# Some Decision Problems for ML Refinement Types

Christian Skalka

Department of Philosophy
Carnegie Mellon University

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Logic and Computation*

January 9, 1998

**Abstract**

The ML `datasort` declaration, described in [Fre94] and [DP97], is an extension to Standard ML allowing the declaration of "refinement" types, which correspond to subsets of pre-existing ML datatypes. Refinement types provide the programmer with a more precise method of expressing data structure, can prevent the occurence of certain run-time exceptions, and assist in the expression and maintenance of some representation invariants. ML refinement types correspond to the regular term and regular tree languages described in [DZ92] and [GS84], except that refinement types are polymorphic. While algorithms have been defined which determine the emptiness, intersection and subset problems for regular terms, the corresponding problems for polymorphic regular terms have not been studied. This thesis formally describes a refinement type language, and defines algorithms relevant to these decision problems, which are proven correct. Along with theoretical interest, these algorithms have practical applications for refinement type checking, as described in [DP97].

# Contents

# 1 Introduction

Standard ML [MTH90] is a functional programming language with polymorphic types, and provides a system of type inference that ensures well-typedness of every expression at compile-time. As is often pointed out, this strongly-typed nature of the language eliminates many kinds of errors which the programmer might otherwise encounter at run-time, which is a principal appeal of the system. However, there is currently no method for defining subtypes of standard ML datatypes, and this lack of precision can sometimes lead to undesirable results.

For example, given the following declaration of lists:

```
datatype 'a list = NIL | CONS of 'a * 'a list
```

one might wish to define some function which will only applied to non-empty lists; for example, the following function which returns the last cons cell in a list:

```
fun lastcons(CONS(h,NIL)) = CONS(h,NIL)
|   lastcons(CONS(_,t)) = lastcons t
```

Of course, the compiler will generate a warning when typing this function, as the matching provided by the function is not exhaustive for lists. However, the programmer might know that the `NIL` case is unreachable due to some representation invariant, in which case the warning would be spurious. Also, because the type of the function in the ML type system is `'a list` $\rightarrow$ `'a list`, it is not a type error to make the application `lastcons NIL`, though this will always cause a run-time exception, as well as being a violation of the presumed representation invariant for `lastcons`. Thus, benefits can clearly be provided by some subtyping system for ML, including increased precision, elimination of certain errors and unecessary warnings, and, in some cases, an enhanced ability to ensure representation invariants through type specifications.

The `datasort` declaration provided by the refinement type system for ML, described by Pfenning and Freeman in [FP91] and more recently by Davies and Pfenning in [DP97], introduces just this sort of subtyping mechanism into the language. With appropriate `datasort` declarations, non-empty lists, as well as empty and singleton lists, can be defined:

```
datasort 'a nonempty = CONS of 'a * 'a list
datasort 'a empty = NIL
datasort 'a singleton = CONS of 'a * 'a empty
```

Then, with the addition of the refinement type checking algorithm given in [DP97], the function `lastcons` can be given the refinement type `'a nonempty` $\rightarrow$ `'a singleton` by the programmer, which will be verified at compile-time; spurious warnings will be eliminated, and applications such as `lastcons NIL` will cause a compile-time error, rather than a run-time exception, alerting the programmer that a representation invariant has been violated.

The refinement type checking algorithm performs "abstract interpretation over a programmer-specified finite lattice of refinements of each ML type" [FP91]. Thus, the algorithm requires that the subtype relationship be decided for the set of refinements of a given ML type. Also, the abstract interpretation described in [DP97] requires that the principal type of datatype constructors be specified, and the principal type is expressed as an intersection type in this scheme. For example, with `'a list`, `'a singleton` and `'a nonempty` as defined above, the constructor `cons` has the type:

$$\begin{array}{l}\texttt{'a} * \texttt{'a list} \to \texttt{'a list} \wedge \\ \texttt{'a} * \texttt{'a list} \to \texttt{'a nonempty} \wedge \\ \texttt{'a} * \texttt{'a empty} \to \texttt{'a singleton}\end{array}$$

Thus, the intersection of refinement types must also be decidable for the purposes of this algorithm. Clearly then, certain decision problems are central to the refinement type inference algorithm, and the implementation of the refinement type system in general. This thesis is concerned with these decision problems, and by giving an algorithm for the intersection of refinement types in particular, it is intended to provide practical as well as theoretical groundwork for the implementation of the type inference system.

The refinement type language described here has much in common with the regular tree grammars described in [GS84] and the regular term grammars described in [DZ92]. In both of these texts, algorithms were given for the emptiness, intersection and subset problems for these regular terms. However, these languages were not polymorphic; indeed, Dart and Zobel in [DZ92] remark, "It is not clear how such types might be manipulated or compared, and this problem has not been well investigated". As shall be shown, certain restrictions on the language allow refinement types to be manipulated and compared in a manner similar to that described in [GS84] and also [DZ92], the polymorphic nature of refinement types notwithstanding. The solution of the emptiness and intersection problems for polymorphic regular term grammars is the principal mathematical contribution of this thesis.

# 2 Characterization of the language

The type and refinement type language described and used in this paper is an analogue of the intended language for ML refinement types. Symbols will be used which are obviously not in the ASCII character set; however, the general structure is identical to the intended object language. Examples will be given which illustrate this correspondance.

## 2.1 Type and refinement type grammars

**Definition 2.1** The grammar for types is:

$$\mathit{mlty} ::= \mathit{mltyvar} \mid \mathit{mlty} * \mathit{mlty} \mid (\mathit{mltyseq})\mathit{mltycons}$$

$$\mathit{mltyseq} ::= \circ \mid \mathit{mlty},\mathit{mltyseq}$$

The symbol $\circ$ in this grammar denotes the empty *mltyseq*. The symbols used to denote types (*mlty*) in general will be lower-cased roman letters; $\mathrm{t}_1, \mathrm{k}_\mathrm{i}$, etc. Type constructors (*mltycons*) will also be denoted by lower-cased roman letters, and will be distinguished from types by context. The symbol used to denote *mltyvar* terms will be $\alpha$. Variable-free types are called *ground types*.

**Example 2.2** The following are types: $(\alpha)\mathrm{c}$, $(\alpha_1, \alpha_2)\mathrm{b}$, and $(\alpha)\mathrm{c} * (\circ)\mathrm{d}$. It is also allowable for the notation $(\circ)$ to be omitted. Thus, the type $(\alpha)\mathrm{c} * (\circ)\mathrm{d}$ may be written as $(\alpha)\mathrm{c} * \mathrm{d}$.

It will be noted immediately that a syntax for function types are not included in this grammar. While it is the intent of the refinement type system to ascribe principal sorts to functions, as in the case of `lastcons` described in the introduction, function types will be disallowed in refinement type declarations to ensure decidability of intersection and subtyping. That is, the exclusion of function types, in part, ensures that the language remains in the realm of regular terms. Since we will be only concerned with declared refinement types in this paper, a consideration of function types and function refinement types is therefore unnecessary.

**Definition 2.3** The grammar for refinement types, which are also called *sorts*, is:

$$sort ::= sortvar \mid sort * sort \mid (sortseq)sortcons$$

$$sortseq ::= \ \circ \mid sort{,}sortseq$$

The symbol $\circ$ in this grammar denotes the empty *sortseq*. The symbols used to denote sort terms in general will be lower-cased italic letters, e.g. $s$, $r$, etc. Sort variables (*sortvar*) will be denoted by $\alpha$ preceded by an italic $r$, e.g. $r\alpha$. This notation is meant to emphasize the refinement type relation for sort variables, described below. Sort constructor (*sortcons*) symbols will be denoted by lower cased Greek letters. Sort variable-free sorts are called *ground sorts*.

**Example 2.4** The following are sorts: $(r\alpha)\varrho$, $(\circ)\beta$, and $(\circ)\beta * (r\alpha_1, r\alpha_2)\zeta$. The same abbreviation used in the empty case of *mltyseq* is also employed in the similar case of *sortseq*, so that $(\circ)\beta$ may be written as $\beta$.

**Definition 2.5** Types and sorts which are built up using the $*$ construct are called product types, or product sorts. Types and sorts which are built from the application of a sort or type constructor to a *sortseq* or *mtyseq* will be called constructed sorts, or constructed types. Thus, $(\alpha)\mathrm{c}$ is a constructed type; $\beta * (r\alpha_1, r\alpha_2)\zeta$ is a product sort. Ground constructed types and ground constructed sorts are called type or sort *symbols*.

## 2.2 Refinement type deduction rules

A central idea in the theory of refinement types is that every sort *refines* some type; the rules for this relation are given below. Since only refinements of the same type are compared or intersected in sort checking, the refinement type relation is a significant factor in the decision problems for sorts explored in this paper; and it will become clear that the refinement type relation goes a long way towards ensuring decidability of these questions. If $s$ refines t, we write $s :: \mathrm{t}$.

**Definition 2.6** An *environment*, denoted by $\mathcal{S}$, is a set of sort and type declarations (which are described below). If a sort $s$ is defined in an environment $\mathcal{S}$ (the rules for which are also descibed below), then we write $\mathcal{S} \vdash s$.

**Definition 2.7** The following rules define the valid deductions for refinement types in an environment $\mathcal{S}$:

*(a)* sort variables: $$\frac{}{\mathcal{S} \vdash r\alpha :: \alpha}$$

*(b)* product sorts: $$\frac{\forall 0 < i \leq n \,.\, \mathcal{S} \vdash r_i :: \mathrm{t}_i}{\mathcal{S} \vdash r_1 * \cdots * r_n :: \mathrm{t}_1 * \cdots * \mathrm{t}_n}$$

*(c)* constructed sorts: $$\frac{\mathcal{S} \vdash \zeta :: \mathrm{c} \qquad \forall 0 < i \leq n \,.\, \mathcal{S} \vdash r_i :: \mathrm{t_i}}{\mathcal{S} \vdash (r_1, \ldots, r_n)\zeta :: (\mathrm{t_1}, \ldots, \mathrm{t_n})\mathrm{c}}$$

The relation $\zeta$ :: c in the third rule, signifying that the sort constructor $\zeta$ refines the type constructor c, is deduced from the declarations of $\zeta$ and c; the rules for this deduction, as well as the syntax and well-formedness rules for declarations, are given in the next section.

It is important to note here is that there is only one refinement of a type variable $\alpha$, which is the corresponding sort variable $r\alpha$. Again, since we are only concerned with comparing refinements of the same type in refinement type inference, if $s_1$ and $s_2$ are to be compared, it must be the case that $s$ :: t and $s_2$ :: t, so that if one of $s_1$ or $s_2$ is a sort variable $r\alpha$, for example, then both are the same sort variable $r\alpha$. Clearly, this constraint alleviates many of the decidability problems normally associated with polymorphic types. Refer to [FP91] for a more complete consideration of these issues.

## 2.3 Type and sort declarations

Type and sort declarations, as in the case of Standard ML, specify constants and value constructors which are associated with the type or sort being declared. Recursion and mutual recursion is allowed, although with constraints as described below.

**Definition 2.8** The grammar for type declarations is:

*tydecl* ::= (*mltyvarseq*)*mltycons* = *tyclauses* <<u>and</u> *tydecl*>

*tyclauses* ::= $\circ$ | *constant* | *valcons*(*mlty*) | (*tyclauses* <u>|</u> *tyclauses*)

Underlined symbols are part of the object language. Syntax enclosed in the symbols "<" and ">" is optional. The symbol $\circ$ denotes the empty *tyclause*. An *mltyvarseq* is analagous to *mltyseq*, except that only unique type variables are allowed; this is a list of the variables which may appear in the corresponding *tyclauses* of a type's declaration. However, this list may contain variables which do not appear in the RHS of the declaration. Constants will be denoted by upper-cased roman letters, as will value constructors (*valcons*). *tyclauses* in the RHS of a declaration will be said to define the type constructor on the LHS, and will be denoted by upper-cased roman letters. As in the case of standard ML, each constant and value constructor in the list of clauses defining a type must be unique.

**Example 2.9** The following are type declarations:

$$(\alpha)\mathrm{c} = N \mid C(\alpha * (\alpha)\mathrm{c})$$

$$(\alpha)\mathrm{y} = L \mid V((\alpha)\mathrm{y}) * \alpha * (\alpha)\mathrm{y})$$

$$\mathrm{n} = Z \mid S(\mathrm{n})$$

These declarations are analogues of the following ML datatype declarations:

```
datatype 'a list = NIL | CONS of 'a * 'a list
datatype 'a tree = LEAF | NODE of 'a tree * 'a * 'a tree
datatype nat = ZERO | SUC of nat
```

**Definition 2.10** The grammar for sort declarations is:

*sortdecl* ::= (*sortvarseq*)*sortcons* = *sortclauses* <<u>and</u> *sortdecl*>

*sortclauses* ::= $\circ$ | *constant* | *valcons*(*sort*) | (*sortclauses* | *sortclauses*)

The symbol $\circ$ denotes the empty sortclause. Syntax enclosed in the symbols "<" and ">" is optional. Constants (*sconstant*) and sort value constructors (*svalcons*) will be denoted by upper-cased italic letters. *sortclauses* will be denoted by upper cased greek letters. If $\Gamma$ defines a sort constructor $\zeta$ in environment $\mathcal{S}$ then we write $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_n)\zeta^n = \Gamma$. This latter notation will be used extensively. Also, while a clause list is not a set, it will be convenient to use set notation when specifying that a constant or a value constructor is in a given clause list, e.g., $F(g) \in \Gamma$. And unlike type declarations, multiple occurences of the same value constructor symbol in a sort declaration is allowable.

Certain constraints are imposed on the appearance of constructors in recursive and mutually recursive declarations, which are defined as follows:

**Definition 2.11** A sort constructor $\zeta_2$ is inductively defined as being *parametrically reachable* by a sort constructor $\zeta_1$ iff $\zeta_1 = \zeta_2$, or $\zeta_2$ appears as a subterm of a sort in the RHS of the $\zeta_1$'s definition, or a sort constructor $\zeta_3$ appears in the RHS of $\zeta_1$'s definition, and $\zeta_2$ is parametrically reachable by $\zeta_3$. Hence, the relation is transitive.

**Definition 2.12** A sort constructor $\zeta$ is *recursive* iff $\zeta$ appears in the RHS of its own definition. Two sort constructors $\zeta_1$ and $\zeta_2$ are *mutually recursive* iff $\zeta_2$ is parametrically reachable by $\zeta_1$ and $\zeta_1$ is parametrically reachable by $\zeta_2$. The declarations, or definitions, of $\zeta_1$ and $\zeta_2$ are mutually recursive if $\zeta_1$ and $\zeta_2$ are.

Mutually recursive definitions must be declared simultaneously using the "and" keyword, as in the case of Standard ML. Also, the following constraint is imposed on sort declarations, which has a significant impact on the decidability issues this thesis is concerned with, as will be demontrated in subsequent proofs:

**Definition 2.13** If a sort constructor $\zeta$ appears in the RHS of its own declaration, or a mutually recursive declaration, then it must be preceded by the same *sortvarseq* which precedes it in the LHS of its declaration, which is called the *parameter sequence* of $\zeta$.

Well-formedness of sort declarations also entails that a sort declaration be a refinement of a type declaration. For a sort declaration to be a refinement of a type declaration, the *sortvarseq* preceding the sort constructor on the RHS of the declaration must refine the corresponding *tyvarseq* in the type declaration, and the *sortclauses* on the LHS must refine the corresponding tyclauses, meaning that each constant and value constructor symbol in a sort declaration is also in the type declaration it refines, and the arguments to any sort value constructor symbol must be a refinement of the type given as an argument for the corresponding type value constructor.

**Definition 2.14** More formally, the deduction rules for declaration refinements are as follows:

$$\textit{(a) constant}: \quad \frac{}{\mathcal{S} \vdash C :: C} \qquad \textit{(b) valcons}: \quad \frac{\mathcal{S} \vdash s :: \mathrm{t}}{\mathcal{S} \vdash F(s) :: F(\mathrm{t})}$$

$$\textit{(c) sortclauses}: \quad \frac{\forall K_s \in \Gamma \;\; \exists K_{\mathrm{t}} \in \mathrm{G} \text{ such that } \mathcal{S} \vdash K_s :: K_{\mathrm{t}}}{\mathcal{S} \vdash \Gamma :: \mathrm{G}}$$

$$\textit{(d) sortdecl}: \quad \frac{\mathcal{S} \vdash \Gamma :: \mathrm{G} \qquad \mathcal{S} \vdash \; r\alpha_i :: \alpha_i \; \forall 0 < i \leq n}{((r\alpha_1, \ldots, r\alpha_n)\zeta = \Gamma\,) :: ((\alpha_1, \ldots, \alpha_n)\mathrm{c} = \mathrm{G}\,)}$$

Since in general we identify declarations with the constructors they define, the shorthand $\zeta :: \mathrm{c}$ is used to signify the refinement relation $((r\alpha_1, \ldots, r\alpha_n)\gamma = \Gamma\,) :: ((\alpha_1, \ldots, \alpha_n)\mathrm{c} = \mathrm{G}\,)$, as in definition 2.7, rule *(c)*. Note also that a declaration with an empty list of clauses can be inferred to refine a number of different types; because of this uncertainty, the "empty declaration" is disallowed in the intended ML refinement type syntax (see [DP97]). However, it is allowed in the present language, since in this thesis context will always make clear the intended type for refinement.

