polynomial simplification order that has domain A and employs only polynomials whose degree is bounded by k. For this purpose, the coefficients of the polynomials are treated as existentially quantified variables in the decision procedure. The main problem with this approach is, of course, the high complexity of Tarski's decision procedure for the first-order theory of real numbers.

5.4.2 Recursive path orders

The main idea underlying recursive path orders is that two terms are compared by first comparing their root symbols, and then recursively comparing the collections of their immediate subterms. These collections can be seen as unordered multisets (which yields the multiset path order), or as ordered tuples (which yields the lexicographic path order), or one can employ a combination of both (which yields the recursive path order with status). In the following, we consider the lexicographic path order in more detail.

Definition 5.4.12 Let Σ be a *finite signature* and $>$ be a strict order on Σ. The **lexicographic path order** $>_{lpo}$ on $T(\Sigma, V)$ induced by $>$ is defined as follows: $s >_{lpo} t$ iff

(LPO1) $t \in \mathcal{V}ar(s)$ and $s \neq t$, or
(LPO2) $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$, and

- **(LPO2a)** there exists $i, 1 \leq i \leq m$, with $s_i \geq_{lpo} t$, or
- **(LPO2b)** $f > g$ and $s >_{lpo} t_j$ for all $j, 1 \leq j \leq n$, or
- **(LPO2c)** $f = g$, $s >_{lpo} t_j$ for all $j, 1 \leq j \leq n$, and there exists $i, 1 \leq i \leq m$, such that $s_1 = t_1, \ldots, s_{i-1} = t_{i-1}$ and $s_i >_{lpo} t_i$.

This definition is recursive since in (LPO2a), (LPO2b), and (LPO2c) it refers to the relation $>_{lpo}$ to be defined. Nevertheless, $>_{lpo}$ is well-defined since the definition of $s >_{lpo} t$ only refers to the relation $>_{lpo}$ applied to pairs of terms that are smaller than the pair s, t. In (LPO2a), $\geq_{lpo}$ stands for the reflexive closure of $>_{lpo}$ (and not for the lexicographic path order induced by $\geq$). In (LPO2c), the collections of immediate subterms are compared with respect to $>_{lex}^{n}$, the n-fold lexicographic product of $>_{lpo}$ with itself, which explains the name *lexicographic* path order. Before showing that $>_{lpo}$ is a simplification order, we consider some examples.

Example 5.4.13 Let $\Sigma = \{f, i, e\}$, where f is binary, i is unary, and e is a constant, and assume that $i > f > e$.

1. $f(x, e) >_{lpo} x$ by (LPO1).
2. $i(e) >_{lpo} e$ by (LPO2a) since we have $e \geq_{lpo} e$.

3. $i(f(x,y)) >_{lpo} f(i(y),i(x))$ by (LPO2b) since $i > f$ and, by (LPO2c), $i(f(x,y)) >_{lpo} i(y)$ and $i(f(x,y)) >_{lpo} i(x)$. The preconditions for case (LPO2c) are satisfied since we have $i(f(x,y)) >_{lpo} y$, $i(f(x,y)) >_{lpo} x$ and $f(x,y) >_{lpo} y$, $f(x,y) >_{lpo} x$ by (LPO1).
4. $f(f(x,y),z) >_{lpo} f(x,f(y,z))$ by (LPO2c) with $i = 1$:
   - $f(f(x,y),z) >_{lpo} x$: this holds because of (LPO1).
   - $f(f(x,y),z) >_{lpo} f(y,z)$: again, we have (LPO2c) with $i = 1$:
     - $f(f(x,y),z) >_{lpo} y$ and $f(f(x,y),z) >_{lpo} z$: by (LPO1).
     - $f(x,y) >_{lpo} y$: by (LPO1).
   - $f(x,y) >_{lpo} x$: by (LPO1).

