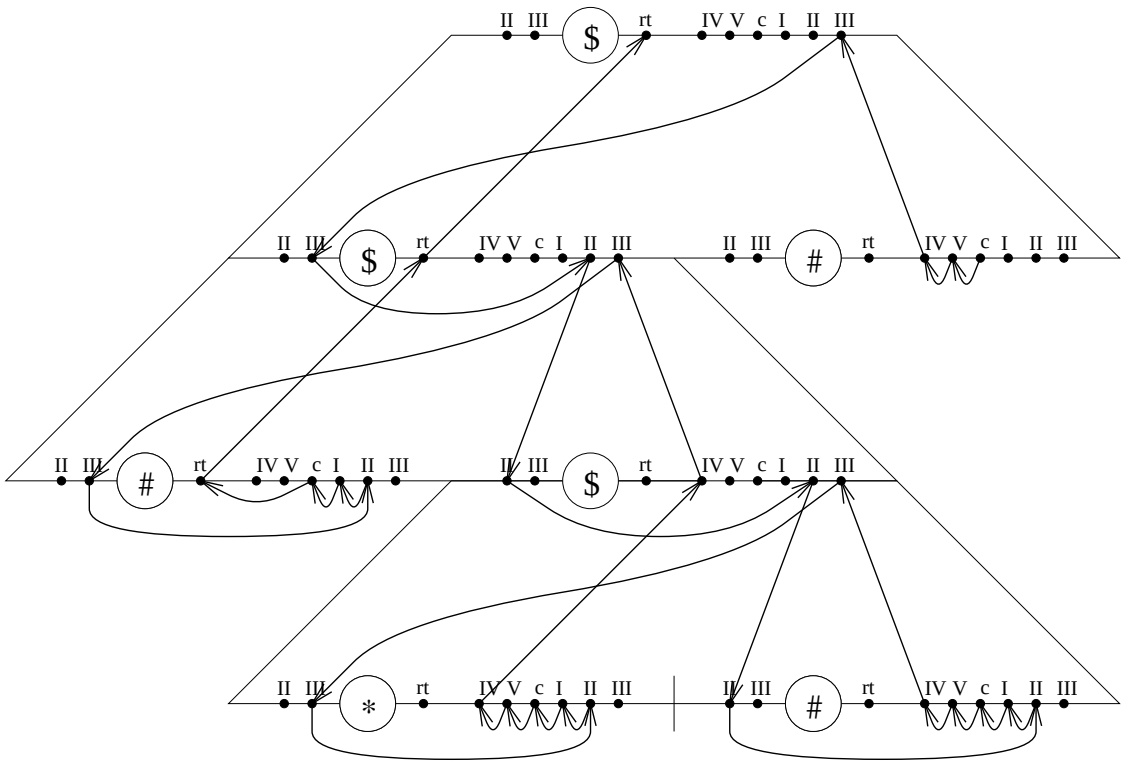

Figure 4.7: The phase-2 dependency graph of t

We can easily infer the values of the attributes. Let the nodes of t be numbered u_1 through u_7 in level order. The values of the β_c are

$$\begin{aligned}\langle \beta_c, u_3 \rangle &= \#', \\ \langle \beta_c, u_7 \rangle &= \#\#', \\ \langle \beta_c, u_6 \rangle &= *\#\#', \\ \langle \beta_c, u_4 \rangle &= \# * \#\#',\end{aligned}$$

(and the other occurrences of β_c have value $\perp$). The rules for all the other attributes merely copy values, so $\langle \alpha_{\text{mean}}, \text{root}(t) \rangle = \# * \#\#'$. □

4.4.4 Correctness

We have to prove that the transducer is non-circular with respect to (F, B), and that it satisfies the WSUR.

Lemma 4.22. *Attribute grammar G is weakly non-circular and phase-2* WSUR *with respect to (F, B).*

Proof. The first phase of the transduction is non-circular, because the flags are computed in the manner of Chapter 2. For the second phase, we want to prove that $\mathrm{WD}^2(t)$ is a forest without labels and with the edges reversed.

Again, keep in mind that the direction the automaton walks and the direction of the attribute dependencies are opposite, and that the terms unique source and destination pertain to the walk of the automaton.