**Example 2.15** The following are valid sort declarations:

$$(r\alpha)\nu = C(r\alpha * (r\alpha)\nu)$$

$$(r\alpha)\kappa = N$$

$$\varepsilon = Z \mid S(o) \text{ and } o = S(\varepsilon)$$

These are analogues of the following ML datasort declarations:

```
datasort 'a nonempty = CONS of 'a * 'a list
datasort 'a empty = NIL
datasort even = ZERO | SUC of odd
   and odd  = SUC of even
```

It is straightforward to infer that in an environment $\mathcal{S}$ containing these declarations and the declarations in example 2.9, $\nu$ and $\kappa$ refine c as given in example 2.9, and $\varepsilon$ and $o$ refine n (just as `nonempty` and `empty` are intended to refine `list`, and `even` and `odd` are intended to refine `nat`).

**Example 2.16** In an environment $\mathcal{S}$ containing the declarations in examples 2.9 and 2.15, the following refinement type relations hold, by definition 2.7:

$$\mathcal{S} \vdash (r\alpha * (\varepsilon)\nu) :: (\alpha * (\mathrm{n})\mathrm{c})$$

$$\mathcal{S} \vdash (r\alpha * (o)\nu)\phi :: (\alpha * (\mathrm{n})\mathrm{c})\mathrm{c}$$

$$\mathcal{S} \vdash o * r\alpha * o :: \mathrm{n} * \alpha * \mathrm{n}$$

**Example 2.17** In an environment $\mathcal{S}$ containing the declarations in example 2.9 (and possibly others), the following sort declarations are not valid:

$$(r\alpha)\mu = C(r\alpha * (\varepsilon)\nu)$$

$$(\varepsilon)\pi = N \mid C(\varepsilon * (\varepsilon)\pi)$$

$$(r\alpha)\phi = R(((r\alpha)\phi)\phi)$$

The first is invalid by definition 2.14, because $(\varepsilon)\nu$ does not refine $(\alpha)\mathrm{c}$. The second is also invalid by definition 2.14 because the sorts in the parameter sequence preceding $\nu$ on the LHS of its definition must be sort variables. The third is invalid by definition 2.13, because a different parameter sequence precedes $\phi$ on the LHS and RHS of its definition.

For upcoming proofs, it is useful to require that all environments contain at least one non-empty sort declaration; thus, the following definition is made:

**Definition 2.18** Declarations for the type u and the sort $\varrho$, which are $\mathrm{u} = G$ and $\varrho = G$, exist in every environment $\mathcal{S}$ by assumption. Thus, for all $\mathcal{S}$, $\mathcal{S} \vdash \varrho :: \mathrm{u}$.

## 3 Correspondance of sorts to values

Types and sorts in this language are intended to correspond to sets of values. The question of emptiness or intersection with regard to sorts is really one regarding the set of values they represent. Therefore, a thorough characterization of the correspondance of sorts to sets of values must be given.

**Definition 3.1** Sort *terms* are inductively defined as follows:

1. a ground sort is a sort term
2. a constant is a sort term
3. if $F$ is a value constructor, and $s$ is a sort, then $F(s)$ is a sort term
4. if $\forall 0 < i \leq n \,.\, s_i$ is a sort term, then $(s_1, \ldots, s_n)$ is a sort term.

Sort *values*, denoted by boldfaced lowercased letters **a**, **b**, **c** etc., are sort symbol free sort terms.

With parametric sorts, there is evidently the question of what set of values to associate with a type variable, or in this case, a sort variable. Does this correspond to "all possible values"? In the refinement type theory, the "meaning" of a parametric sort with regard to sets of values is given in terms of all possible instantiations of that parametric sort. For example, the question of whether a given parametric sort $s$ is empty, is a question of whether every instantiation of $s$ corresponds to an empty set of values. A more precise definition of "instantiation", in terms of substitutions, is given below.

## 3.1 Type and sort substitutions

**Definition 3.2** The grammars for type and sort substitutions are:

$$\textit{tysubst} ::= \circ \mid \{\textit{tysubst}, \textit{mlty}/\textit{mltyvar}\}$$

$$\textit{sortsubst} ::= \circ \mid \{\textit{sortsubst}, \textit{sort}/\textit{sortvar}\}$$

The symbol $\circ$ in this grammar denotes the empty substitution, which may also be denoted by $\{\}$. Type substitutions will be denoted by $\Xi$, while sort substitutions will be denoted by $\Theta$. For each element $s/r\alpha$ in a sort substitution $\Theta$, $s$ is called the *binding* for $r\alpha$, likewise for type substitutions. It is allowable that a binding be a parametric sort.

**Definition 3.3** The sort obtained by the application of a sort substitution $\Theta = \{s_1/r\alpha_1, \ldots, s_n/r\alpha_j\}$ to a sort $r$ is defined as follows (and similarly for types):

- $\Theta(r\alpha_i) = \begin{cases} s_i \text{ if } s_i/r\alpha_i \text{ in } \Theta \\ r\alpha_i \text{ otherwise} \end{cases}$
- $\Theta(s_1 * \cdots * s_n) = \Theta(s_1) * \cdots * \Theta(s_n)$
- $\Theta((s_1, \ldots, s_n)\zeta) = (\Theta(s_1), \ldots, \Theta(s_n))\zeta$

Note that for any sort symbol $s$, $\Theta(s) = s$. If $\Theta(r)$ is a ground sort, then $\Theta(r)$ is said to be a *ground instance* of $r$, and $\Theta$ is a *ground substitution* for $r$.

**Example 3.4** Let $\Theta = \{\varrho/r\alpha_1, (\beta)\gamma/r\alpha_2\}$, and let $s = r\alpha_1 * (r\alpha_2)\nu$. Then $\Theta(s) = \varrho * ((\beta)\gamma)\nu$.

**Definition 3.5** The *application set* of a substitution $\Theta$ and a set of $n$-ary sort tuples $\Psi$, written $\Theta(\Psi)$, is defined as $\{(\Theta(s_1), \ldots, \Theta(s_n)) \mid (s_1, \ldots, s_n) \in \Psi\}$. This includes the unary case.

**Definition 3.6** The refinement type relation on sort and type substitutions is defined as follows:

$$\frac{\forall 0 < i \leq n \;.\; \mathcal{S} \vdash s_i :: \mathrm{t}_i}{\mathcal{S} \vdash \{s_1/r\alpha_1, \ldots, s_n/r\alpha_n\} \;::\; \{\mathrm{t}_1/\alpha_1, \ldots, \mathrm{t}_n/\alpha_n\}}$$

**Proposition 3.7** If $\mathcal{S} \vdash \Theta :: \Xi$ and $\mathcal{S} \vdash s :: \mathrm{t}$ then $\mathcal{S} \vdash \Theta(s) :: \Xi(\mathrm{t})$.

**Proof** By structural induction on $s$.

*case(i)* If $s$ is a sort variable $r\alpha$, then $\mathrm{t} = \alpha$ by definition 2.7. If there exists $s'/r\alpha$ in $\Theta$, then by definition 3.6 there exists $\mathrm{t}'/\alpha$ in $\Xi$ such that $\mathcal{S} \vdash s' :: \mathrm{t}'$, so $\Theta(r) = s'$ and $\Xi(\mathrm{t}) = \mathrm{t}'$, thus $\mathcal{S} \vdash \Theta(s) :: \Xi(\mathrm{t})$. If there is no binding in $\Theta$ for $r\alpha$, then there is no binding for $\alpha$ in $\Xi$, so $\Theta(s) = r\alpha$ and $\Xi(\mathrm{t}) = \alpha$ by definition 3.3, thus $\mathcal{S} \vdash \Theta(r) :: \Xi(\mathrm{t})$.

*case(ii)* If $s$ is a product sort $s_1 * \cdots * s_j$, then t is a product type $\mathrm{t}_1 * \cdots * \mathrm{t}_j$. By the induction hypothesis, $\forall 0 < i \leq j \;.\; \mathcal{S} \vdash \Theta(s_i) :: \Xi(\mathrm{t}_i)$, and by definition 3.3, $\Theta(s_1 * \cdots * s_j) = \Theta(s_1) * \cdots * \Theta(s_j)$, likewise $\Xi(\mathrm{t}_1 * \cdots * \mathrm{t}_j) = \Xi(\mathrm{t}_1) * \cdots * \Xi(\mathrm{t}_j)$, therefore this case holds.

*case (iii)* If $s$ is a constructed sort $(s_1, \ldots, s_j)\phi$, then $\mathrm{t}$ is a constructed type $(\mathrm{t}_1, \ldots, \mathrm{t}_j)\mathrm{c}$ by definition 3.6. By the induction hypothesis, $\forall 0 < i \leq j \;.\; \mathcal{S} \vdash \Theta(s_i) :: \Xi(\mathrm{t}_i)$, and by definition 3.3, $\Theta((s_1, \ldots, s_j)\phi = (\Theta(s_1), \ldots, \Theta(s_j))\phi$, likewise $\Xi((\mathrm{t}_1, \ldots, \mathrm{t}_j)\mathrm{c}) = (\Xi(\mathrm{t}_1), \ldots, \Xi(\mathrm{t}_j))\mathrm{c}$, therefore this case holds. $\Box$

**Definition 3.8** The *identity substitution*, denoted $\Theta_\epsilon$, is either the empty substitution, or the substitution in which every binding is identical to the variable it binds. Thus, for all sorts $s$, $\Theta_\epsilon(s) = s$.

**Definition 3.9** For all sorts $s$, the *$\varrho$-substitution*, denoted $\Theta_\varrho$, is a ground substitution for $s$, and each binding in $\Theta_\varrho$ is $\varrho$.

**Definition 3.10** The *composition* of substitutions $\Theta_1 = \left\{s_1/r\alpha_1^1, \ldots, s_j/r\alpha_j^1\right\}$ and $\Theta_2 = \{r_1/r\alpha_1^2, \ldots, r_k/r\alpha_k^2\}$, written $\Theta_1 \circ \Theta_2$, is the substitution obtained from the sequence:

$$\left\{\Theta_1(r_1)/r\alpha_1^2, \ldots, \Theta_1(r_k)/r\alpha_k^2, s_1/r\alpha_1^1, \ldots, s_j/r\alpha_j^1\right\}$$

By deleting any element $s_i/r\alpha_i^1$ for which $\exists 0 < n \leq k \;.\; r\alpha_i^1 = r\alpha_n^2$.

**Example 3.11** Let $\Theta_1 = \{\varrho/r\alpha_1, (\beta)\gamma/r\alpha_2\}$, and let $\Theta_2 = \{r\alpha_1 * (r\alpha_2)\nu/r\alpha_1\}$. Then $\Theta_1 \circ \Theta_2 = \{\varrho * ((\beta)\gamma)\nu/r\alpha_1, (\beta)\gamma/r\alpha_2\}$.

**Proposition 3.12** For substitutions $\Theta_1 = \left\{s_1/r\alpha_1^1, \ldots, s_j/r\alpha_j^1\right\}$ and $\Theta_2 = \{r_1/r\alpha_1^2, \ldots, r_k/r\alpha_k^2\}$, the following statements hold:

1. For all sorts $s$, $\Theta_1 \circ \Theta_2(s) = \Theta_1(\Theta_2(s))$
2. For all sorts $s$, $\Theta_\epsilon \circ \Theta_1(s) = \Theta_1 \circ \Theta_\epsilon(s) = \Theta_1(s)$

**Proof** 1. Let $s$ be some sort. Then this follows from a proof by structural induction on $s$:

*case (i)* If $s$ is a sort variable $r\alpha_i$, then if $\neg\exists 0 < n \leq k \;.\; r\alpha_i = r\alpha_i^2$, then $\Theta_1(\Theta_2(s)) = \Theta_1(s) = \Theta_1 \circ \Theta_2(s)$ by definitions 3.10 and 3.3. If $\exists 0 < n \leq k \;.\; r\alpha_i = r\alpha_i^2$, then $\Theta_1(\Theta_2(s)) = \Theta_1(r_i) = \Theta_1 \circ \Theta_2(s)$ by definitions 3.12 and 3.3.

*case (ii)* If $s$ is a product sort $s_1' * \cdots * s_m'$, then $\Theta_1(\Theta_2(s_1' * \cdots * s_m')) = \Theta_1(\Theta_2(s_1') * \cdots * \Theta_2(s_m')) = \Theta_1(\Theta_2(s_1')) * \cdots * \Theta_1(\Theta_2(s_m'))$, and likewise $\Theta_1 \circ \Theta_2(s_1' * \cdots * s_m') = \Theta_1 \circ \Theta_2(s_1') * \cdots * \Theta_1 \circ \Theta_2(s_m')$. Therefore by the induction hypothesis, $\forall 0 < i \leq m \;.\; \Theta_1 \circ \Theta_2(s_i') = \Theta_1(\Theta_2(s_i'))$, so clearly the statement holds in this case.

*case (iii)* If $s$ is a constructed sort $(s_1', \ldots, s_m')\zeta$, then this case follows by the induction hypothesis similarly as in case *(ii)*.

2. By the preceding, $\Theta_1 \circ \Theta_\epsilon(s) = \Theta_1(\Theta_\epsilon(s))$, and $\Theta_\epsilon \circ \Theta_1(s) = \Theta_\epsilon(\Theta_1(s))$. But $\Theta_\epsilon(s) = s$ and $\Theta_\epsilon(\Theta_1(s)) = \Theta_1(s)$ by definition 3.8, therefore this statement holds. $\Box$

The following proposition will be very useful for later proofs.

**Proposition 3.13** For all environments $\mathcal{S}$ and constructed sorts $(r_1, \ldots, r_j)\phi$, where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\phi = \Gamma$, if $g$ is a sort appearing in $\Gamma$, then for all $\Theta$, $\Theta \circ \{r_1/r\alpha_1, \ldots, r_j/r\alpha_j\}(g) = \{\Theta(r_1)/r\alpha_1, \ldots, \Theta(r_j)/r\alpha_j\}(g)$.

**Proof** By definition 3.10, $\Theta \circ \{r_1/r\alpha_1, \ldots, r_j/r\alpha_j\}$ contains all the same bindings as in $\{\Theta(r_1)/r\alpha_1, \ldots, \Theta(r_j)/r\alpha_j\}$, plus any bindings $s_1/r\alpha_i$ in $\Theta$ such that $r\alpha_i$ is not one of $r\alpha_1, \ldots, r\alpha_j$. But by definition 2.14, any sort variables in $g$ are one of $r\alpha_1, \ldots, r\alpha_j$, so clearly the proposition holds.

## 3.2 Derivation of values

As mentioned previously, sets of values are associated with ground sorts. In the present paper we formalize this correspondance in terms of grammars, associating a value with a ground sort iff it can be derived from that sort.

**Definition 3.14** A *sort grammar* consists of terminals in the form of constants and value constructor symbols, non-terminals in the form of ground sorts, and a set of productions. The notation $\Rightarrow_\mathcal{S}$ denotes a valid production step in an environment $\mathcal{S}$. For all environments $\mathcal{S}$, the production associated with a product sort $s = s_1 * \cdots * s_j$ is as follows:

$$s_1 * \cdots * s_j \Rightarrow_\mathcal{S} (s_1, \ldots, s_j)$$

The productions associated with sort symbols are derived from declarations. As is clearly the case, however, sort declarations do not necessarily provide the full set of productions in a grammar. Rather, they serve as a "schema" by which productions can be built up via instantiations specified in the parameter list in a sort symbol. That is, the productions associated with some sort symbol $(r_1, \ldots, r_n)\zeta$ are obtained by instantiating the clauses in $\zeta$'s declaration with the substitution specified by the parameter list $(r_1, \ldots, r_n)$. For example, let $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_n)\zeta = C_1 \mid \cdots \mid C_j \mid F_1(g_1) \mid \cdots \mid F_k(g_k)$. Then the following productions are associated with the sort $(r_1, \ldots, r_n)\zeta^n$:

$$\begin{aligned}
&(r_1, \ldots, r_n)\zeta \Rightarrow_\mathcal{S} C_1 \\
&\vdots \\
&(r_1, \ldots, r_n)\zeta \Rightarrow_\mathcal{S} C_n \\
&(r_1, \ldots, r_n)\zeta \Rightarrow_\mathcal{S} F_1(\{r_1/r\alpha_1, \ldots, r_n/r\alpha_n\}(g_1)) \\
&\vdots \\
&(r_1, \ldots, r_n)\zeta \Rightarrow_\mathcal{S} F_n(\{r_1/r\alpha_1, \ldots, r_n/r\alpha_n\}(g_k))
\end{aligned}$$

Since by assumption each of $r_1, \ldots, r_n$ are ground, and by definition 2.14 any variables appearing in $g_1, \ldots, g_k$ are in $r\alpha_1, \ldots, r\alpha_n$, each production in this grammar takes sort terms to sort terms. And clearly, any sorts appearing in the RHS of one of these productions also has an associated set of productions in the grammar, which are generated in the same manner. We might well ask, then, whether the full set of productions which define the grammar for a given sort is finite. Indeed, this question is intimately related to the decidability issues we are concerned with here, and will be discussed extensively in the next section.