Theorem 5.4.14 *For any strict order $>$ on Σ, the induced lexicographic path order $>_{lpo}$ is a simplification order on $T(\Sigma, V)$.*

Proof (1) Before we can show transitivity, we need an auxiliary result, which we prove by induction on $|s| + |t|$:

$$s >_{lpo} t \text{ implies } \mathcal{V}ar(s) \supseteq \mathcal{V}ar(t).$$

In (LPO1), $t = x$ is a variable that occurs in s and thus $\mathcal{V}ar(t) = \{x\} \subseteq \mathcal{V}ar(s)$. In (LPO2a), $s_i \geq_{lpo} t$ yields $\mathcal{V}ar(s) \supseteq \mathcal{V}ar(s_i) \supseteq \mathcal{V}ar(t)$ by induction. In (LPO2b) and (LPO2c), $s >_{lpo} t_j$ for all $j, 1 \leq j \leq n$, yields $\mathcal{V}ar(s) \supseteq \mathcal{V}ar(t_j)$ for all $j, 1 \leq j \leq n$, by induction, and thus $\mathcal{V}ar(s) \supseteq \bigcup_{j=1}^{n} \mathcal{V}ar(t_j) = \mathcal{V}ar(t)$.

(2) To show *transitivity*, we assume that $r >_{lpo} s$ and $s >_{lpo} t$. We prove $r >_{lpo} t$ by induction on $|r| + |s| + |t|$. Obviously, $r >_{lpo} s$ implies that r is not a variable, and $s >_{lpo} t$ implies that s is not a variable.

First, assume that $t = x$ is a variable. To obtain $r >_{lpo} t$ by (LPO1), it is sufficient to show that the variable x occurs in r. Because of $s >_{lpo} t = x$, we know that x occurs in s. In addition, as we have shown in (1), $r >_{lpo} s$ implies $\mathcal{V}ar(r) \supseteq \mathcal{V}ar(s)$, and thus $x \in \mathcal{V}ar(r)$.

Now, assume $r = f(r_1, \ldots, r_l)$, $s = g(s_1, \ldots, s_m)$, and $t = h(t_1, \ldots, t_n)$. First, we consider the two cases where one of the inequalities is due to (LPO2a):

- $r >_{lpo} s$ is an instance of (LPO2a), i.e. there exists $i, 1 \leq i \leq l$, such that $r_i \geq_{lpo} s$. By induction, we obtain $r_i \geq_{lpo} t$, and thus $r >_{lpo} t$ holds by (LPO2a).
- $s >_{lpo} t$ is an instance of (LPO2a), and $r >_{lpo} s$ is an instance of (LPO2b) or (LPO2c). We have $r >_{lpo} s_j$ for all $j, 1 \leq j \leq m$, and $s_i \geq_{lpo} t$ for some $i, 1 \leq i \leq m$. By induction, $r >_{lpo} s_i \geq_{lpo} t$ yields $r >_{lpo} t$.

Thus, we may assume that both inequalities are due to (LPO2b) or (LPO2c). This implies $f \geq h$ and $s >_{lpo} t_j$ for all $j, 1 \leq j \leq l$. By induction, $r >_{lpo} s >_{lpo} t_j$ yields $r >_{lpo} t_j$ for all $j, 1 \leq j \leq l$. If $f > h$, this is sufficient to obtain $r >_{lpo} t$. Otherwise, we have $f = g = h$, and thus both inequalities are due to (LPO2c). Now, $r >_{lpo} t$ can be shown as in the proof of transitivity of the lexicographic product (where the induction hypothesis yields transitivity for the subterms).

(3) Because we already know that $>_{lpo}$ is transitive, the *subterm property* is proved if we succeed in showing that $f(\ldots, s, \ldots) >_{lpo} s$ for all function symbols f and terms s. If $s = x$ is a variable, then $f(\ldots, s, \ldots) >_{lpo} s$ is an instance of (LPO1). Otherwise, it is an instance of (LPO2a) since $s \geq_{lpo} s$.

(4) *Closure under substitutions*, i.e. $s >_{lpo} t$ implies $\sigma(s) >_{lpo} \sigma(t)$ for all terms s, t and all substitutions σ, is shown by induction on $|s| + |t|$. For (LPO1), $t = x$ is a variable occurring in s and $s \neq t$. Thus, $\sigma(t)$ is a strict subterm of $\sigma(s)$, and we obtain $\sigma(s) >_{lpo} \sigma(t)$ as a consequence of the subterm property. In (LPO2a), $s_i \geq_{lpo} t$ implies $\sigma(s_i) \geq_{lpo} \sigma(t)$ by induction. Similar induction arguments apply in the remaining two cases.