First of all, if $t \notin \mathrm{dom}(\mathcal{T})$, all node and edge formulas are false and so all phase-2 semantic instructions are equivalent to $\langle \beta, u \rangle = \bot$, for the appropriate β and u. Hence there are no dependencies, and $\mathrm{WD}^2(t)$ is a discrete graph.

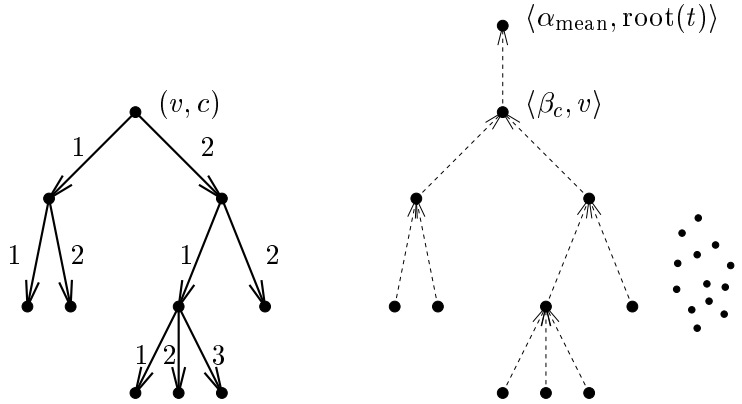

Figure 4.8: To the left: t', to the right: $\mathrm{WD}^2(t)$, where dashed lines symbolize paths

Suppose $t \in \mathrm{dom}(\mathcal{T})$, and let $t' = \mathcal{T}(t) \in T_\Delta$. We claim that $\mathrm{WD}^2(t)$ has the reversed-forest form suggested in Figure 4.8. We will make this claim more precise and then prove it. For the sake of readability, the edge relation $\rightarrow_{t'}$ will be denoted by a bold arrow $\mathbf{\rightarrow}$. The edge relation $\rightarrow_{\mathrm{WD}^2(t)}$ will be denoted by an ordinary arrow $\rightarrow$. We bijectively associate the nodes (v, c) of t' with the nodes $\langle \beta_c, v \rangle$ in $\mathrm{WD}^2(t)$ for which $\langle \mathrm{def}_c, v \rangle = \mathrm{true}$.

The phase-2 dependency graph $\mathrm{WD}^2(t)$ has nodes $\langle \beta_c, v \rangle$, nodes $\langle \beta_{\mathrm{root}}, v \rangle$, and nodes $\langle \beta_{e,q}, v \rangle$ and $\langle \beta_{e,q,*}, v \rangle$, for all $v \in V_t$. The nodes $\langle \beta_{e,q}, v \rangle$ and $\langle \beta_{e,q,*}, v \rangle$ are called *intermediate nodes* because of their role in the dependency graph. The edges of $\mathrm{WD}^2(t)$ are arranged as follows. There is a single path $\langle \beta_{c'}, v' \rangle \rightarrow^* \langle \beta_c, v \rangle$ through intermediate nodes only, if there is an edge $(v, c) \mathbf{\rightarrow} (v', c')$. The intermediate nodes all have exactly one incoming and exactly one outgoing edge. Let (v, c) be the root of t', and let $v = \mathrm{root}(t) \cdot i_1 \cdot \cdots \cdot i_n$. There is one more path $\langle \beta_c, \mathrm{root}(t) \cdot i_1 \cdot \cdots \cdot i_n \rangle \rightarrow \langle \beta_{\mathrm{root}}, \mathrm{root}(t) \cdot i_1 \cdot \cdots \cdot i_{n-1} \rangle \rightarrow \langle \beta_{\mathrm{root}}, \mathrm{root}(t) \cdot i_1 \cdot \cdots \cdot i_{n-2} \rangle \rightarrow \cdots \rightarrow \langle \beta_{\mathrm{root}}, \mathrm{root}(t) \rangle$. There are no other edges, and $\mathrm{WD}^2(t)$ may have isolated nodes (viz., all nodes that have def = false).

We prove the claim, that $\mathrm{WD}^2(t)$ has exactly the above mentioned nodes and edges, and thus is a reversed unlabelled forest, which implies that G is phase-2 WSUR and weakly non-circular with respect to (F, B).