**Definition 3.15** A *derivation* is a sequence of sort terms, where a single ground sort $s$ in the $i$th sort term in the sequence is replaced by a sort term K to produce the $i+1$st sort term in the sequence, such that $s \Rightarrow_{\mathcal{S}} K$ is a valid production in the grammar for $s$ as defined above. Each such step is called a *reduction*. A derivation $s \Rightarrow_{\mathcal{S}} \cdots \Rightarrow_{\mathcal{S}} \mathbf{c}$ of any number of steps, including 0 in the case of the *identity derivation*, can be written $s \Rightarrow^*_{\mathcal{S}} \mathbf{c}$; the shorthand $s \Rightarrow^n_{\mathcal{S}} \mathbf{c}$ is given to mean "$s$ derives $\mathbf{c}$ in $n$ steps." The *identity derivation* takes a sort term to itself.

**Definition 3.16** A *normal form* derivation reduces the leftmost sort in a term first, and reduces product sorts to a tuples before reducing any component sort symbols of the product sort.

**Example 3.17** Let $\mathcal{S}$ be an environment containing the declarations $\beta = T \mid F$ and $(r\alpha)\gamma = N \mid C(r\alpha * (r\alpha)\gamma)$. Then the value $C(T, C(F, N)))$ can be derived from the sort symbol $(\beta)\gamma$ via the following normal form derivation:

$$(\beta)\gamma \Rightarrow_{\mathcal{S}} C(\beta * (\beta)\gamma) \Rightarrow_{\mathcal{S}} C(\beta, (\beta)\gamma) \Rightarrow_{\mathcal{S}} C(T, (\beta)\gamma) \Rightarrow_{\mathcal{S}}$$

$$C(T, C(\beta * (\beta)\gamma)) \Rightarrow_{\mathcal{S}} C(T, C(\beta, (\beta)\gamma)) \Rightarrow_{\mathcal{S}} C(T, C(F, (\beta)\gamma)) \Rightarrow_{\mathcal{S}} C(T, C(F, N))$$

The next few definitions demonstrate some characteristics of derivations. Theorem 3.20 establishes a particularly important result for the analysis of derivations.

**Definition 3.18** A sort symbol $s$ *occurs* in a term $t$ iff:

1. $t$ is a sort symbol, and $t = s$
2. $t$ is a sort term $(t_1, \ldots, t_j)$ or $t_1 * \cdots * t_j$, and there exists $0 < i \le j$ such that $s$ occurs in $t_i$.
3. $t$ is a value constructor term $F(u)$, and $s$ occurs in $u$.

Note that by this definition, any subterm $r$ of a sort symbol $s$ which occurs in $t$ is not an *occurence* of $r$ in $t$; for example, the symbol $(\beta)\gamma$ occurs in $F((\beta)\gamma)$, but the symbol $\beta$ does not.

**Definition 3.19** A sort symbol $r$ *occurs in a derivation* $s \Rightarrow^*_{\mathcal{S}} t$ iff there exists a reduction step $u \Rightarrow_{\mathcal{S}} v$ in $s \Rightarrow^*_{\mathcal{S}} t$ such that $s$ occurs in $u$ or $v$. For example, both $\beta$ and $(\beta)\gamma$ occur in the derivation given in example 3.17, above.

**Theorem 3.20** *For all environments $\mathcal{S}$, sorts $s$ and terms $t$, the following statements hold:*

1. If $s$ is a product sort $s_1 * \cdots * s_j$ and $t$ is a term $(t_1, \ldots, t_j)$ then there exists a derivation $s \Rightarrow^n_{\mathcal{S}} t$ iff for all $0 < i \le j$, there exists a derivation $s_i \Rightarrow^{k_i}_{\mathcal{S}} t_i$, where $k_i < n$.
2. If $s$ is a constructed sort $(s_1, \ldots, s_j)\zeta$ where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\zeta = \Gamma$, and $t$ is a value constructor term $F(u)$, then there exists a derivation $s \Rightarrow^n_{\mathcal{S}} t$ iff there exists $F(g) \in \Gamma$ such that there exists a derivation $\Theta(g) \Rightarrow^k_{\mathcal{S}} u$, where $\Theta = \{s_1/r\alpha_1, \ldots, s_j/r\alpha_j\}$ and $k < n$.

**Proof** 1. Suppose on the one hand that $s_1 * \cdots * s_j \Rightarrow^n_{\mathcal{S}} (t_1, \ldots, t_j)$. By definition 3.16, for all $0 < i \le j$, $s_i$ will be reduced to $t_i$ before any of $s_{i+1}, \ldots, s_j$ are reduced at all. Also, once $s_i$ is reduced to $t_i$, it is not reduced in any subsequent step. Thus, for all $0 < i \le j$, the derivation steps in $s_1 * \cdots * s_j \Rightarrow^n_{\mathcal{S}} (t_1, \ldots, t_j)$ which reduce $s_i$ to $t_i$ are of the form $(t_1, \ldots, t_{i-1}, u, s_{i+1}, \ldots, s_j) \Rightarrow_{\mathcal{S}} (t_1, \ldots, t_{i-1}, v, s_{i+1}, \ldots, s_j)$, where $u \Rightarrow_{\mathcal{S}} v$ is a valid derivation step by definition 3.15. Let $s_i \Rightarrow^{k_i}_{\mathcal{S}} t_i$ be the derivation obtained by replacing each derivation step $(t_1, \ldots, t_{i-1}, u, s_{i+1}, \ldots, s_j) \Rightarrow_{\mathcal{S}} (t_1, \ldots, t_{i-1}, v, s_{i+1}, \ldots, s_j)$ in $s_1 * \cdots * s_j \Rightarrow_{\mathcal{S}}^n (t_1, \ldots, t_j)$ by $u \Rightarrow_{\mathcal{S}} v$. Since each step $u \Rightarrow_{\mathcal{S}} v$ is a valid derivation step, therefore $s_i \Rightarrow_{\mathcal{S}}^{k_i} t_i$ is a valid derivation, by definition 3.15, and clearly $k_i < n$.

Suppose on the other hand that for all $0 < i \leq j$, there exists a derivation $s_i \Rightarrow_{\mathcal{S}}^{k_i} t_i$. For all $0 < i \leq j$, let $(t_1, \ldots, t_{i-1}, s_i, s_{i+1}, \ldots, s_j) \Rightarrow_{\mathcal{S}}^{k_i} (t_1, \ldots, t_{i-1}, t_i, s_{i+1}, \ldots, s_j)$ be the derivation obtained by replacing each derivation step $u \Rightarrow_{\mathcal{S}} v$ in $s_i \Rightarrow_{\mathcal{S}}^{k_i} t_i$ by the derivation step $(t_1, \ldots, t_{i-1}, u, s_{i+1}, \ldots, s_j) \Rightarrow_{\mathcal{S}} (t_1, \ldots, t_{i-1}, v, s_{i+1}, \ldots, s_j)$, which is a valid derivation step by definition 3.15. Let $(s_1, \ldots, s_j) \Rightarrow_{\mathcal{S}}^n (t_1, \ldots, t_j)$ be the derivation obtained by concatenating these derivations so constructed, where $n = k_1 + \cdots + k_j$. Since each derivation step in $(s_1, \ldots, s_j) \Rightarrow_{\mathcal{S}}^n (t_1, \ldots, t_j)$ is valid, and $s_1 * \cdots * s_j \Rightarrow_{\mathcal{S}} (s_1, \ldots, s_j)$ is a valid derivation step by definition 3.14, therefore $s_1 * \cdots * s_j \Rightarrow_{\mathcal{S}}^{n+1} (t_1, \ldots, t_j)$ is a valid derivation, and clearly for all $i$, $k_i < n + 1$.

2. Suppose on the one hand that there exists a derivation $(s_1, \ldots, s_j)\zeta \Rightarrow_{\mathcal{S}}^n F(u)$. Let $(s_1, \ldots, s_j)\zeta \Rightarrow_{\mathcal{S}} F(g')$ be the first step in this derivation. By definition 3.15, this must be a valid derivation step; therefore by definition 3.14, there must exist a value constructor $F(g) \in \Gamma$ such that $g' = \Theta(g)$. Furthermore, let $g' \Rightarrow_{\mathcal{S}}^{n-1} u$ be the derivation obtained by replacing each derivation step $F(v_1) \Rightarrow_{\mathcal{S}} F(v_2)$ in $F(g') \Rightarrow_{\mathcal{S}}^{n-1} F(u)$ by the derivation step $v_1 \Rightarrow_{\mathcal{S}} v_2$, which is a valid derivation step by definition 3.15. Thus, since each step in the derivation $g' \Rightarrow_{\mathcal{S}}^{n-1} u$ is valid, $g' \Rightarrow_{\mathcal{S}}^{n-1} u$ is a valid derivation by definition 3.15, and clearly $n - 1 < n$.

Suppose on the other hand that there exists a value constructor $F(g) \in \Gamma$ such that $g' \Rightarrow_{\mathcal{S}}^n u$, where $g' = \Theta(g)$. Let $F(g') \Rightarrow_{\mathcal{S}}^n F(u)$ be the derivation obtained by replacing each derivation step $v_1 \Rightarrow_{\mathcal{S}} v_2$ in $g' \Rightarrow_{\mathcal{S}}^n u$ by the derivation step $F(v_1) \Rightarrow_{\mathcal{S}} F(v_2)$, which is a valid derivation step by definition 3.15. Thus, since each step in the derivation $F(g') \Rightarrow_{\mathcal{S}}^n F(u)$ is valid, therefore $F(g') \Rightarrow_{\mathcal{S}}^n F(u)$ is a valid derivation, and $(s_1, \ldots, s_j)\zeta \Rightarrow_{\mathcal{S}} F(g')$ s a valid derivation step by definition 3.14, therefore $(s_1, \ldots, s_j)\zeta \Rightarrow_{\mathcal{S}}^{n+1} F(u)$ is a valid derivation, and clearly $n < n + 1$. □

**Definition 3.21** Any sort symbol $s'$ appearing in a derivation $s \Rightarrow_{\mathcal{S}}^* t$ must clearly be reduced to a subterm $t'$ of $t$ in the derivation $s \Rightarrow_{\mathcal{S}}^* t$. Also, by the preceding proposition, it is clear that a derivation $s' \Rightarrow_{\mathcal{S}}^* t'$ can be "extracted" from the derivation derivation $s \Rightarrow_{\mathcal{S}}^* t$, in a manner similar to that described by the proposition. The derivation $s' \Rightarrow_{\mathcal{S}}^* t'$ so obtained is called a *subderivation* of $s \Rightarrow_{\mathcal{S}}^* t$.

With a thorough understanding of derivations, the following definition can be made, which constitutes a semantic interpretation of sort terms.

**Definition 3.22** The set of values corresponding to a sort term $t$ in an environment $\mathcal{S}$ is the set of values which can be derived from $t$ by a valid derivation in $\mathcal{S}$. We write $[t]_{\mathcal{S}}$ to denote the set of values corresponding to $t$ in $\mathcal{S}$. The expression $[t]_{\mathcal{S}}$ is then formally defined as:

$$[t]_{\mathcal{S}} = \{x \mid t \Rightarrow_{\mathcal{S}}^* x\}$$

where $x$ denotes a sort value and $\Rightarrow_{\mathcal{S}}^*$ is the reflexive, transitive closure of $\Rightarrow_{\mathcal{S}}$.

**Example 3.23** Let $\mathcal{S}$ be an environment containing the declarations $\beta = T \mid F$ and $(r\alpha)\gamma = N \mid C(r\alpha * (r\alpha)\gamma)$, and recall the declaration of $\varrho$ which must exist in $\mathcal{S}$ by definition 2.18. Thus:

$$[\beta]_{\mathcal{S}} = \{T, F\}$$

$$[\beta * \varrho]_{\mathcal{S}} = \{(T, O), (F, O)\}$$

$$[(\beta)\gamma]_{\mathcal{S}} = \{N, C(T, N), C(F, N), C(T, C(T, N)), \ldots\}$$

**Theorem 3.24** *For all sort terms $t$ and environments $\mathcal{S}$, the following statements hold for each possible case of $t$:*

1. $[C]_{\mathcal{S}} = \{C\}$

2. $[F(u)]_{\mathcal{S}} = \bigcup_{x \in [u]_{\mathcal{S}}} \{F(x)\}$

3. $[(t_1, \cdots, t_n)]_{\mathcal{S}} = \bigcup_{x_i \in [t_i]_{\mathcal{S}}} \{(x_1, \ldots, x_n)\}$

4. $[s_1 * \cdots * s_n]_{\mathcal{S}} = \bigcup_{x_i \in [s_i]_{\mathcal{S}}} \{(x_1, \ldots, x_n)\}$

5. $[(s_1, \ldots, s_n)\zeta]_{\mathcal{S}} = (\bigcup_{F(r) \in \Gamma} [F(\Theta(r))]_{\mathcal{S}}) \cup (\bigcup_{C \in \Gamma} [C]_{\mathcal{S}})$

   where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_n)\zeta = \Gamma$ and $\Theta = \{s_1/r\alpha_1, \ldots, s_n/r\alpha_n\}$

**Proof** 1. By definition 3.14, $C$ is a terminal symbol, which is also a value, so this statement clearly holds.

2. By definition 3.14, the symbol $F$ in the value constructor term $F(u)$ is a terminal symbol, and by definition 3.1, $u$ is a sort term. Therefore, by definition 3.22, $[u]_{\mathcal{S}} = \{x \mid u \Rightarrow_{\mathcal{S}}^* x\}$, and clearly $\forall x \,.\, (u \Rightarrow_{\mathcal{S}}^* x) \leftrightarrow (F(u) \Rightarrow_{\mathcal{S}}^* F(x))$ by definition 3.14. Therefore, $\forall x \,.\, F(x) \in [F(u)]_{\mathcal{S}} \leftrightarrow x \in [u]_{\mathcal{S}}$, so this satement holds.

3. By definition 3.14, the set $\{x \mid (t_1, \ldots, t_n) \Rightarrow_{\mathcal{S}}^* x\}$ is identical to the set $\{(x_1, \ldots, x_n) \mid \forall 0 < i \leq n \,.\, t_i \Rightarrow_{\mathcal{S}}^* x_i\}$. And by definition 3.22, $\forall 0 < i \leq n \,.\, \forall x_i \,.\, x_i \in [t_i]_{\mathcal{S}} \leftrightarrow t_i \Rightarrow_{\mathcal{S}}^* x_i$. Therefore the set $\{x \mid (t_1, \ldots, t_n) \Rightarrow_{\mathcal{S}}^* x\}$ is identical to the set $\{(x_1, \ldots, x_n) \mid \forall 0 < i \leq n \,.\, x_i \in [t_i]_{\mathcal{S}}\}$, and $\{x \mid (t_1, \ldots, t_n) \Rightarrow_{\mathcal{S}}^* x\} = [(t_1, \ldots, t_n)]_{\mathcal{S}}$ by definition 3.22, so this statement holds.

4. By definition 3.1, $\forall 0 < i \leq n \,.\, s_i$ is a sort term, and since $s_1 * \cdots * s_n \Rightarrow_{\mathcal{S}} (s_1, \ldots, s_n)$ is a valid derivation step by definition 3.14, therefore this statement follows by the same reasoning as in 3.

5. Let $s = (s_1, \ldots, s_n)\zeta$, and let $\Gamma = F_1(g_1) \mid \cdots \mid F_j(g_j) \mid C_1 \mid \cdots \mid C_k$. Then by definition 3.14, for all $0 < i < j$, $s \Rightarrow_{\mathcal{S}} F_i(\Theta(g_i))$ is a valid derivation step, and for all $0 < i < k$, $s \Rightarrow_{\mathcal{S}} C_i$ is a valid derivation step. Thus, the set $\{x \mid s \Rightarrow_{\mathcal{S}}^* x\}$ is equivalent to the set $\cup_{i=1}^{j} \{F_i(y_i) \mid F_i(\Theta(g_i)) \Rightarrow_{\mathcal{S}}^* F(y_i)\} \cup \{C_1, \ldots, C_k\}$. And since for all $0 < i < j$, $F_i(\Theta(g_i)) \Rightarrow_{\mathcal{S}}^* x \leftrightarrow x \in [F_i(\Theta(g_i))]_{\mathcal{S}}$ by definition 3.22, therefore this statement holds. $\square$

### 3.3 Intermediate sorts

The previous section posed the problem of whether any given sort has a finite number of productions associated with its grammar. If we violate the constraint on sort declarations given in definition 2.13, it is easy to see how a sort might be declared which has an infinite grammar. For example, if we add the declaration $(r\alpha)\pi = G \mid F(r\alpha * ((r\alpha)\pi)\pi)$ to an environment $\mathcal{S}$, the grammar associated with the sort $(\varrho)\pi$ contains an infinite number of productions:

$$\begin{aligned}
&(\varrho)\pi \Rightarrow_{\mathcal{S}} G,\\
&(\varrho)\pi \Rightarrow_{\mathcal{S}} F(\varrho * ((\varrho)\pi)\pi),\\
&((\varrho)\pi)\pi \Rightarrow_{\mathcal{S}} G,\\
&((\varrho)\pi)\pi \Rightarrow_{\mathcal{S}} F((\varrho)\pi * (((\varrho)\pi)\pi)\pi),\\
&(((\varrho)\pi)\pi)\pi \Rightarrow_{\mathcal{S}} G,\\
&(((\varrho)\pi)\pi)\pi \Rightarrow_{\mathcal{S}} F(((\varrho)\pi)\pi * ((((\varrho)\pi)\pi)\pi)\pi),\\
&\vdots
\end{aligned}$$

The declaration given for $\pi$ is of course illegal, since $\pi$ has different arguments on the RHS and LHS of its declaration. But even with the constraint on declarations given in definition 2.13 on page 6, it is not evident that these sorts of infinite grammars will not occur.