(5) To show *compatibility with Σ-operations*, we assume that $s >_{lpo} t$, $f \in \Sigma^{(n)}$, and $s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n \in T(\Sigma, V)$. Then

$$f(s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots, s_n) >_{lpo} f(s_1, \ldots, s_{i-1}, t, s_{i+1}, \ldots, s_n)$$

is obtained as an instance of (LPO2c): the subterm property yields

$$f(s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots, s_n) >_{lpo} s_j$$

for all $j \in \{1, \ldots, i-1, i+1, \ldots, n\}$, and $f(s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots, s_n) >_{lpo} s$. Together with the assumption $s >_{lpo} t$, this last inequality implies $f(s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots, s_n) >_{lpo} t$ by transitivity. Finally, $s_1 = s_1, \ldots, s_{i-1} = s_{i-1}$ and $s >_{lpo} t$ are obvious.

(6) In order to show *irreflexivity* of $>_{lpo}$, we assume that there exists a term s such that $s >_{lpo} s$, and try to refute this assumption by induction on the size of s. If $s = x$ is a variable, then the only possible case is (LPO1). However, the condition "$s \neq s$" necessary for this case to apply is obviously not satisfied.

Thus, assume that $s = f(s_1, \ldots, s_n)$. Obviously, (LPO1) and (LPO2b) cannot apply. For (LPO2c), there must exist an $i, 1 \geq i \geq n$, such that $s_i >_{lpo} s_i$. By induction, we know that this is not possible. For (LPO2a), we have on the one hand an index $i, 1 \leq i \leq n$, such that $s_i \geq_{lpo} s$. On the other hand, the subterm property yields $s >_{lpo} s_i$. Transitivity implies $s_i >_{lpo} s_i$, which contradicts our induction hypothesis. □

One advantage of lexicographic path orders over polynomial orders is that they do not impose a doubly-exponential bound on the length of reduction sequences. In fact, they can even be used to show termination of term rewriting systems with reduction sequences whose length cannot be bounded by a primitive recursive function:

Example 5.4.15 Termination of the term rewriting system R_{Ack} introduced in Example 5.3.11 can be shown using the lexicographic path order that is induced by $a > s$.

A further nice feature of lexicographic path orders is the fact that it is decidable whether termination of a given finite term rewriting system can be shown with the help of such an order.

Proposition 5.4.16 *Let Σ be a finite signature, $s, t \in T(\Sigma, V)$, and R a finite term rewriting system over $T(\Sigma, V)$.*

1. *For a given lexicographic path order, $s >_{lpo} t$ can be decided in time polynomial in the size of s, t.*
2. *The question of whether termination of R can be shown using some lexicographic path order on $T(\Sigma, V)$ is an NP-complete problem.*

The first statement of the proposition is an easy consequence of the definition of lexicographic path orders (Exercise 5.25). An NP-algorithm for the problem addressed in the second statement of the proposition simply guesses an order $>$ on Σ, and then uses the polynomial algorithm of the first statement to check whether this guess was correct. NP-hardness of the problem for the multiset path order is shown in [149]. This proof can easily be adapted to the lexicographic path order.

The applicability of this approach for showing termination can be increased by allowing for different ways of comparing the collections of subterms in case (LPO2c) of Definition 5.4.12. Instead of comparing the tuples $(s_1, \ldots, s_m)$ and $(t_1, \ldots, t_m)$ lexicographically from left to right, one can also define an order where they are always compared lexicographically from right to left. More generally, one can associate each function symbol with a fixed permutation of its arguments, and then compare the tuples of immediate subterms lexicographically along this permutation. To obtain the **multiset path order** $>_{mpo}$ induced by a strict order $>$ on Σ, one considers the multisets $\{s_1, \ldots, s_m\}$ and $\{t_1, \ldots, t_m\}$, and compares them with the multiset order induced by $>_{mpo}$. The fact that this yields a well-defined simplification order can be shown by a proof that is similar to the one for $>_{lpo}$ [72]. In the **recursive path order with status**, these different approaches are combined: each function symbol is equipped with a status that determines whether the collections of subterms are compared by the multiset order, or lexicographically with respect to a permutation associated with the function symbol. The following is an example of a rewrite system where only the combination of the multiset and the lexicographic status yields a recursive path order that can show its termination:

Example 5.4.17 Let s be a unary and $+, *$ be binary function symbols. We consider the term rewriting system R that consists of the rules

$$\begin{aligned}(x+y)+z &\rightarrow x+(y+z),\\ x*s(y) &\rightarrow x+(y*x).\end{aligned}$$

The first rule can only be oriented in this direction with a recursive path order that assigns lexicographic status (from left to right) to $+$. In order to orient the second rule from left to right, we need $* > +$. In addition, $x*s(y)$ must be larger than $y*x$, which can only be achieved by assigning multiset status to $*$. Note that the additional rule

$$x+s(y) \rightarrow s(y+x)$$

would require $+ > s$ and multiset status for $+$, which implies that all three rules together cannot be shown to be terminating with a recursive path order with status.