First we discuss the path involving β_{root} attributes. A flag $\langle \mathrm{def}_{\mathrm{root}}, u \rangle$ is true iff u is on the unique path from the root of t to node v for which $\langle \mathrm{root}_c, v \rangle = \mathrm{true}$ (v and c

are unique). Looking at the rules for β_{root}, we can then see that the path involving β_{root} attributes has the form

$$\langle \beta_c, \text{root}(t) \cdot i_1 \cdot \cdots \cdot i_n \rangle \to \langle \beta_{\text{root}}, \text{root}(t) \cdot i_1 \cdot \cdots \cdot i_{n-1} \rangle \to \cdots \to \langle \beta_{\text{root}}, \text{root}(t) \rangle.$$

From here on we concentrate on the more difficult paths involving the attributes β_c, $\beta_{e,q}$, and $\beta_{e,q,*}$. The line of thought is the following. For every edge in t' there is a walk of an automaton. A walk of an automaton induces a path in the phase-2 weak-dependency graph, in which a configuration (v, q) of automaton A_e is associated with node $\langle \beta_{e,q}, v \rangle$ in $\text{WD}^2(t)$, and a step in the automaton corresponds to a dependency (or possibly two, involving an attribute $\beta_{e,q',*}$). Different edges in t' give different walks. Since different walks of one automaton are disjoint and therefore induce disjoint paths in the dependency graph (as far as the intermediary nodes are concerned), and two different automata induce disjoint paths as well, different edges in t induce disjoint paths in $\text{WD}^2(t)$.

First we will formally show that an edge in t' implies a path in the dependency graph, and we will give the precise form of such a path. Suppose there is an edge $(v_0, c) \xrightarrow{j} (v_n, c')$, and let $e = (j, c, c')$. By the uwp there are unique $q_1, \ldots, q_{n-1} \in Q_e$, $q_n \in F_e$, $v_1, \ldots, v_{n-1}$ and $d_1, \ldots, d_n$ such that

$$(v_0, q_{e,0}) \overset{d_1}{\twoheadrightarrow}_e (v_1, q_1) \overset{d_2}{\twoheadrightarrow}_e \ldots \overset{d_n}{\twoheadrightarrow}_e (v_n, q_n).$$

Let q_0 be $q_{e,0}$. The above walk of the automaton implies that

- $\forall i \in [0, n] : \langle \text{def}_{e,q_i}, v_i \rangle = \text{true}$, and
- $\forall i \in [1, n] :$ if $\exists l \in \text{rks}(\Sigma) : d_i = \uparrow_l$ then $\langle \text{def}_{e,q_i,*}, v_{i-1} \rangle = \text{true}$.

It also implies the following dependencies for all i.

- If $d_i \neq \uparrow_l$ for every l, then $\langle \beta_{e,q_{i-1}}, v_{i-1} \rangle \leftarrow \langle \beta_{e,q_i}, v_i \rangle$, and
- If $d_i = \uparrow_l$ for some l, then $\langle \beta_{e,q_{i-1}}, v_{i-1} \rangle \leftarrow \langle \beta_{e,q_i,*}, v_{i-1} \rangle \leftarrow \langle \beta_{e,q_i}, v_i \rangle$, and furthermore
- $\langle \beta_c, v_0 \rangle \leftarrow \langle \beta_{e,q_0}, v_0 \rangle$, and $\langle \beta_{e,q_n}, v_n \rangle \leftarrow \langle \beta_{c'}, v_n \rangle$.

See also Figure 4.9, in which such a path in the dependency graph is sketched, assuming that v_i is the parent of v_{i-1}.

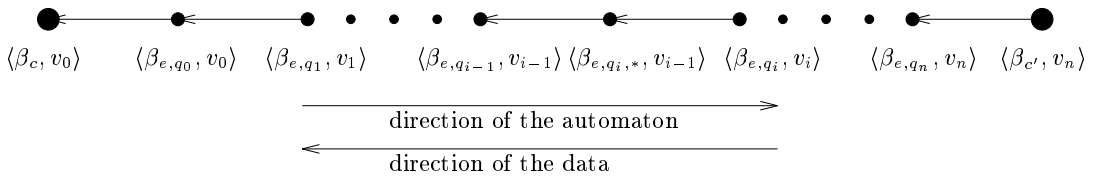

Figure 4.9: A path in the dependency graph

Now we shall show that any edge in the dependency graph is on a path of dependencies like the one described above. We first consider three cases for three different combinations of the starred and unstarred form of the $\beta_{e,q}$ attributes.