As discovered through research done for this thesis, the decidability issues we're concerned with are intimately related to these questions. In this thesis, the strategy for solving these questions involves a consideration of sort declarations; thus, the approach is similar to that adopted in [DZ92], where the algorithms given for deciding intersection, emptiness and subtyping for types function by "unrolling" and comparing declarations associated with types. But it is not enough to simply examine uninstantiated declarations in the case of sorts. For example, given the declarations $(r\alpha_1, r\alpha_2)\sigma = L(r\alpha_1) \mid R(r\alpha_2)$ and $(r\alpha_1, r\alpha_2)\kappa = I(r\alpha_1, r\alpha_2)$, and assuming that some nullary sort $\lambda$ is empty, we might ask, is $(\lambda, \varrho)\sigma$ empty? Is $(\lambda, \varrho)\kappa$ empty?

In the former case the answer is no, while in the latter case, the answer is yes. But to decide this, we must make reference to the context of the instantiations in each case– the values constructors for $\sigma$ "polarize" the parameters $(\lambda, \varrho)$, while these parameters both appear in the only value constructor term in the declaration of $\kappa$. The gist of this is that the question of emptiness for some sort $(r_1, \ldots, r_n)\zeta$ is not as simple as determining whether each of $r_1, \ldots, r_n$ is empty, and whether the sorts in $\zeta$'s declaration are empty, which initially seems like a reasonable course of action. Rather, information about the context in which one of $r_1, \ldots, r_n$ might occur in a derivation from $(r_1, \ldots, r_n)\zeta$ is essential.

The strategy we took for this thesis, then, was to carry out the elaboration of sort grammars as far as possible– as far as possible in the sense that, as mentioned, the algorithms are intended to deal with parametric sorts as well as ground sorts. Thus, a full grammar cannot always be obtained, since grammars are only properly associated with ground sorts. The "intermediate" solution was to generate new declarations, which are obtained from the instantiation of pre-existing declarations. For example, given the declaration $(r\alpha)\phi = F(r\alpha * (r\alpha)\phi) \mid C$, the declarations obtained from the sort $(((r\alpha)\phi)\phi)\phi$ are:

$$(((r\alpha)\phi)\phi)\phi = F(((r\alpha)\phi)\phi * (((r\alpha)\phi)\phi)\phi) \mid C$$
$$((r\alpha)\phi)\phi = F((r\alpha)\phi * ((r\alpha)\phi)\phi) \mid C$$

It is also convenient to rename sorts when carrying out these elaborations; for example, $(((r\alpha)\phi)\phi)\phi$ can be renamed as $(r\alpha)\eta_1$, and so long as the correct structure is retained in the declarations, these renamings are insignificant. Thus, the elaborated declarations given above can be given in renamed form as:

$$(r\alpha)\eta_1 = F((r\alpha)\eta_2 * (r\alpha)\eta_1) \mid C$$
$$(r\alpha)\eta_2 = F((r\alpha)\phi * (r\alpha)\eta_2) \mid C$$

and clearly $(((r\alpha)\phi)\phi)\phi$ is the same as $(r\alpha)\eta_1$– that is, any equivalent instantiation of the two will yield equivalent grammars– likewise $((r\alpha)\phi)\phi$ is the same as $(r\alpha)\eta_2$.

These renamed sorts which are asociated with elaborated grammars are called *intermediate* sorts. In the next section, an algorithm is given which generates an intermediate sort corresponding to any given sort. It yields a normalized data structure which is more amenable to decision problems such as emptiness and intersection, and constructively answers the previously posed question about the finiteness of sort grammars, since termination and correctness is proven for the algorithm.

Since intermediate sorts are data structures which will only be used algorithmically in this thesis, a fairly restricted definition is given for them. Essentially, we wish to ensure that any time a constructed intermediate sort appears, it appears with the same parameter sequence as that given in its declaration, and that the declarations of intermediate sorts themselves contain only intermediate sorts. Intermediate sort constructors will be denoted by the symbol $\eta$ from now on; otherwise, intermediate sort syntax is identical to sort syntax.

**Definition 3.25** For all environments $\mathcal{S}$, the deduction rules for intermediate sorts are as follows (where $I(s)$ means "$s$ is an intermediate sort"):

*(a)* intermediate sort variables: $$\frac{}{\mathcal{S} \vdash I(r\alpha)}$$

*(b)* intermediate product sorts: $$\frac{\forall 0 < i \leq j \,.\, \mathcal{S} \vdash I(s_i)}{\mathcal{S} \vdash I(s_1 * \cdots * s_j)}$$

*(c)* intermediate constructed sorts: $$\frac{\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta = \Omega \qquad \forall F(g) \in \Omega \,.\, \mathcal{S} \vdash I(g)}{\mathcal{S} \vdash I((r\alpha_1, \ldots, r\alpha_j)\eta)}$$

Intermediate sorts also stand in a refinement type relation to some type; specifically, the same type that the sort which they elaborate refines.

**Definition 3.26** The intermediate sort refinement type deduction rules in an environment $\mathcal{S}$ are given as follows:

*(a)* intermediate sort variables: $$\frac{}{\mathcal{S} \vdash r\alpha :: \alpha}$$

*(b)* intermediate product sorts: $$\frac{\forall 0 < i \leq j \;.\; \mathcal{S} \vdash s_i :: \mathrm{t}_i}{\mathcal{S} \vdash\ s_1 * \cdots * s_j :: \mathrm{t}_1 * \cdots * \mathrm{t}_j}$$

*(c)* intermediate constructed sorts: $$\frac{(r\alpha_1, \ldots, r\alpha_k)\eta :: (\mathrm{t}_1, \ldots, \mathrm{t}_j)\mathrm{c} \;\text{ in annotation}}{\mathcal{S} \vdash\ (r\alpha_1, \ldots, r\alpha_k)\eta :: (\mathrm{t}_1, \ldots, \mathrm{t}_j)\mathrm{c}}$$

"In annotation" in the third rule signifies that a refinement type relation has been deduced from a declaration, the rules for which are given below. The well-formedness rules for intermediate sort declarations reflect the fact that these declarations represent elaborated sort declarations, with instantiations which have been "carried through" from the parameter list in a constructed sort.

**Definition 3.27** The deduction rules for well-formedness of an intermediate sort declaration are as follows:

*(a)* intermediate sort clauses:

$$\frac{\forall C \in \Omega \;.\; C \in \mathrm{G} \qquad \forall F(g) \in \Omega \;.\; \exists F(\mathrm{t}) \in \mathrm{G} \;.\; \mathcal{S} \vdash g :: \Xi(\mathrm{t})}{\mathcal{S} \vdash \Omega :: (\Xi, \mathrm{G})}$$

*(b)* intermediate sort declarations:

$$\frac{\Xi = \{\mathrm{t}_1/\alpha_1, \ldots, \mathrm{t}_j/\alpha_j\} \qquad \mathcal{S} \vdash \Omega :: (\Xi, \mathrm{G})}{\mathcal{S} \vdash\ ((r\alpha'_1, \ldots, r\alpha'_k)\eta = \Omega) :: (\Xi, (\alpha_1, \ldots, \alpha_j)\phi = \mathrm{G})}$$ where $(\alpha'_1, \ldots, \alpha'_k)$ are the type variables in $(\mathrm{t}_1, \ldots, \mathrm{t}_j)$

It is also required that the parameter list of a sort constructor in its declaration contains bindings for any sort variable appearing in the RHS of its declaration. As it turns out, this is automatically ensured by the well-formedness rules given above, as is demonstrated by proposition 3.29 below.

**Proposition 3.28** For all intermediate sorts $s$, types $\mathrm{t}$ and environments $\mathcal{S}$, if $\mathcal{S} \vdash s :: \mathrm{t}$ then the sort variables in $s$ are refinements of the type variables in $\mathrm{t}$.

**Proof** By structural induction on $s$.

*case (i)* If $s$ is a sort variable $r\alpha$, then $\mathrm{t} = \alpha$ by definition 3.26.

*case (ii)* If $s$ is a product sort $s_1 * \cdots * s_j$, then $\mathrm{t} = \mathrm{t}_1 * \cdots * \mathrm{t}_j$ and $\forall 0 < i \leq j \;.\; \mathcal{S} \vdash s_i :: \mathrm{t}_i$ by definition 3.26, so this case holds by the induction hypothesis.

*case (iii)* If $s$ is an intermediate constructed sort, then each of the sort variables in $s$ are refinements of the type variables in $t$ by definitions 3.27 and 3.25. $\square$

**Proposition 3.29** Let $\mathcal{S} \vdash (\alpha_1, \ldots, \alpha_j)\mathrm{c} = \mathrm{G}$, let $\Xi = \{\mathrm{t}_1/\alpha_1, \ldots, \mathrm{t}_j/\alpha_j\}$, and suppose that for some set of intermediate sort clauses $\Omega$, $\mathcal{S} \vdash \Omega :: (\Xi, \mathrm{G})$. Then any sort variables appearing in $\Omega$ are refinements of the type variables in $(\mathrm{t}_1, \ldots, \mathrm{t}_j)\mathrm{c}$.

**Proof** Since $\mathcal{S} \vdash \Omega :: (\Xi, \mathrm{G})$ by assumption, therefore for all $F(g) \in \Omega$ there exists $F(\mathrm{g}) \in \mathrm{G}$ such that $\mathcal{S} \vdash g :: \Xi(\mathrm{g})$. And, any variable $\alpha_i$ in g must be one of $(\alpha_1, \ldots, \alpha_j)$, by definition 2.8, therefore any variable in g must have a binding in $\Xi$. Now, since any variable in g must have a binding in $\Xi$, therefore any variables in $\Xi(\mathrm{g})$ must be in $\mathrm{t}_1, \ldots, \mathrm{t}_j$, the bindings in $\Xi$. And therefore any variables in $\Xi(\mathrm{g})$ are clearly in $(\mathrm{t}_1, \ldots, \mathrm{t}_j)\mathrm{c}$. But since $\mathcal{S} \vdash g :: \Xi(\mathrm{g})$, therefore by proposition 3.28, the sort variables in $g$ are refinements of the type variables in $\Xi(\mathrm{g})$. Therefore, for all $F(g) \in \Omega$, the sort variables in $g$ are refinements of the type variables in $(\mathrm{t}_1, \ldots, \mathrm{t}_j)\mathrm{c}$, so the proposition holds. $\square$

Intermediate sorts can also be instantiated in the same way as normal sorts, and instantiated intermediate sorts are also associated with grammars in the same way as normal sorts. That is, definitions 3.3 and 3.22 also apply to intermediate sorts. Thus, theorem 3.24 also holds for instantiated intermediate sorts. However, since every constructed intermediate sort is in uninstantiated form (if it is not nullary), as are all the sorts in the RHS of any intermediate sort declaration, the process of obtaining a grammar from an instantiated sort is greatly simplified, as is demonstrated by the following proposition.

**Proposition 3.30** For all environments $\mathcal{S}$, intermediate constructed sorts $(r\alpha_1, \ldots, r\alpha_n)\eta$ and ground substitutions $\Theta$, where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_n)\eta = \Omega$, the following statements hold:

1. $[\Theta((r\alpha_1, \ldots, r\alpha_n)\eta)]_{\mathcal{S}} = \bigcup_{F(g)\in\Omega} [F(\Theta(g))]_{\mathcal{S}} \cup \{C \mid C \in \Omega\}$
2. $\Theta((r\alpha_1, \ldots, r\alpha_n)\eta) \Rightarrow^n_{\mathcal{S}} F(t)$ iff there exists a value constructor $F(g) \in \Omega$, and $\Theta(g) \Rightarrow^k_{\mathcal{S}} t$, with $k < n$.

**Proof** 1. By definition 3.25, the sort variable sequence $(r\alpha_1, \ldots, r\alpha_n)$ is identical to the parameters of $\eta$ in its declaration, and by definition 3.3, $\Theta((r\alpha_1, \ldots, r\alpha_n)\eta) = (\Theta(r\alpha_1), \ldots, \Theta(r\alpha_n))\eta$. Let $\Theta' = \{\Theta(r\alpha_1)/r\alpha_1, \ldots, \Theta(r\alpha_n)/r\alpha_n\}$. Then by theorem 3.24:

$$[\Theta((r\alpha_1, \ldots, r\alpha_n)\eta)]_{\mathcal{S}} = \bigcup_{F(g)\in\Omega} [F(\Theta'(g))]_{\mathcal{S}} \cup \{C \mid C \in \Omega\}$$

But clearly every binding in $\Theta'$ is in $\Theta$, and any sort variable appearing in $g$ must be one of $r\alpha_1, \ldots, r\alpha_n$ by definition 2.14, thus $\Theta'(g) = \Theta(g)$, so this statement holds.

2. This follows from 1 and theorem 3.20 $\square$

# 4 Sort elaboration

The algorithm for elaborating sort declarations, called *elab*, is given in figure 1. It "unrolls" the definition of a sort $s$ by examining declarations associated with the constructors in $s$, and returns an intermediate sort $s'$ defined in an environment $\mathcal{S}'$ which is equivalent to $s$. More specifically,

$elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M}) =$

let $\langle(\Theta_1, \zeta_1), \ldots, (\Theta_n, \zeta_n)\rangle = \mathcal{R}$

- if $s = r\alpha_i$ then if $n = 0$ return $(r\alpha_i, \mathcal{S})$ else return $elab(\Theta_n(r\alpha_i), \mathcal{S}, \langle(\Theta_1, \zeta_1), \ldots, (\Theta_{n-1}, \zeta_{n-1})\rangle, \mathcal{M})$
- if $s = s_1 * \cdots * s_j$ then:
  - $\forall 0 < i \leq j$ . let $(s'_i, \mathcal{S}_i) = elab(s_i, \mathcal{R}, \mathcal{S}, \mathcal{M})$
  - return $(s'_1 * \cdots * s'_j, \cup_{i=1}^{j} \mathcal{S}_i)$
- if $s = (s_1, \ldots, s_j)\phi$ then:
  - let $r = \Theta_1 \circ \cdots \circ \Theta_n(s)$
  - if $\exists s'$ . $(r, s') \in \mathcal{M}$ then return $(s', \mathcal{S})$
  - otherwise, where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\phi = \Gamma$:

    let:
    $(r\alpha_1^f, \ldots, r\alpha_k^f) =$ the sort variables in $r$
    $\Theta_f = \{s_1/r\alpha_1, \ldots, s_j/r\alpha_j\}$
    $\Omega = C_1 \mid \cdots \mid C_i$, where $C_1, \ldots, C_i$ are the constants in $\Gamma$
    $\mathcal{S}_f = \mathcal{S} \cup \left\{(r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \circ\right\}$, where $\eta_f$ is a fresh intermediate sort constructor symbol
    $\mathcal{M}_f = \mathcal{M} \cup \left\{(r, (r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f)\right\}$
    $\mathcal{R}_f = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_n, \zeta_n), (\Theta_f, \phi)\rangle$

    for all $F(g) \in \Gamma$:
    let $(g', \mathcal{S}_{g'}) = elab(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$
    let $\Omega = \Omega \mid F(g')$

    let $\mathcal{S}' = \bigcup_{F(g') \in \Omega} \mathcal{S}_{g'} - \left\{(r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \circ\right\}$
    let $\mathcal{S}'' = \mathcal{S} \cup \mathcal{S}' \cup \left\{(r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \Omega\right\}$
    return $((r\alpha_1, \ldots, r\alpha_k)\eta_f, \mathcal{S}'')$

**Figure 1:** The *elab* algorithm

the result which is claimed for *elab* is that for all sorts $s$, environments $\mathcal{S}$ and ground substitutions $\Theta$ (and where $\langle\rangle$ denotes an empty sequence), $elab(s, \mathcal{S}, \langle\rangle, \emptyset)$ terminates, returning an (intermediate sort, environment) pair $(s', \mathcal{S}')$ such that $[\Theta(s)]_{\mathcal{S}} = [\Theta(s')]_{\mathcal{S}'}$, leaving all pre-existing definitions unchanged (see lemma 4.28 for a formal definition of this latter condition). This result requires a considerable amount of proof, which is given in the next few sections.