5.4.3 Recursive path orders in ML

Building on the type of terms (see Section 4.7) and the lexicographic order *lex* already defined, the definition of $>_{lpo}$ is easily turned into ML code:

```
(* (string * string -> order) -> term * term -> order *)
fun lpo ord (s,t) = case (s,t) of
      (s, V x) => if s = t then EQ
                  else if occurs x s then GR (*LPO1*) else NGE
    | (V _, T _) => NGE
    | (T(f,ss), T(g,ts)) => (*LPO2*)
        if forall (fn si => lpo ord (si,t) = NGE) ss
        then case ord(f,g) of
               GR => if forall (fn ti => lpo ord (s,ti) = GR) ts
                     then GR (*LPO2b*) else NGE
             | EQ => if forall (fn ti => lpo ord (s,ti) = GR) ts
                     then lex (lpo ord) (ss,ts) (*LPO2c*)
                     else NGE
             | NGE => NGE
        else GR (*LPO2a*);
```

If *ord* implements the order $>$ on the function symbols, then *lpo ord* implements $>_{lpo}$.

Although we have already indicated the places in the code which correspond to particular clauses in the definition of the lexicographic path order, the following comments should answer any remaining questions.

- The branches returning *NGE* are the result of analysing in which cases neither $s >_{lpo} t$ nor $s = t$ holds. For example, if s is a variable and t is not, then $s >_{lpo} t$ cannot hold because (LPO1) requires t to be a variable and (LPO2) requires s not to be a variable. This justifies the line `(V _,T _) => NGE`.
- Case (LPO2a) is slightly disguised because we have replaced the test $\exists 1 \le i \le m.\ s_i \ge_{lpo} t$ by $\forall 1 \le i \le m.\ s_i \not\ge_{lpo} t$.
- Case (LPO2c) is simplified by appealing to the functional *lex* for comparing the subterms lexicographically. Note that the definition of $>_{lpo}$ avoids the use of $(>_{lpo})_{lex}$ because $(>_{lpo})_{lex}$ is only well-defined if $>_{lpo}$ is a strict order, something we do not yet know while defining $>_{lpo}$.

The precise implementation of the parameter *ord* is not germane to the subject of this book. Suffice it to say that the most straightforward representation is a list of pairs `(f,g)` meaning $f > g$. There should also be a function to compute the transitive closure of such a list, which obviates the need to supply `(f,h)` in addition to `(f,g)` and `(g,h)`.

The recursive path order with status is a generalization of the lexicographic and multiset path orders. At the implementation level it means that *rpo* is an abstraction of *lpo* w.r.t. the functional *lex*:

```
(* rpo: (string -> (term * term -> order) -> term list * term list -> order)
        -> (string * string -> order) -> term * term -> order *)
fun rpo stat ord (s,t) = case (s,t) of
      (s, V x) => if s = t then EQ
                  else if occurs x s then GR else NGE
    | (V _, T _) => NGE
    | (T(f,ss), T(g,ts)) =>
        if forall (fn si => rpo stat ord (si,t) = NGE) ss
        then case ord(f,g) of
               GR => if forall (fn ti => rpo stat ord (s,ti) = GR) ts
                     then GR else NGE
             | EQ => if forall (fn ti => rpo stat ord (s,ti) = GR) ts
                     then (stat f) (rpo stat ord) (ss,ts)
                     else NGE
             | NGE => NGE
        else GR;
```

The additional parameter *stat* maps each function symbol to the appropriate subterm order, e.g. as in `(fn "f" => lex | "g" => mul, ...)`.

Note that *rpo* is one of the rare examples of a natural third-order function: one of its parameters is a function which itself takes a function as an argument.