1. If, for some e, q, q', u, u', $\langle \beta_{e,q}, u\rangle \leftarrow \langle \beta_{e,q'}, u'\rangle$,
then $\langle \mathrm{def}_{e,q}, u\rangle = \text{true}$, $\langle \mathrm{def}_{e,q'}, u'\rangle = \text{true}$ and $\exists d \in D(\Pi_e)$ with $d \neq \uparrow_l$ for every l, such that $\delta_e(q,d) = q'$, and $(u,u') \in R_t(d)$.
Hence, $\exists v, w : (v, q_{e,0}) \twoheadrightarrow_e^* (u,q) \twoheadrightarrow_e^* (w, q_f)$ for some $q_f \in F_e$,
$\exists v', w' : (v', q_{e,0}) \twoheadrightarrow_e^* (u', q') \twoheadrightarrow_e^* (w', q_f)$ for some $q_f \in F_e$, and
$(u,q) \overset{d}{\twoheadrightarrow}_e (u', q')$. This implies

$$(v, q_{e,0}) \twoheadrightarrow_e^* (u,q) \overset{d}{\twoheadrightarrow}_e (u', q') \twoheadrightarrow_e^* (w', q_f), \text{ for some } q_f \in F_e,$$

and thus $(t, v, w') \models \chi_e(x,y)$, and so (Remark 4.3),

$$(v, c) \overset{j}{\rightarrow} (w', c').$$

There is only one walk of automaton A_e from (v,c) to (w',c'), which, as described above, corresponds to a path in the dependency graph, with $\langle \beta_{e,q}, u\rangle \leftarrow \langle \beta_{e,q'}, u'\rangle$ on it (because $(u,q) \overset{d}{\twoheadrightarrow}_e (u', q')$ is a step in that walk, and $d \neq \uparrow_l$ for every l).

2. If $\langle \beta_{e,q',*}, u'\rangle \leftarrow \langle \beta_{e,q}, u\rangle$,
then $q = q'$, $u' = u \cdot l$ for some l, and $\langle \mathrm{def}_{e,q,*}, u \cdot l\rangle = \text{true}$.
Hence, $\exists v, w, \exists q'' : (v, q_{e,0}) \twoheadrightarrow_e^* (u \cdot l, q'') \overset{\uparrow_l}{\twoheadrightarrow}_e (u,q) \twoheadrightarrow_e^* (w, q_f)$ for some $q_f \in F_e$, and thus

$$(v, c) \overset{j}{\rightarrow} (w', c').$$

So there is a path in the dependency graph corresponding to this walk, that has $\langle \beta_{e,q',*}, u'\rangle \leftarrow \langle \beta_{e,q}, u\rangle$ on it.

3. If $\langle \beta_{e,q}, u\rangle \leftarrow \langle \beta_{e,q',*}, u'\rangle$,
then $u = u' = u'' \cdot l$ for some u'' and l, $\langle \mathrm{def}_{e,q}, u\rangle = \text{true}$, $\langle \mathrm{def}_{e,q',*}, u\rangle = \text{true}$ and $\delta_e(q, \uparrow_l) = q'$.
Hence, $\exists v, w : (v, q_{e,0}) \twoheadrightarrow_e^* (u'' \cdot l, q) \twoheadrightarrow_e^* (w, q_f)$ for some $q_f \in F_e$,
$\exists v', w', \exists q'' : (v', q_{e,0}) \twoheadrightarrow_e^* (u'' \cdot l, q'') \overset{\uparrow_l}{\twoheadrightarrow}_e (u'', q') \twoheadrightarrow_e^* (w', q_f)$ for some $q_f \in F_e$, and
$(u'' \cdot l, q) \overset{\uparrow_l}{\twoheadrightarrow}_e (u'', q')$. This implies

$$(v, q_{e,0}) \twoheadrightarrow_e^* (u'' \cdot l, q) \overset{\uparrow_l}{\twoheadrightarrow}_e (u'', q') \twoheadrightarrow_e^* (w', q_f), \text{ for some } q_f \in F_e,$$

and thus

$$(v, c) \overset{j}{\rightarrow} (w', c').$$

And through the same reasoning as above, $\langle \beta_{e,q}, u\rangle \leftarrow \langle \beta_{e,q',*}, u'\rangle$ is on the path in the dependency graph that corresponds to this walk.