The *elab* algorithm memoizes sort symbols, so that intermediate sort definitions corresponding to previously encountered sort definitions will not be recomputed. Of course, simply memoizing sort symbols which have already been encountered in the computation begs the question concerning the finiteness of sort grammars, and doesn't provide any insight regarding termination of the algorithm. Thus, *elab* uses a data structure, termed an *accumulator sequence*, having properties which support a proof of termination. The accumulator sequence essentially "puts off" the instantiations which are carried through sort declarations, accumulating them in the form of substitutions, and applying these substitutions when appropriate. This allows the algorithm to deal with sort declarations in uninstantiated form, revealing aspects of sort grammars that imply their finiteness.

**Example 4.1** Suppose that an environment $\mathcal{S}$ contains the declarations $(r\alpha)\nu = N \mid C(r\alpha * (r\alpha)\nu)$ and $\beta = T \mid F$ (which define list and boolean sorts). The following diagram represents the recursive call tree resulting from the initial call $elab((\beta)\nu, \mathcal{S}, \langle\rangle, \emptyset)$:

$$
\begin{tikzcd}
 & elab((\beta)\nu, \mathcal{S}, \langle\rangle, \emptyset) \arrow[d] & \\
 & elab(r\alpha * (r\alpha)\nu, \mathcal{S} \cup \{\eta_{f_1} = \circ\}, \langle(\{\beta/r\alpha\}, \nu)\rangle, \{((\beta)\nu, \eta_{f_1})\}) \arrow[dl] \arrow[dr] & \\
elab(r\alpha, \mathcal{S} \cup \{\eta_{f_1} = \circ\}, \langle(\{\beta/r\alpha\}, \nu)\rangle, \{((\beta)\nu, \eta_{f_1})\}) \arrow[d] & & elab((r\alpha)\nu, \mathcal{S} \cup \{\eta_{f_1} = \circ\}, \langle(\{\beta/r\alpha\}, \nu)\rangle, \{((\beta)\nu, \eta_{f_1})\}) \\
elab(\beta, \mathcal{S} \cup \{\eta_{f_1} = \circ\}, \langle\rangle, \{((\beta)\nu, \eta_{f_1})\}) & &
\end{tikzcd}
$$

These calls return the following values:

$$
\begin{tikzcd}
 & (\eta_{f_1}, \mathcal{S} \cup \{\eta_{f_1} = N \mid C(\eta_{f_2} * \eta_{f_1}),\ \eta_{f_2} = T \mid F\}) \arrow[d, dash] & \\
 & (\eta_{f_2} * \eta_{f_1}, \mathcal{S} \cup \{\eta_{f_1} = \circ,\ \eta_{f_2} = T \mid F\}) \arrow[dl, dash] \arrow[dr, dash] & \\
(\eta_{f_2}, \mathcal{S} \cup \{\eta_{f_1} = \circ,\ \eta_{f_2} = T \mid F\}) \arrow[d, dash] & & (\eta_{f_1}, \mathcal{S} \cup \{\eta_{f_1} = \circ\}) \\
(\eta_{f_2}, \mathcal{S} \cup \{\eta_{f_1} = \circ,\ \eta_{f_2} = T \mid F\}) & &
\end{tikzcd}
$$

It is easily verified that $\eta_{f_1}$ in the environment $\mathcal{S} \cup \{\eta_{f_1} = N \mid C(\eta_{f_2} * \eta_{f_1}),\ \eta_{f_2} = T \mid F\}$ is a correct elaboration of $(\beta)\nu$ in $\mathcal{S}$; that is, $\eta_{f_1}$ and $(\beta)\nu$ are associated with equivalent grammars.

## 4.1 Invariants of *elab*

In this section, some basic groundwork will be established for the later termination, soundness and completeness proofs.

**Definition 4.2** An *accumulator sequence* $\mathcal{R}$ is an ordered sequence of pairs $(\Theta, \zeta)$ such that each $\zeta$ is a sort constructor and each $\Theta$ is a substitution which contains bindings only for each of $(r\alpha_1, \ldots, r\alpha_n)$, where $(r\alpha_1, \ldots, r\alpha_n)$ is the parameter sequence preceding $\zeta$ in the LHS of its declaration. In any such pair $(\Theta, \zeta)$ in an accumulator sequence, $\zeta$ is called the *tag* for that pair.

**Definition 4.3** An accumulator sequence $\mathcal{R}_1 = \langle(\Theta^1_1, \zeta^1_1), \ldots, (\Theta^1_n, \zeta^1_n)\rangle$ is an *initial sequence* of an accumulator sequence $\mathcal{R}_2 = \langle(\Theta^2_1, \zeta^2_1), \ldots, (\Theta^2_m, \zeta^2_m)\rangle$ if and only if $n \leq m \wedge \forall 0 < i \leq n \,.\, (\Theta^1_i, \zeta^1_i) = (\Theta^2_i, \zeta^2_i)$; $\mathcal{R}_1$ is a *strictly* initial sequence of $\mathcal{R}_2$ if $n < m$.

**Definition 4.4** A matching $\mathcal{M}$ is a set of pairs $(s_1, s_2)$ such that $s_1$ is a constructed sort and $s_2$ is an intermediate constructed sort. $s_1$ is called a *key* in $\mathcal{M}$, and $s_2$ is called a *matching for* $s_1$ in $\mathcal{M}$.

The following definition gives the desired representation invariant for the parameters of *elab*. In a subsequent lemma, this invariant will be shown to hold in any recursive call to *elab*.

**Definition 4.5** For all 4-tuples $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$, the relation *evalid* holds iff each of the following conditions are satisfied:

1. $\mathcal{S}$ is an environment consisting of a finite number of sort declarations, $s$ is a sort, $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_n, \zeta_n)\rangle$ is an accumulator sequence, and $\mathcal{M}$ is a matching.
2. $\mathcal{S} \vdash s$, $\forall 0 < i \leq n$ . $\zeta_i$ is defined in $\mathcal{S}$, each binding in $\Theta_i$ is defined in $\mathcal{S}$, and for all $(s_1, s_2) \in \mathcal{M}$, $\mathcal{S} \vdash s_1$ and $\mathcal{S} \vdash s_2$.
3. If $(s_1, (r\alpha_1, \ldots, r\alpha_n)\eta) \in \mathcal{M}$, then $\eta$ is not parametrically reachable by any tag in $\mathcal{R}$, nor by any constructor symbol in any substitution in $\mathcal{R}$, nor by any constructor symbol in $s$.
4. For all $1 < i \leq n$, $\Theta_i((r\alpha_1^i, \ldots, r\alpha_x^i)\zeta_i)$ is a subterm of a sort appearing in the RHS of $\zeta_{i-1}$'s declaration (where $(r\alpha_1^i, \ldots, r\alpha_x^i)$ is the parameter sequence preceding each $\zeta_i$ in its declaration), and $s$ is a subterm of a sort appearing in the RHS of $\zeta_n's$ declaration.
5. For all $1 < i \leq n$, $\Theta_1 \circ \cdots \circ \Theta_i((r\alpha_1^i, \ldots, r\alpha_x^i)\zeta_i)$ is a key in $\mathcal{M}$, where $(r\alpha_1^i, \ldots, r\alpha_x^i)$ is the parameter sequence preceding $\zeta_i$ in its declaration.

The following propositions, which follow from the preceding definition, demonstrate the effects which the constraint on declarations given in definition 2.13 has on sort elaboration, and how these effects are reflected in the accumulator sequences used by *elab*. These propositions are of central importance to the proof of termination for *elab*.

**Proposition 4.6** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_n, \zeta_n)\rangle$, if $s$ is a sort constructor term $(r_1, \ldots, r_j)\phi$ and $\exists 0 < i \leq n$ . $\phi = \zeta_i$, then each of $\zeta_i, \ldots, \zeta_n$ and $\phi$ are mutually recursive, and each of $\Theta_{i+1}, \ldots, \Theta_n$ is the identity substitution and $(r_1, \ldots, r_j)$ is identical to the parameter sequence preceding $\phi$ in its declaration.

**Proof** Suppose that for a particular 4-tuple $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, $s$ is a sort constructor term $(r_1, \ldots, r_j)\phi$ and $\exists 0 < i \leq n$ . $\phi = \zeta_i$. Let $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_n, \zeta_n)\rangle$. By definition 4.5, for all $i < j \leq n$, $\Theta_j((r\alpha_1^j, \ldots, r\alpha_x^j)\zeta_j)$ is a subterm of a sort appearing in the RHS of $\zeta_{j-1}$'s declaration, where $(r\alpha_1^j, \ldots, r\alpha_x^j)$ is the parameter sequence preceding each $\zeta_j$ in the LHS of its declaration, and $s$ is a subterm of a sort appearing in the RHS of $\zeta_n's$ declaration. Hence, by definition 2.11, for all $i < j \leq n$, $\zeta_j$ is parametrically reachable by $\zeta_{j-1}$, therefore each $\zeta_j$ is parametrically reachable by $\zeta_i$, by definition 2.11. And $\phi$ is parametrically reachable from $\zeta_n$ by definitions 4.5 and 2.11, and therefore parametrically reachable by each of $\zeta_i, \ldots, \zeta_n$. But $\phi = \zeta_i$ by assumption, therefore by definition 2.12, all of $\zeta_i, \ldots, \zeta_n$ and $\phi$ are mutually recursive.

Now, if for all $i < j \leq n$, $\Theta_j((r\alpha_1^j, \ldots, r\alpha_x^j)\zeta_j)$ is a subterm of a sort appearing in the RHS of $\zeta_{j-1}$'s declaration, and all of $\zeta_i, \ldots, \zeta_n$ are mutually recursive, then each $\Theta_j$ must be the identity substitution, since $(r\alpha_1^j, \ldots, r\alpha_x^j)$ is the sort sequence preceding $\zeta_j$ in the RHS of $\zeta_{j-1}$'s declaration, by the constraint which has been imposed on mutually recursive definitions (see definition 2.14). Likewise, $(r_1, \ldots, r_j)$ is the parameter sequence preceding $\phi$ in the LHS of it's declaration, since $\zeta_n$ and $\phi$ are mutually recursive, and $(r_1, \ldots, r_j)\phi$ appears in the RHS of $\zeta_n's$ declaration by definition 4.5. Therefore the proposition holds. $\square$

**Proposition 4.7** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that *evalid*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_n, \zeta_n)\rangle$, if $s$ is a sort constructor term $(r_1, \ldots, r_j)\phi$ and $\exists 0 < i \leq n \,.\, \phi = \zeta_i$, then $\Theta_1 \circ \cdots \circ \Theta_n((r_1, \ldots, r_j)\phi)$ is a key in $\mathcal{M}$.

**Proof** Suppose *evalid*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_n, \zeta_n)\rangle$, and $s$ is a sort constructor term $(r_1, \ldots, r_j)\phi$ and $\exists 0 < i \leq n \,.\, \phi = \zeta_i$. Then by the preceding corollary, each of $\Theta_{i+1}$ through $\Theta_n$ is the identity substitution, and $(r_1, \ldots, r_j)$ is identical to the parameter sequence preceding $\phi$ in its declaration. But then $\Theta_1 \circ \cdots \circ \Theta_n((r_1, \ldots, r_j)\phi) = \Theta_1 \circ \cdots \circ \Theta_i((r\alpha_1, \ldots, r\alpha_j)\phi)$ by proposition 3.12, where $(r\alpha_1, \ldots, r\alpha_j)$ is the parameter sequence preceding $\phi$ in its declaration. And since *evalid*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, then by definition 4.5, $\Theta_1 \circ \cdots \circ \Theta_i((r\alpha_1, \ldots, r\alpha_j)\phi)$ is a key in $\mathcal{M}$, hence $\Theta_1 \circ \cdots \circ \Theta_n((r_1, \ldots, r_j)\phi)$ is a key in $\mathcal{M}$. $\square$

The following lemmas demonstrate the preservation of the representation invariant given in definition 4.5 for *elab*.

**Lemma 4.8** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that *evalid*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, if *elab*$(s', \mathcal{S}', \mathcal{R}', \mathcal{M}')$ is an immediate recursive call of *elab*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$, then *evalid*$(s', \mathcal{S}', \mathcal{R}', \mathcal{M}')$ holds.

**Proof** Let there be $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that *evalid*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, and let $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_n, \zeta_n)\rangle$. The proof then proceeds by a consideration of those cases in which a recursive call to *elab*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ occurs:

*case (i)* $s = r\alpha_i$, and $n > 0$. In this case, by the definition of *elab*, the recursive call *elab*$(\Theta_n(r\alpha_i), \mathcal{S}, \mathcal{R}', \mathcal{M})$ is made, where $\mathcal{R}' = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_{n-1}, \zeta_{n-1})\rangle$. To show that each of the conditions in 4.5 are satisfied:

1. By assumption $\mathcal{R}$ is an accumulator sequence, therefore $\mathcal{R}'$ is an accumulator sequence, since $\mathcal{R}'$ is an initial sequence of $\mathcal{R}$. And since $\Theta_n$ is a sort substitution, therefore $\Theta_n(r\alpha_1)$ is a sort. Since *evalid*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds by assumption, $\mathcal{S}$ is a finite environment and $\mathcal{M}$ is a matching from sort constructor terms to sort constructor terms.

2. Since by definition 4.5 every binding in $\Theta_n$ is defined in $\mathcal{S}$, therefore $\mathcal{S} \vdash \Theta_n(r\alpha_i)$. Each of $\zeta_1, \ldots, \zeta_{n-1}$ are defined in $\mathcal{S}$ by assumption, as are $\Theta_1, \ldots, \Theta_{n-1}$. And for all $(s_1, s_2) \in \mathcal{M}$, $\mathcal{S} \vdash s_1$ and $\mathcal{S} \vdash s_2$ since *evalid*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds by assumption.

3. If $(s_1, (r\alpha_1, \ldots, r\alpha_n)\eta) \in \mathcal{M}$, then $\eta$ is not parametrically reachable by any tag in $\mathcal{R}'$, nor by any constructor symbol in any substitution in $\mathcal{R}'$, since $\mathcal{R}'$ is an initial sequence of $\mathcal{R}$, and *evalid*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds by assumption. And since also $\eta$ is not parametrically reachable by any constructor symbol in any binding in $\Theta_n$, therefore $\eta$ is not a constructor symbol in $\Theta_n(r\alpha_i)$.

4. This condition also holds because $\mathcal{R}'$ is an initial sequence of $\mathcal{R}$.

5. This condition also holds because $\mathcal{R}'$ is an initial sequence of $\mathcal{R}$.

*case (ii)* $s = s_1 * \cdots * s_j$. In this case, by the definition of *elab*, for all $0 < i \leq j$ the recursive calls $elab(s_i, \mathcal{S}, \mathcal{R}, \mathcal{M})$ are made. Since by definition 2.7, $\forall 0 < i \leq j \;.\; \mathcal{S} \vdash s_i$, and $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ by assumption, therefore all the conditions in definition 4.5 are clearly satisfied, so that $\forall 0 < i \leq j \;.\; evalid(s_i, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds.

*case (iii)* $s = (s_1, \ldots, s_j)\phi$, and $\neg\exists(r_1, r_2) \in \mathcal{M}$ such that $r_1 = \Theta_1 \circ \cdots \circ \Theta_n(s)$.

Let:

$$\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\phi = \Gamma$$
$$\Theta_f = \{r_1/r\alpha_1, \ldots, r_j/r\alpha_j\}$$
$$(r\alpha_1^f, \ldots, r\alpha_k^f) = \text{the sort variables in } (\Theta_1 \circ \cdots \circ \Theta_n(s))$$
$$\mathcal{M}_f = \mathcal{M} \cup \left\{(\Theta_1 \circ \cdots \circ \Theta_n(s), (r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f)\right\}, \text{ where } \eta_f \text{ is fresh}$$
$$\mathcal{R}_f = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_n, \zeta_n), (\Theta_f, \phi)\rangle$$
$$\mathcal{S}_f = \mathcal{S} \cup \left\{(r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \circ\right\}$$

Then in this case, by the definition of *elab*, for each $F(g) \in \Gamma$ the recursive call $elab(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$ occurs. To show that for each such recursive call, $evalid(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$ holds:

1. Since $\mathcal{S}$ is a finite environment of sort delarations by assumption, and $\mathcal{S}_f = \mathcal{S} \cup \{\, (r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \circ \,\}$, therefore $\mathcal{S}_f$ is a finite environment of sort declarations. Since $F(g)$ is a value constructor term in the definition of $\phi$, $g$ is a sort. Clearly, there are bindings in $\Theta_f$ only for each of $(r\alpha_1, \ldots, r\alpha_j)$, and $\mathcal{R}$ is an accumulator sequence by assumption, so by definition 4.2 $\mathcal{R}'$ is an accumulator sequence, and also $\mathcal{M}'$ is a matching, since $\mathcal{M}$ is by assumption and $(\Theta_1 \circ \cdots \circ \Theta_n(s), (r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f)$ is a (constructed sort, intermediate constructed sort) pair. Therefore this condition holds.

2. Since $\mathcal{S}_f = \mathcal{S} \cup \left\{(r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \circ\right\}$, so that the declarations in $\mathcal{S}$ are subsumed in $\mathcal{S}_f$, therefore any sort which is defined in $\mathcal{S}$ is defined in $\mathcal{S}_f$. Thus, each of $\zeta_1, \ldots, \zeta_n$ and $\phi$ are defined in $\mathcal{S}_f$, as are each of the bindings in $\Theta_1, \ldots, \Theta_n$ and $\Theta_f$, since $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds by assumption; therefore also $\mathcal{S}_f \vdash \Theta_1 \circ \cdots \circ \Theta_n(s)$. Since $F(g) \in \Gamma$, therefore $\mathcal{S} \vdash g$, by definition 2.14, therefore $\mathcal{S}_f \vdash g$. And clearly $\mathcal{S}_f \vdash (r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f$, so that for all $(s_1, s_2) \in \mathcal{M}_f$, $\mathcal{S}_f \vdash s_1$ and $\mathcal{S}_f \vdash s_2$. Therefore this condition holds.

3. By assumption, for all $0 < i \leq n$, the bindings in $\Theta_i$ are defined in $\mathcal{S}$, as are each $\zeta_i$. Also, $\mathcal{S} \vdash (r_1, \ldots, r_j)\phi$, so $\mathcal{S} \vdash g$ by 2.14, and each of of $r_1, \ldots, r_j$, the bindings in $\Theta_f$, are defined in $\mathcal{S}$. Thus, since $\eta_f$ is a fresh constructor symbol, i.e. is not defined in $\mathcal{S}$, $\eta_f$ is not parametrically reachable by any tag in $\mathcal{R}_f$ nor any constructor symbol appearing in the substitutions in $\mathcal{R}_f$. Also, for every pair $(s_1, s_2) \in \mathcal{M}$, the outermost constructor symbol $\eta$ in any $s_2$ is not parametrically reachable by any constructor symbol in $(r_1, \ldots, r_j)\phi$ by assumption, nor by any tags in $\mathcal{R}$ or any constructor symbol in any substitution in $\mathcal{R}$, nor by any constructor symbol in $g$. Thus, since $\mathcal{M}_f = \mathcal{M} \cup \left\{(\Theta_1 \circ \cdots \circ \Theta_n(s), (r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f)\right\}$, and $\mathcal{R}_f = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_n, \zeta_n), (\Theta_f, \phi)\rangle$, therefore this condition holds.

4. Clearly, $g$ is a subterm of a sort in $\phi$'s declaration, since $F(g) \in \Gamma$ by assumption. Also, $(r_1,\ldots,r_j)\phi$ is a subterm of a sort appearing in the RHS of $\zeta_n$'s declaration by assumption, therefore $\Theta_f((r\alpha_1,\ldots,r\alpha_j)\phi)$ is a subterm of a sort appearing in the RHS of $\zeta_n$'s declaration. And by assumption, $\forall 0<i\leq n$ . $\Theta_i((r\alpha_1^i,\ldots,r\alpha_x^i)\zeta_i)$ is a subterm of a sort appearing in the RHS of $\zeta_{i-1}$'s declaration. Therefore, since $\mathcal{R}_f = \langle(\Theta_1,\zeta_1),\ldots,(\Theta_n,\zeta_n),(\Theta_f,\phi)\rangle$, this condition holds.

5. Since by the definition of *elab*, $(\Theta_1\circ\cdots\circ\Theta_n(s),(r\alpha_1^f,\ldots,r\alpha_k^f)\eta_f)$ is added to $\mathcal{M}$ to yield $\mathcal{M}_f$, this condition holds.

This covers all the cases of lemma 4.8 which were necessary to prove. $\Box$

**Lemma 4.9** For all $(s,\mathcal{S},\mathcal{R},\mathcal{M})$ such that $evalid(s,\mathcal{S},\mathcal{R},\mathcal{M})$ holds, if $elab(s',\mathcal{S}',\mathcal{R}',\mathcal{M}')$ is any descendant recursive call of $elab(s,\mathcal{S},\mathcal{R},\mathcal{M})$, then $evalid(s',\mathcal{S}',\mathcal{R}',\mathcal{M}')$ holds.

**Proof** This follows immediately from lemma 4.8. $\Box$

The following definition and lemma are helpful for the demonstration of later proofs. The need for them arises when these proofs must deal with the case in which *elab* is called on a sort variable. In this case, the sort variable is instantiated using the last substitution in the accumulator sequence (if one exists); but this instantiation may also yield a sort variable, and so on. The results obtained from the lemma greatly simplify how this situation is dealt with.

**Definition 4.10** For all compositions of substitutions $\Theta_1\circ\cdots\circ\Theta_n$ and sorts $s$, the first sort in the sequence $s$, $\Theta_n(s)$, $\Theta_{n-1}(\Theta_n(s))$, …, $\Theta_1(\cdots(\Theta_n(s))\cdots)$ which is not a sort variable, if one exists, is called the *primary sort* of $\Theta_1\circ\cdots\circ\Theta_n(s)$. It follows from this, definition 3.3 and proposition 3.12 that for all substitutions $\Theta$ such that $s' = \Theta\circ\Theta_1\circ\cdots\circ\Theta_n(s)$ is ground, if there exists a primary sort $r = \Theta_i(\cdots(\Theta_n(s))\cdots)$ in $\Theta_1\circ\cdots\circ\Theta_n(s)$, then if $s' = s_1' * \cdots * s_j'$ then $r = r_1 * \cdots * r_j$, and if $s' = (s_1',\ldots,s_j')\zeta$ then $r = (r_1,\ldots,r_j)\zeta$.

**Example 4.11** $\varrho * (r\alpha_1)\phi$ is the primary sort of $\{\varrho/r\alpha_1\}\circ\{\varrho*(r\alpha_1)\phi/r\alpha_1\}\circ\{r\alpha_1/r\alpha_2\}\,(r\alpha_2)$. $(r\alpha_1,\varrho)\beta$ is the primary sort of $\{\varrho/r\alpha_1\}\circ\{(r\alpha_1,\varrho)\beta/r\alpha_1\}\,(r\alpha_1)$.

**Lemma 4.12** For all $(s,\mathcal{S},\mathcal{R},\mathcal{M})$ such that $evalid(s,\mathcal{S},\mathcal{R},\mathcal{M})$ holds, where $\mathcal{R} = \langle(\Theta_1,\zeta_1),\ldots,(\Theta_m,\zeta_m)\rangle$, the following two claims hold for $elab(s,\mathcal{S},\mathcal{R},\mathcal{M})$:

***claim(1):*** If there exists a primary sort $r = \Theta_i(\cdots(\Theta_m(s))\cdots)$ in $\Theta_1\circ\cdots\circ\Theta_m(s)$, then $elab(s,\mathcal{S},\mathcal{R},\mathcal{M}) = elab(r,\mathcal{S},\mathcal{R}',\mathcal{M})$, where $\mathcal{R}' = \langle(\Theta_1,\zeta_1),\ldots,(\Theta_{i-1},\zeta_{i-1})\rangle$.

**Proof** By induction on $n$, the number of substitution applications in $\Theta_i(\cdots(\Theta_m(s))\cdots)$, the primary sort of $\Theta_1\circ\cdots\circ\Theta_m(s)$.

**Basis** $(n = 0)$ If $n = 0$, then $r = s$, and $\mathcal{R}' = \mathcal{R}$, and therefore clearly $elab(s,\mathcal{S},\mathcal{R},\mathcal{M}) = elab(r,\mathcal{S},\mathcal{R}',\mathcal{M})$ in this case.

**Induction Hypothesis** The claim holds for all $i < n$.

**Induction Step** $(n > 0)$ In this case, $s$ is a sort variable $r\alpha$ by definition 4.10. Therefore, by the definition of *elab*, $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M}) = elab(s', \mathcal{S}, \mathcal{R}'', \mathcal{M})$, where $\mathcal{R}'' = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_{m-1}, \zeta_{m-1})\rangle$ and $s' = \Theta_m(s)$. Now clearly, $r = \Theta_i(\cdots(\Theta_{m-1}(s'))\cdots)$, where $\Theta_i(\cdots(\Theta_{m-1}(s'))\cdots)$ contains fewer than $n$ substitution applications. And by lemma 4.8, $evalid(s', \mathcal{S}, \mathcal{R}'', \mathcal{M})$ holds, therefore by the induction hypothesis, $elab(s', \mathcal{S}, \mathcal{R}'', \mathcal{M}) = elab(r, \mathcal{S}, \mathcal{R}', \mathcal{M})$. And since $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M}) = elab(s', \mathcal{S}, \mathcal{R}'', \mathcal{M})$, therefore $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M}) = elab(r, \mathcal{S}, \mathcal{R}', \mathcal{M})$, which proves claim 1.

***claim(2):*** If there does not exist a primary sort in $\Theta_1 \circ \cdots \circ \Theta_m(s)$, then $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ returns $(r, \mathcal{S})$, where $r = \Theta_1(\cdots(\Theta_m(s))\cdots)$.

**Proof** By induction on $m$.

**Basis** $(m = 0)$ If $m = 0$, and $s$ is not a primary sort, i.e. $s$ is a sort variable $r\alpha$ by definition 4.10, then by the definition of *elab*, $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M}) = (s, \mathcal{S})$.

**Induction Hypothesis** The claim holds for all $i < n$.

**Induction Step** $(m > 0)$ If there does not exist a primary sort in $\Theta_1 \circ \cdots \circ \Theta_m(s)$, then $s$ is a sort variable $r\alpha$ by definition 4.10. And therefore by the definition of *elab*, $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M}) = elab(s', \mathcal{S}, \mathcal{R}'', \mathcal{M})$, where $\mathcal{R}'' = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_{m-1}, \zeta_{m-1})\rangle$ and $s' = \Theta_m(s)$. By lemma 4.8, $evalid(s', \mathcal{S}, \mathcal{R}'', \mathcal{M})$ holds, and obviously $m - 1 < m$, therefore by the induction hypothesis, $elab(s', \mathcal{S}, \mathcal{R}'', \mathcal{M})$ returns $(\Theta_1(\cdots(\Theta_{m-1}(s'))\cdots), \mathcal{S})$. But $s' = \Theta_m(s)$, and $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M}) = elab(s', \mathcal{S}, \mathcal{R}'', \mathcal{M})$, therefore $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ returns $(\Theta_1(\cdots(\Theta_m(s))\cdots), \mathcal{S})$, which proves the claim. $\square$

## 4.2 Termination of *elab*

Proving termination of *elab* necessitates the definition of an ordering on accumulator sequences, which is shown to be well-founded on the set of accumulator sequences reachable by a given call to *elab*. Then, the accumulator sequence appearing as a formal argument to *elab* is shown to descend in this ordering, for those cases in which *elab* is called on constructed sorts. In other cases, formal arguments to *elab* are shown to descend in another well-founded ordering. The result follows.

**Definition 4.13** If $s$ is a sort variable, then $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ is called a *variable* call. If $s$ is a product sort, then $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ is called a *product* call. If $s$ is a sort constructor term, then $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ is called a *constructor* call.

**Lemma 4.14** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, there does not exist an infinite chain of recursive calls to $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that each call in the chain is a variable or product call.

**Proof** Define the ordering $\prec_l$ on (sort,accumulator sequence) pairs as follows: $((s_1, \mathcal{R}_1) \prec_l (s_2, \mathcal{R}_2)) \leftrightarrow ((length(\mathcal{R}_1) < length(\mathcal{R}_2)) \vee (length(\mathcal{R}_1) = length(\mathcal{R}_2) \wedge s_1$ is a subterm of $s_2))$. Since all sorts are finite terms, this is a lexicographic ordering. Now, suppose that

$elab(r_1^1, r_2^1, \mathcal{S}_1, \mathcal{R}_1)$ and $elab(r_1^2, r_2^2, \mathcal{S}_2, \mathcal{R}_2)$ are recursive calls to *elab*, and $elab(r_1^2, r_2^2, \mathcal{S}_2, \mathcal{R}_2)$ is an immediate recursive call of $elab(r_1^1, r_2^1, \mathcal{S}_1, \mathcal{R}_1)$. If $elab(r_1^1, r_2^1, \mathcal{S}_1, \mathcal{R}_1)$ is a variable call, then by the definition of *elab*, $\mathcal{R}_2$ is an initial sequence of $\mathcal{R}_1$, so $(r_1^2, \mathcal{R}_2) \prec_l (r_1^1, \mathcal{R}_1)$ in this case. If $elab(r_1^1, r_2^1, \mathcal{S}_1, \mathcal{R}_1)$ is a product call, then by the definition of *elab*, $r_1^2$ is a subterm of $r_1^1$ and $\mathcal{R}_2 = \mathcal{R}_1$, so $(r_1^2, \mathcal{R}_2) \prec_l (r_1^1, \mathcal{R}_1)$ in this case. Therefore, the 1st and 3rd arguments to *elab* in a chain of variable or product calls decrease lexicographically in each recursive call. Therefore each such chain must be finite, so that either the algorithm terminates, or a constructor call occurs. $\square$

The following definition gives (what will be proven to be) a strict ordering on accumulator sequences. It is fairly unwieldy, and proof of transitivity is somewhat complicated. But it is a necessary step in proving termination for *elab*.

**Definition 4.15** For all accumulator sequences $\mathcal{R}_1 = \langle(\Theta_1^1, \zeta_1^1), \ldots, (\Theta_m^1, \zeta_m^1)\rangle$ and $\mathcal{R}_2 = \langle(\Theta_1^2, \zeta_1^2), \ldots, (\Theta_n^2, \zeta_n^2)\rangle$, the relation $\prec_a$ has the following definition: $(\mathcal{R}_2 \prec_a \mathcal{R}_1) \leftrightarrow (\exists 0 < i \leq m \ . \ \forall 0 < j \leq i-1 \ . \ (\Theta_j^2, \zeta_j^2) = (\Theta_j^1, \zeta_j^1)$ and $\Theta_i^2((r\alpha_1, \ldots, r\alpha_k)\zeta_i^2)$ is a subterm of some binding in $\Theta_i^1) \vee (\forall 0 < i \leq m \ . \ (\Theta_i^2, \zeta_i^2) = (\Theta_i^1, \zeta_i^1) \wedge n > m)$, where $(r\alpha_1, \ldots, r\alpha_k)$ is the parameter sequence preceding $\zeta_i^2$ in the LHS of its declaration. Note that this definition is a disjunction, which will be relevant to the cases considered in some lemmas below.

**Lemma 4.16** The relation $\prec_a$ is irreflexive.

**Proof** Suppose $\mathcal{R}$ is a particular accumulator sequence, and let $\mathcal{R} = \langle(\Theta^1, \zeta^1), \ldots, (\Theta_j, \zeta_j)\rangle$. Then $\mathcal{R} \not\prec_a \mathcal{R}$, which is demonstrated by a consideration of two claims:

**claim *(i)*** $\neg(\forall 0 < i \leq j \ . \ (\Theta_i, \zeta_i) = (\Theta_i, \zeta_i) \wedge j > j)$. This is clearly true, since $j$ is not greater than itself.

**claim *(ii)*** $\neg\exists 0 < i \leq j \ . \ \forall 0 < j \leq i-1 \ . \ (\Theta_j, \zeta_j) = (\Theta_j, \zeta_j) \wedge \Theta_i((r\alpha_1, \ldots, r\alpha_k)\zeta_i)$ is a subterm of some binding in $\Theta_i$, where $(r\alpha_1, \ldots, r\alpha_k)$ is the parameter sequence preceding $\zeta_i$ in the LHS of its declaration. This must be the case since by definition 4.2 $\Theta_i$ has bindings only for each of $r\alpha_1, \ldots, r\alpha_k$; thus, all the bindings in $\Theta_i$ are strict and distinct subterms of $\Theta_i((r\alpha_1, \ldots, r\alpha_k)\zeta_i)$.

Thus, since both claim *(i)* and claim *(ii)* hold, $\mathcal{R} \not\prec_a \mathcal{R}$ by definition 4.15, therefore $\prec_a$ is irreflexive. $\square$

**Lemma 4.17** The relation $\prec_a$ is transitive.

To improve readability of the following proof, assumptions will be labeled *[1]*, *[2]*, ...