Two cases for two different combinations of the β_c and $\beta_{e,q}$ attributes are still to be considered.

1. If $\langle \beta_c, u \rangle \leftarrow \langle \beta_{e,q}, u' \rangle$,
   then $u = u'$, $q = q_{e,0}$, $\langle \mathrm{def}_c, u \rangle = \mathrm{true}$, and $\langle \mathrm{def}_{e,q_{e,0}}, u \rangle = \mathrm{true}$. From the latter it follows that $\exists w : (u, q_{e,0}) \twoheadrightarrow_e^* (w, q_f)$ for some $q_f \in F_e$, and thus

   $$(u, c) \xrightarrow{j} (w', c').$$

   So $\langle \beta_c, u \rangle \leftarrow \langle \beta_{e,q}, u' \rangle$ is on the path in the dependency graph that corresponds to this walk.

2. If $\langle \beta_{e,q}, u \rangle \leftarrow \langle \beta_{c'}, u' \rangle$,
   then $u = u'$, $q \in F_e$, $\langle \mathrm{def}_{e,q}, u \rangle = \mathrm{true}$, and $\langle \mathrm{def}_{c'}, u \rangle = \mathrm{true}$. From the former it follows that $\exists v : (v, q_{e,0}) \twoheadrightarrow_e^* (u, q)$, and thus

   $$(v, c) \xrightarrow{j} (u, c').$$

   So $\langle \beta_{e,q}, u \rangle \leftarrow \langle \beta_{c'}, u' \rangle$ is on the path in the dependency graph that corresponds to this walk.

Now we have established that all dependencies are on a path $\langle \beta_{c'}, v' \rangle \rightarrow^* \langle \beta_c, v \rangle$ through intermediate nodes only. Last we show that such a path has no branches, i.e., the intermediate nodes do not have more than one incoming and one outgoing node. It is clear that no such node has more than one incoming edge, since such an attribute always depends on one other attribute only.

Suppose a node has two outgoing edges. We prove a contradiction. Two outgoing edges would imply that there are an e, a $q_n \in F_e$ and two possible paths of the automaton:

$$(v_0, q_{e,0}) \twoheadrightarrow_e \ldots \twoheadrightarrow_e (v_i, q_i) \twoheadrightarrow_e (v_{i+1}, q_{i+1}) \twoheadrightarrow_e \ldots \twoheadrightarrow_e (v_n, q_n),$$

and

$$(v'_0, q_{e,0}) \twoheadrightarrow_e \ldots \twoheadrightarrow_e (v'_i, q'_i) \twoheadrightarrow_e (v_{i+1}, q_{i+1}) \twoheadrightarrow_e \ldots \twoheadrightarrow_e (v_n, q_n),$$

with $(v'_i, q'_i) \neq (v_i, q_i)$. But by the unique walk property, (v_{i+1}, q_{i+1}) lies on a unique walk.

We conclude that the dependency graph has the aforementioned forest shape, and thus that G is WSUR and weakly non-circular with respect to (F, B). □

We have proven that the attribute grammar is weakly non-circular and phase-2 WSUR with respect to (F, B). It follows, by Lemma 1.41, that every tree t has a unique decoration $\mathrm{dec}_{G,t}$. Because the intended meaning of the attributes (as given in Subsection 4.4.2) satisfies the semantic rules, it is equal to that unique decoration. Hence,

$$\mathcal{G}(t) = \mathrm{dec}_{G,t}(\langle \alpha_{\mathrm{mean}}, \mathrm{root}(t) \rangle) = \mathrm{dec}_{G,t}(\langle \beta_{\mathrm{root}}, \mathrm{root}(t) \rangle) = t' = \mathcal{T}(t).$$

Thus, we conclude the following lemma.

Lemma 4.23. *For every* MSO *tree transducer T there is a* WSUR *attributed tree transducer with flags G with $\mathcal{T} = \mathcal{G}$, or*

$$\text{MSOTT} \subseteq \text{SFATT} .$$

From Lemmata 4.17 and 4.23, we conclude the following equality.