Suppose $\mathcal{R}_1 = \langle(\Theta_1^1, \zeta_1^1), \ldots, (\Theta_m^1, \zeta_m^1)\rangle$, $\mathcal{R}_2 = \langle(\Theta_1^2, \zeta_1^2), \ldots, (\Theta_n^2, \zeta_n^2)\rangle$ and $\mathcal{R}_3 = \langle(\Theta_1^3, \zeta_1^3), \ldots, (\Theta_z^1, \zeta_z^1)\rangle$ are accumulator sequences such that $\mathcal{R}_3 \prec_a \mathcal{R}_2$ and $\mathcal{R}_2 \prec_a \mathcal{R}_1$. Then it is necessary to demonstrate $\mathcal{R}_3 \prec_a \mathcal{R}_1$. Since $\mathcal{R}_2 \prec_a \mathcal{R}_1$ by assumption, then by definition 4.15 one of two cases must hold:

*case(i) [1]* $\exists 0 < i \leq m \ . \ \forall 0 < j \leq i-1 \ . \ (\Theta_j^2, \zeta_j^2) = (\Theta_j^1, \zeta_j^1) \wedge \Theta_i^2((r\alpha_1^2, \ldots, r\alpha_k^2)\zeta_i^2)$ is a subterm of some binding in $\Theta_i^1)$, where $(r\alpha_1^2, \ldots, r\alpha_k^2)$ is the parameter sequence preceding $\zeta_i^2$ in the LHS of its declaration.

This case can then be broken into subcases, since by assumption $\mathcal{R}_3 \prec_a \mathcal{R}_2$, so that by definition 4.15 one of two subcases must hold:

*a) [2]* $\exists 0<u\leq n \;.\; \forall 0<v\leq u-1 \;.\; (\Theta^3_v, \zeta^3_v) = (\Theta^2_v, \zeta^2_v) \wedge \Theta^3_u((r\alpha^3_1, \ldots, r\alpha^3_l)\zeta^3_u)$ is a subterm of some binding in $\Theta^2_u$), where $(r\alpha^3_1, \ldots, r\alpha^3_l)$ is the parameter sequence preceding $\zeta^3_u$ in the LHS of its declaration.

Suppose on the one hand that $u \leq i$. Then clearly $\forall 0 < v \leq u-1 \;.\; (\Theta^3_v, \zeta^3_v) = (\Theta^1_v, \zeta^1_v)$, since $\forall 0 < v \leq u-1 \;.\; (\Theta^3_v, \zeta^3_v) = (\Theta^2_v, \zeta^2_v)$ by *[2]*, and $\forall 0 < j \leq i-1 \;.\; (\Theta^2_j, \zeta^2_j) = (\Theta^1_j, \zeta^1_j)$ by *[1]*. Further, since $\Theta^3_u((r\alpha^3_1, \ldots, r\alpha^3_l)\zeta^3_u)$ is a subterm of some binding in $\Theta^2_u$ by *[2]*, then if $u < i$, $\Theta^3_u((r\alpha^3_1, \ldots, r\alpha^3_l)\zeta^3_u)$ is a subterm of some binding in $\Theta^1_u$, since if $u < i$ then $\Theta^2_u = \Theta^1_u$ by *[1]* and *[2]*. If $u = i$, then $\Theta^2_u((r\alpha^2_1, \ldots, r\alpha^2_k)\zeta^2_u)$ is a subterm of some binding in $\Theta^1_u$ by *[1]*. And since $\mathcal{R}_2$ is an accumulator sequence, therefore by definition 4.2, $\Theta^2_u$ has bindings only for each of $r\alpha^2_1, \ldots, r\alpha^2_k$. But then each binding in $\Theta^2_u$ must also be a subterm of some binding in $\Theta^1_u$, and therefore it must be the case that $\Theta^3_u((r\alpha^3_1, \ldots, r\alpha^3_l)\zeta^3_u)$ is a subterm of some binding in $\Theta^1_u$, since it is a subterm of some binding in $\Theta^2_u$.

Suppose on the other hand that $u > i$. Therefore $\forall 0 < j \leq i-1 \;.\; (\Theta^3_v, \zeta^3_v) = (\Theta^1_v, \zeta^1_v) \wedge \Theta^3_u((r\alpha^3_1, \ldots, r\alpha^3_l)\zeta^3_u)$ is a subterm of some binding in $\Theta^1_u$), since by *[2]*, $\forall 0<v\leq u-1 \;.\; (\Theta^3_v, \zeta^3_v) = (\Theta^2_v, \zeta^2_v)$, and by *[1]*, $\forall 0<j\leq i-1 \;.\; (\Theta^2_v, \zeta^2_v) = (\Theta^1_v, \zeta^1_v)) \wedge (\Theta^2_u((r\alpha^2_1, \ldots, r\alpha^2_l)\zeta^2_u)$ is a subterm of some binding in $\Theta^1_u$.

*b) [3]* $\forall 0<u\leq n \;.\; (\Theta^3_u, \zeta^3_u) = (\Theta^2_i, \zeta^2_i) \wedge z > n$.

Since $\mathcal{R}_2$ is an initial sequence of $\mathcal{R}_3$ by *[3]*, and by *[1]*, $\exists 0<i\leq m \;.\; \forall 0<j\leq i-1 \;.\; (\Theta^2_j, \zeta^2_j) = (\Theta^1_j, \zeta^1_j) \wedge \Theta^2_i((r\alpha^2_1, \ldots, r\alpha^2_k)\zeta^2_i)$ is a subterm of some binding in $\Theta^1_i$), therefore $\exists 0<i\leq m \;.\; \forall 0< j\leq i-1 \;.\; (\Theta^3_j, \zeta^3_j) = (\Theta^1_j, \zeta^1_j) \wedge (\Theta^3_i((r\alpha^3_1, \ldots, r\alpha^3_k)\zeta^3_i)$ is a subterm of some binding in $\Theta^1_i$). And thus, in both subcases of case *(i)*, $\mathcal{R}_3 \prec_a \mathcal{R}_1$.

*case (ii) [4]* $\forall 0<i\leq m \;.\; (\Theta^2_i, \zeta^2_i) = (\Theta^1_i, \zeta^1_i) \wedge n > m$.

As in the previous case, there are two subcases:

*a) [5]* $\exists 0<u\leq n \;.\; \forall 0<v\leq u-1 \;.\; (\Theta^3_v, \zeta^3_v) = (\Theta^2_v, \zeta^2_v) \wedge \Theta^3_u((r\alpha^3_1, \ldots, r\alpha^3_l)\zeta^3_u)$ is a subterm of some binding in $\Theta^2_u$), where $(r\alpha^3_1, \ldots, r\alpha^3_l)$ is the parameter sequence preceding $\zeta^3_u$ in the LHS of its declaration.

Suppose on the one hand that $u \leq m$. Since $\mathcal{R}_1$ is an initial sequence of $\mathcal{R}_2$ by *[4]*, and by *[5]*, $\forall 0 < v \leq u-1 \;.\; (\Theta^3_v, \zeta^3_v) = (\Theta^2_v, \zeta^2_v)$, therefore $\forall 0 < v \leq u-1 \;.\; (\Theta^3_v, \zeta^3_v) = (\Theta^1_v, \zeta^1_v)$; and since $\Theta^1_u = \Theta^2_u$ by *[4]*, thus $\Theta^3_u((r\alpha^3_1, \ldots, r\alpha^3_l)\zeta^3_u)$ is a subterm of some binding in $\Theta^1_u$, since it is a subterm of some binding in $\Theta^2_u$ by *[5]*. Suppose on the other hand that $u > m$, and thus $z > m$.

Since $\forall 0 < i \leq m\ .\ (\Theta^2_i, \zeta^2_i) = (\Theta^1_i, \zeta^1_i)$ by *[4]*, and $\forall 0 < v \leq u - 1\ .\ (\Theta^3_v, \zeta^3_v) = (\Theta^2_v, \zeta^2_v)$ by *[5]*, therefore $\forall 0 < u \leq m\ .\ (\Theta^3_u, \zeta^3_u) = (\Theta^1_i, \zeta^1_i) \wedge z > m$.

*b) [6]* $\forall 0 < u \leq n\ .\ (\Theta^3_u, \zeta^3_u) = (\Theta^2_i, \zeta^2_i) \wedge z > n$.

If $z > n$, then $z > m$, since $n > m$ by *[4]*. And since $\forall 0 < i \leq m\ .\ (\Theta^2_i, \zeta^2_i) = (\Theta^1_i, \zeta^1_i)$ by *[4]*, and $\forall 0 < u \leq n\ .\ (\Theta^3_u, \zeta^3_u) = (\Theta^2_i, \zeta^2_i)$ by *[6]*, therefore $\forall 0 < u \leq m\ .\ (\Theta^3_u, \zeta^3_u) = (\Theta^1_i, \zeta^1_i) \wedge z > m$.

This covers both subcase of case *(ii)*, which covers each case to be proven, therefore if $\mathcal{R}_3 \prec_a \mathcal{R}_2$ and $\mathcal{R}_2 \prec_a \mathcal{R}_1$ then $\mathcal{R}_3 \prec_a \mathcal{R}_1$, i.e. the relation $\prec_a$ is transitive. $\square$

**Definition 4.18** Two constructor calls to *elab*, say $elab_1$ and $elab_2$, are are said to be *consecutive* if without loss of generality $elab_2$ is a descendant recursive call of $elab_1$, and there does not exist another constructor call $elab_3$ such that $elab_3$ is a descendant recursive call of $elab_1$ and an ancestor of $elab_2$.

**Lemma 4.19** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, if any two recursive calls $elab(s_1, \mathcal{S}_1, \mathcal{R}_1, \mathcal{M}_1)$ and $elab(s_2, \mathcal{S}_2, \mathcal{R}_2, \mathcal{M}_2)$ of $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ are immediate recursive calls of two consecutive constructor calls, with $elab(s_2, \mathcal{S}_2, \mathcal{R}_2, \mathcal{M}_2)$ a descendant of $elab(s_1, \mathcal{S}_1, \mathcal{R}_1, \mathcal{M}_1)$, then $\mathcal{R}_2 \prec_a \mathcal{R}_1$.

**Proof** Suppose $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ is a tuple such that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds. Let $elab(s_1, \mathcal{S}_1, \mathcal{R}_1, \mathcal{M}_1)$ and $elab(s_2, \mathcal{S}_2, \mathcal{R}_2, \mathcal{M}_2)$ be immediate descendants of consecutive constructor calls, each of which is a descendent of *elab*. Let $elab(s'_2, \mathcal{S}'_2, \mathcal{R}'_2, \mathcal{M}'_2)$ be the parent call of $elab(s_2, \mathcal{S}_2, \mathcal{R}_2, \mathcal{R}_2)$, and let $\mathcal{R}_1 = \langle (\Theta_1, \zeta_1, \ldots, (\Theta_m, \zeta_m) \rangle$. Then the proof proceeds via the following claim:

***claim:*** $\mathcal{R}'_2 = \mathcal{R}_1$, or $\mathcal{R}'_2$ is a strictly initial sequence of $\mathcal{R}_1$ of length $j < m$, and $s'_2$ is a subterm of some binding in $\Theta_{j+1}$.

**proof** By induction on the number of calls between $elab(s_1, \mathcal{S}_1, \mathcal{R}_1, \mathcal{M}_1)$ and $elab(s'_2, \mathcal{S}'_2, \mathcal{R}'_2, \mathcal{M}'_2)$.

**basis** If $elab(s_1, \mathcal{S}_1, \mathcal{R}_1, \mathcal{M}_1)$ and $elab(s'_2, \mathcal{S}'_2, \mathcal{R}'_2, \mathcal{M}'_2)$ are the same recursive call, then clearly the claim holds.

**induction step** Let $elab(s_p, \mathcal{S}_p, \mathcal{R}_p, \mathcal{M}_p)$ be the parent of $elab(s'_2, \mathcal{S}'_2, \mathcal{R}'_2, \mathcal{M}'_2)$, so that $\mathcal{R}_p = \mathcal{R}_1$, or $\mathcal{R}_p$ is a strictly initial sequence of $\mathcal{R}_1$ of length $k < m$ and $s_p$ is a subterm of some binding in $\Theta_{k+1}$, by the induction hypothesis. Suppose on the one hand that $elab(s_p, \mathcal{S}_p, \mathcal{R}_p, \mathcal{M}_p)$ is a product call; then $\mathcal{R}'_2 = \mathcal{R}_p$ and $s'_2$ is a subterm of $s_p$ by the definition of *elab*, so clearly the claim holds in this case. Suppose on the other hand that $elab(s_p, \mathcal{S}_p, \mathcal{R}_p, \mathcal{M}_p)$ is a variable call, where $s_p = r\alpha_i$; then $\mathcal{R}'_2$ is a strictly initial sequence of $\mathcal{R}_p$ of length $k - 1$, and $s'_2 = \Theta_k(r\alpha_i)$, by the definition of *elab*. But since $evalid(s_p, \mathcal{S}_p, \mathcal{R}_p, \mathcal{M}_p)$ holds by lemma 4.9, therefore $r\alpha_i$ appears in the RHS of $\zeta_k$'s declaration, and so must have a binding in $\Theta_k$, by definitions 4.2, 2.14 4.5. Thus, $s'_2$ is a subterm of a binding in $\Theta_k$, and since $\mathcal{R}'_2$ is a strictly initial sequence of $\mathcal{R}_p$ of length $k - 1$, therefore $\mathcal{R}'_2$ is a strictly initial sequence of $\mathcal{R}_1$ of length $k - 1$, since $\mathcal{R}_p$ is an initial sequence of $\mathcal{R}_1$. And this covers all possible cases, so the claim holds.

Now, let $s_2' = (r_1, \ldots, r_k)\phi$, and let $\Theta' = \{r_1/r\alpha_1, \ldots, r_k/r\alpha_k\}$, where $(r\alpha_1, \ldots, r\alpha_k)$ is the parameter sequence for $\phi$ in its declaration. Then by the definition of *elab* and the above claim, $\mathcal{R}_2 = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_j, \zeta_j), (\Theta', \phi)\rangle$ for $j \leq m$. Suppose $j = m$; then clearly $\mathcal{R}_2 \prec_a \mathcal{R}_1$ by definition 4.15. Suppose on the other hand that $j < m$; then by the above claim, $(r_1, \ldots, r_k)\phi$ is a subterm of some binding in $\Theta_{j+1}$, and since $(r_1, \ldots, r_k)\phi = \Theta'((r\alpha_1, \ldots, r\alpha_k)\phi)$, therefore $\mathcal{R}_2 \prec_a \mathcal{R}_1$ in this case, by definition 4.15. And this covers all possible cases, so the proposition holds. $\square$

Having established that $\prec_a$ is a strict ordering, the following propositions establish that the set of accumulator sequences reachable from any given call to *elab* is finite, proving that $\prec_a$ is well-founded on this set.

**Definition 4.20** An accumulator sequence is said to be *reachable* by a call $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ if it can appear as the third parameter of any descendant recursive call of $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$.

**Proposition 4.21** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, there is a finite upper bound on the number of sort constructors which can appear as tags in the accumulator sequences reachable from $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$.

**Proof** Suppose that for some $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$, $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_m, \zeta_m)\rangle$. Let the set of accumulator sequences reachable from $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ be $\Upsilon$.

Let $elab(s', \mathcal{S}', \mathcal{R}', \mathcal{M}')$ be a descendant recursive call of $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $\mathcal{R}' \neq \mathcal{R}$, where $\mathcal{R}' = \langle(\Theta_1', \zeta_1'), \ldots, (\Theta_j', \zeta_j')\rangle$. By lemma 4.9, $evalid(s', \mathcal{S}', \mathcal{R}', \mathcal{M}')$ holds, therefore each of $\zeta_1', \ldots, \zeta_j'$ must be declared in $\mathcal{S}'$, by definition 4.5. Let $\mathcal{S}^-$ be the declarations which are in $\mathcal{S}'$ but not in $\mathcal{S}$. Clearly, each $(r\alpha_1, \ldots, r\alpha_k)\eta$ which is declared in $\mathcal{S}^-$ was added to the environment by a recursive call to $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$. But by the definition of *elab*, a declaration for $(r\alpha_1, \ldots, r\alpha_k)\eta$ is added to the environment in a recursive call iff a pair $(r, (r\alpha_1, \ldots, r\alpha_k)\eta)$ is added to the matching in that same recursive call. Therefore, for each $(r\alpha_1, \ldots, r\alpha_k)\eta$ in $\mathcal{S}^-$ there exists a pair $(r, (r\alpha_1, \ldots, r\alpha_k)\eta)$ in $\mathcal{M}'$. Thus, none of $\zeta_1', \ldots, \zeta_j'$ are declared in $\mathcal{S}^-$, therefore they must be declared in $\mathcal{S}$, since by definition 4.5 they are defined in $\mathcal{S}'$. And since there are a finite number of declarations in $\mathcal{S}$ by definition 4.5, therefore there are a finite number of constructors which can appear as tags in any accumulator sequence in $\Upsilon$. $\square$

**Definition 4.22** For all accumulator sequences $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_m, \zeta_m)\rangle$, the first element $(\Theta_1, \zeta_1)$ is called the *head* of $\mathcal{R}$, while the elements $(\Theta_2, \zeta_2), \ldots, (\Theta_m, \zeta_m)$ are the *tail* of $\mathcal{R}$.

**Proposition 4.23** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, there are a finite number of substitutions which can appear in the tail of any accumulator sequence reachable from $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$.

**Proof** Suppose that for some $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$, $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_m, \zeta_m)\rangle$. Let the set of accumulator sequences reachable by $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ be $\Upsilon$.