Theorem 4.24.

$$\text{MSOTT} = \text{SFATT} .$$

4.5 Complexity

An MSO definable tree transduction can be evaluated in linear time. We can show this in two different ways, either from the definition of an MSO transducer, or from the definition of the equivalent attributed tree transducer.

Theorem 4.25. *For every* MSO *tree transducer T and tree t, $\mathcal{T}(t)$ can be constructed in time linear in the size of t.*

Proof. First we will prove the result from the definition of MSO tree transducers. The domain formula can be checked in linear time, by using a tree automaton (Lemma 1.45). The node formulas can be evaluated in linear time too, by Theorem 2.20. Last, since the edge formulas define partial functions (cf. the discussion after Lemma 4.19), by Theorem 3.22, the edge formulas can be checked in linear time as well.

The result is also easily obtained by looking at attributed tree transducers. For every node of the tree we have a constant number of attributes, so the number of attributes is linear in the size of the tree. For every attribute we have to evaluate one rule, and every given rule can be computed in constant time, since they either concern a finite semantic domain, or tree substitution. Hence, the attribute grammar can be evaluated in linear time in the size of the tree. □

Bibliography

[Boc76] Gregor V. Bochmann. Semantic evaluation from left to right. *Communications of the ACM*, 19:55–62, 1976.

[CM79] Laurian M. Chirica and David F. Martin. An order-algebraic definition of Knuthian semantics. *Mathematical Systems Theory*, 13:1–27, 1979.

[Coh81] P.M. Cohn. *Universal Algebra, revised edition.* Reidel, 1981.

[Cou90] Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 5, pages 193–242. Elsevier, 1990.

[Cou91] Bruno Courcelle. The monadic second order logic of graphs V: On closing the gap between definability and recognizability. *Theoretical Computer Science*, 80:153–202, 1991.

[Cou92] Bruno Courcelle. Monadic second-order definable graph transductions. In *CAAP*, volume 581 of *Lecture Notes in Computer Science*, pages 124–144. Springer, 1992.

[Cou94] Bruno Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126:53–75, 1994.

[DJL88] Pierre Deransart, Martin Jourdan, and Bernard Lorho. *Attribute Grammars, Definitions, Systems and Bibliography*, volume 323 of *Lecture Notes in Computer Science.* Springer, 1988.

[Don70] John Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.

[Eng81] Joost Engelfriet. Tree transducers and syntax-directed semantics. Technical Report Memorandum nr. 363, Twente University of Technology, The Netherlands, 1981.

[Eng84] Joost Engelfriet. Attribute grammars: Attribute evaluation methods. In B. Lorho, editor, *Methods and Tools for Compiler Construction*, pages 103–138. Cambridge University Press, 1984.

[Eng89] Joost Engelfriet. Context-free NCE graph grammars. In *Fundamentals of Computation Theory*, volume 380 of *Lecture Notes in Computer Science*, pages 148–161. Springer, 1989.

[Eng91] Joost Engelfriet. A characterization of context free NCE graph languages by monadic second order logic on trees. In *Graph Grammars and their Application to Computer Science*, volume 532 of *Lecture Notes Computer Science*, pages 311–327. Springer, 1991.

[Fül81] Zoltán Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.

[FV95] Zoltán Fülöp and Sándor Vágvölgyi. Attributed tree transducers cannot induce all deterministic bottom-up tree transformations. *Information and Computation*, 116:231–240, 1995.

[Gie88] R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Informatica*, 25:355–423, 1988.

[Gin75] S. Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages*, volume 2 of *Fundamental Studies in Computer Science*. North-Holland, 1975.

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[Knu68] Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2:127–145, 1968. Correction: *Mathematical Systems Theory*, 5: 95–96, 1971.

[KS93] Nils Klarlund and Michael L. Schwartzbach. Graph types. In *Proceedings of the 20th Conference on Principles of Programming Languages*, pages 196–205, 1993.

[Oos89] V. van Oostrom. Graafgrammatica's en 2^e orde logica. Master's thesis, University of Leiden, 1989. In dutch.

[TW68] J.W. Thatcher and J.B Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2:57–81, 1968.

[Wei87] Klaus Weihrauch. *Computability*, volume 9 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1987.