Let $elab(s', \mathcal{S}', \mathcal{R}', \mathcal{M}')$ be a descendant recursive call of $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $\mathcal{R}' \neq \mathcal{R}$, where $\mathcal{R}' = \langle(\Theta'_1, \zeta'_1), \ldots, (\Theta'_j, \zeta'_j)\rangle$. By lemma 4.9, $evalid(s', \mathcal{S}', \mathcal{R}', \mathcal{M}')$ holds, therefore by definition 4.5, for all $1 < i \leq j$ each binding in $\Theta'_i$ is a subterm of a sort appearing in the RHS of $\zeta'_{i-1}$'s declaration. And since by proposition 4.21 there are a finite number of constructors which can appear as tags in any element of $\Upsilon$, namely the constructors declared in $\mathcal{S}$, therefore there are finitely many declarations associated with these tags, and thus a finite number of sorts, and subterms of those sorts, which can appear as a binding in any $\Theta'_i$. Also, by definition 4.2, any $\Theta'_i$ has bindings only for the parameters of $\zeta'_i$ as declared in $\mathcal{S}$, so there are a finite number of such sort variable sequences which can be bound in any $\Theta'$. Therefore there are only finitely many substitutions which can appear in the tail of any accumulator sequence in $\Upsilon$. □

**Proposition 4.24** For all $(s, \mathcal{S}, \langle\rangle, \mathcal{M})$ such that $evalid(s, \mathcal{S}, \langle\rangle, \mathcal{M})$ holds, there are a finite number of substitutions which can appear in the head of any accumulator sequence reachable from $elab(s, \mathcal{S}, \langle\rangle, \mathcal{M})$.

Suppose that $evalid(s, \mathcal{S}, \langle\rangle, \mathcal{M})$ holds for some $(s, \mathcal{S}, \langle\rangle, \mathcal{M})$, and let the set of accumulator sequences reachable by $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ be $\Upsilon$. If a new head is added to the accumulator sequence in a recursive call, then clearly a constructor call of the form $elab((r_1, \ldots, r_j)\phi, \mathcal{S}', \langle\rangle, \mathcal{M}')$ must occur, since by the definition of *elab* elements are added to the accumulator sequence in a recursive call only if a constructor call occurs, and they are added to the end of the sequence.Thus, all substitutions which can appear in the head of some element of $\Upsilon$ must appear in the head of the accumulator sequence in the immediate recursive call of some call $elab((r_1, \ldots, r_j)\phi, \mathcal{S}', \langle\rangle, \mathcal{M}')$. The proof then proceeds via the following claim:

***claim:*** For all recursive constructor calls $elab((r_1, \ldots, r_j)\phi, \mathcal{S}', \langle\rangle, \mathcal{M}')$ of $elab(s, \mathcal{S}, \langle\rangle, \mathcal{M})$, all the bindings in the substitution $\Theta' = \{r_1/r\alpha_1, \ldots, r_j/r\alpha_j\}$ are subterms of $s$, where $(r\alpha_1, \ldots, r\alpha_j)$ is the parameter sequence for $\phi$ in its declaration.

**proof** By induction on $n$, the number of constructor calls occurring between $elab(s, \mathcal{S}, \langle\rangle, \mathcal{M})$ and $elab((r_1, \ldots, r_j)\phi, \mathcal{S}', \langle\rangle, \mathcal{M}')$.

**basis** Let $elab_1 = elab(s, \mathcal{S}, \langle\rangle, \mathcal{M})$ and $elab_2 = elab((r_1, \ldots, r_j)\phi, \mathcal{S}', \langle\rangle, \mathcal{M}')$. If $elab_2$ is a recursive call of $elab_1$, and no constructor calls occur between $elab_2$ and $elab_1$, then each of the calls between $elab_2$ and $elab_1$ must be product calls, since if a variable call occurs along with an empty accumulator sequence, the algorithm terminates. And therefore by the definition of *elab*, $(r_1, \ldots, r_j)\phi$ must be a subterm of $s$, since in each product call a subterm of the first parameter in each call will be passed as the first parameter to the next call. Thus, the bindings in $\{r_1/r\alpha_1, \ldots, r_j/r\alpha_j\}$ are subterms of $s$, where $(r\alpha_1, \ldots, r\alpha_j)$ is the parameter sequence for $\phi$ in its declaration.

**induction hypothesis** The claim holds for all $i < n$.

**induction step** Let $elab_1 = elab(s, \mathcal{S}, \langle\rangle, \mathcal{M})$ and $elab_2 = elab((r_1, \ldots, r_j)\phi, \mathcal{S}', \langle\rangle, \mathcal{M}')$. If $elab_2$ is a recursive call of $elab_1$, and there are $n$ constructor calls occuring between $elab_2$ and $elab_1$, then there must be some constructor call $elab_3$ which is between $elab_1$ and $elab_2$, such that $elab_3$ and $elab_2$ are consecutive. Let $elab(s_r, \mathcal{S}_r, \mathcal{R}_r, \mathcal{M}_r)$ be the immediate recursive call of $elab_3$, and let the substitution in the head of $\mathcal{R}_r$ be $\Theta_r$. Also, let $\Theta' = \{r_1/r\alpha_1, \ldots, r_j/r\alpha_j\}$, where $(r\alpha_1, \ldots, r\alpha_j)$ is the parameter sequence for $\phi$ in its declaration. By the definition of *elab*, the accumulator sequence in the immediate recursive call of $elab_2$ is $\langle(\Theta', \phi)\rangle$. And, it must be the case that each of $r_1, \ldots, r_j$ is a subterm of some binding in $\Theta_r$, for suppose on the contrary that it wasn't. Then $\Theta'((r\alpha_1, \ldots, r\alpha_j)\phi)$ cannot be a subterm of some binding in $\Theta_r$, so by definition 4.5, $\langle(\Theta', \phi)\rangle \not\prec_a \mathcal{R}_r$. But by lemma 4.19, $\langle(\Theta', \phi)\rangle \prec_a \mathcal{R}_r$, which is a contradiction, therefore each of $r_1, \ldots, r_j$ is a subterm of some binding in $\Theta_r$. But by the induction hypothesis, every binding in $\Theta_r$ is a subterm of some binding in $s$, therefore the claim holds.

Therefore, since the bindings in any substitution $\Theta'$ appearing in the head of an element of $\Upsilon$ must be a subterm of $s$, therefore there are a finite number of sorts which can be appear as bindings in some $\Theta'$. Also, by lemma 4.21, there are a finite number of tags $\phi$ which can appear in any element of $\Upsilon$, therefore there are a finite number of tags which can appear along with $\Theta'$ in the head of some element of $\Upsilon$. But since by definition 4.2, any $\Theta'$ has bindings only for the parameters of $\phi$ as declared in $\mathcal{S}$, therefore there are a finite number of such sort variable sequences which can be bound in any $\Theta'$. And therefore, there can only be finitely many $\Theta'$ which can appear in the head of an element of $\Upsilon$. $\Box$

**Proposition 4.25** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, there are a finite number of substitutions which can appear in the head of any accumulator sequence reachable from $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$.

Suppose that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds for some $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_m, \zeta_m)\rangle$, and let the set of accumulator sequences reachable by $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ be $\Upsilon$. If $m = 0$ then the result follows by the previous proposition. Otherwise, suppose $m > 0$. If a new head $(\Theta', \zeta')$ is added to the accumulator sequence in a recursive call such that $\Theta' \neq \Theta_1$, then clearly a constructor call of the form $elab((r_1, \ldots, r_j)\phi, \mathcal{S}', \langle\rangle, \mathcal{M}')$ must occur, since by the definition of *elab* elements are added to the accumulator sequence in a recursive call only if a constructor call occurs, and they are added to the end of the sequence. Thus, all substitutions which appear in the head of some element of $\Upsilon$ must occur in the head of the accumulator sequence in the immediate recursive call of some call $elab((r_1, \ldots, r_j)\phi, \mathcal{S}', \langle\rangle, \mathcal{M}')$. The proof then proceeds via the following claim:

***claim:*** For all recursive constructor calls $elab((r_1, \ldots, r_j)\phi, \mathcal{S}', \langle\rangle, \mathcal{M}')$ of $elab(s, \mathcal{S}, \langle\rangle, \mathcal{M})$, all the bindings in the substitution $\Theta' = \{r_1/r\alpha_1, \ldots, r_j/r\alpha_j\}$ are subterms of some binding in $\Theta_1$, where $(r\alpha_1, \ldots, r\alpha_j)$ is the parameter sequence for $\phi$ in its declaration.

**proof** By induction on $n$, the number of constructor calls occurring between $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ and $elab((r_1, \ldots, r_j)\phi, \mathcal{S}', \langle\rangle, \mathcal{M}')$.

**basis** Let $elab_1 = elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ and $elab_2 = elab((r_1, \ldots, r_j)\phi, \mathcal{S}', \langle\rangle, \mathcal{M}')$, and let $\Theta' = \{r_1/r\alpha_1, \ldots, r_j/r\alpha_j\}$ where $(r\alpha_1, \ldots, r\alpha_j)$ is the parameter sequence for $\phi$ in its declaration. If $elab_2$ is a recursive call of $elab_1$, and no constructor calls occur between $elab_2$ and $elab_1$, then each of the calls between $elab_2$ and $elab_1$ must be a product call or a variable call. And clearly, there must be some call $elab(\Theta_1(r\alpha), \mathcal{S}, \langle\rangle, \mathcal{M})$ between $elab_1$ and $elab_2$, where by the same reasoning in the basis of the previous proposition, each of $r_1, \ldots, r_j$ are subterms of $\Theta_1(r\alpha)$, and hence of some binding in $\Theta_1$.

**induction hypothesis** The claim holds for all $i < n$.

**induction step** Let $elab_1 = elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ and $elab_2 = elab((r_1, \ldots, r_j)\phi, \mathcal{S}', \langle\rangle, \mathcal{M}')$. If $elab_2$ is a recursive call of $elab_1$, and there are $n$ constructor calls occuring between $elab_2$ and $elab_1$, then there must be some constructor call $elab_3$ which is between $elab_1$ and $elab_2$, such that $elab_3$ and $elab_2$ are consecutive. Let $elab(s_r, \mathcal{S}_r, \mathcal{R}_r, \mathcal{M}_r)$ be the immediate recursive call of $elab_3$, and let the substitution in the head of $\mathcal{R}_r$ be $\Theta_r$. Also, let $\Theta' = \{r_1/r\alpha_1, \ldots, r_j/r\alpha_j\}$, where $(r\alpha_1, \ldots, r\alpha_j)$ is the parameter sequence for $\phi$ in its declaration. Therefore, by the same reasoning as in the induction step of the previous claim, each of $r_1, \ldots, r_j$ is a subterm of some binding in $\Theta_r$. But by the induction hypothesis, every binding in $\Theta_r$ is a subterm of some binding in $\Theta_1$, therefore the claim holds.

Thus, since the bindings in any substitution $\Theta'$ appearing in the head of an element of $\Upsilon$ must be a subterm of a some binding in $\Theta_1$, therefore there are a finite number of sorts which can appear as bindings in some $\Theta'$. Also, by lemma 4.21, there are a finite number of tags $\phi$ which can appear in any element of $\Upsilon$, therefore there are a finite number of tags which can appear along with $\Theta'$ in the head of some element of $\Upsilon$. But since by definition 4.2, any $\Theta'$ has bindings only for the parameters of $\phi$ as declared in $\mathcal{S}$, therefore there are a finite number of such sort variable sequences which can be bound in any $\Theta'$. Therefore, there can only be finitely many $\Theta'$ which can appear in the head of an element of $\Upsilon$. □

**Lemma 4.26** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, the set of accumulator sequences reachable from $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ is finite.

**Proof** Suppose that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds for some $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_m, \zeta_m)\rangle$, and let the set of reachable accumulator sequences by $\Upsilon$. By proposition 4.21, there are a finite number of tags which can appear in the elements of $\Upsilon$. And, by propositions 4.23, 4.24 and 4.25, there are a finite number of substitutions which can appear in the elements of $\Upsilon$. But then there are a finite number of pairs $(\Theta, \zeta)$ which can appear in the elements of $\Upsilon$. Let the set of all such pairs be $\Psi$.

Also, by lemma 4.7, if a constructor call $evalid((r_1, \ldots, r_j)\phi), \mathcal{S}', \mathcal{R}', \mathcal{M}')$ occurs, where $\mathcal{R}' = \langle(\Theta'_1, \zeta'_1), \ldots (\Theta'_v, \zeta'_v)\rangle$, and $\exists 0 < i \leq v \ . \ \phi = \zeta_i$ then $\Theta'_1 \circ \cdots \circ \Theta'_v((r_1, \ldots, r_j)\phi)$ is a key in $\mathcal{M}'$. But then the algorithm terminates, by the definition of $elab$. Therefore, the same constructor cannot appear as a tag in the same reachable accumulator sequence twice. Thus, there is a finite upper bound on the length of any element of $\Upsilon$, namely the number of constructors declared in $\mathcal{S}$. Let this number be $k$.

Therefore, since any accumulator sequence is a permutation of the elements of $\Psi$, of length less than or equal to $k$, there is a finite upper bound on the number of accumulator sequences in $\Upsilon$. □

Now that the groundwork has been laid, it is easy enough to give the following proof of termination for *elab*.

**Theorem 4.27** *For all* $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ *such that* $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ *holds,* $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ *terminates.*

**Proof** Suppose $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds for some $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$; then $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ terminates. For suppose on the contrary that it does not; then there must exist an infinite chain of recursive calls to $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$. And, since by lemma 4.14, any chain of variable and/or product calls must be finite, then there must exist an infinite number of constructor calls, and hence immediate recursive calls of constructor calls, in any infinite chain of recursive calls to $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$. Let $elab_1, elab_2, elab_3, \ldots$ be this chain of immediate recursive calls of constructor calls. Since by lemma 4.26 there is a finite number of reachable accumulator sequences for any call to $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$, therefore for at least one pair $elab_j$ and $elab_k$ in this chain, with $j > k$, the accumulator sequence $\mathcal{R}_j$ in $elab_j$ and the accumulator sequence $\mathcal{R}_k$ in $elab_k$ are the same. And therefore, $\mathcal{R}_j \not\prec_a \mathcal{R}_k$, since by lemma 4.16, $\prec_a$ is irreflexive. But by lemma 4.19 and transitivity of $\prec_a$, $\mathcal{R}_j \prec_a \mathcal{R}_k$, which is a contradiction. Therefore $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ terminates. $\square$

## 4.3 Preservation of the refinement type relation

It is important to show that *elab* does indeed return an intermediate sort, satisfying the preconditions given in definition 3.25. It is also important to show that the refinement type relation is preserved by *elab*; that is, that the intermediate sort returned by *elab* refines the same type as the sort which *elab* was initially called on. This section provides these necessary results. The following lemma establishes that the definitions of pre-existing sorts (i.e., sorts which are defined in the environment prior to elaboration) remain unchanged by *elab*.

**Lemma 4.28** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, if $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ returns $(s', \mathcal{S}')$, then for all sorts $r$, if $\mathcal{S} \vdash r :: \mathrm{t}$, then $\mathcal{S}' \vdash r :: \mathrm{t}$ and for all $\Theta$, $[\Theta(r)]_{\mathcal{S}} = [\Theta(r)]_{\mathcal{S}'}$.

**Proof** By well-founded induction on $n$, the number of recursive calls to *elab*.

**Induction Hypothesis** The lemma holds for all $m < n$.

Suppose that for some $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$, $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_k, \zeta_k)\rangle$, and $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ returns $(s', \mathcal{S}')$ in $n$ recursive calls.

*case(i)* If $s$ is a sort variable $r\alpha$, and $\mathcal{R} = \langle\rangle$, then by the definition of *elab*, $\mathcal{S}' = \mathcal{S}$, so clearly the lemma holds in this case.

*case(ii)* If $s$ is a sort variable $r\alpha$, then by the definition of *elab*, $(s', \mathcal{S}') = elab(\Theta_k(r\alpha), \mathcal{S}, \mathcal{R}', \mathcal{M})$, where $\mathcal{R}' = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_{k-1}, \zeta_{k-1})\rangle$. Therefore, this case holds by the induction hypothesis.

*case(iii)* If $s$ is a product sort $s_1 * \cdots * s_j$, then by the definition of *elab*, $\forall 0 < i \leq j$ . $(s'_i, \mathcal{S}_i) = elab(s_i, \mathcal{S}, \mathcal{R}, \mathcal{M})$, $\mathcal{S}' = \cup_{i=0}^{j} \mathcal{S}_i$. By the induction hypothesis, the lemma holds for each $\mathcal{S}_i$, therefore clearly the lemma holds in this case.

*case(iv)* If $s$ is a constructed sort $(s_1, \ldots, s_j)\phi$, Then by the definition of *elab*, for each value constructor term $F(g)$ in the declaration of $\phi$, the recursive call $elab(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$ is made, where $\mathcal{S}_f$ is $\mathcal{S}$ plus the declaration of some new intermediate sort constructor $\eta_f$. But then for all sorts $r$, if $\mathcal{S} \vdash r :: \mathrm{t}$, then $\mathcal{S}_f \vdash r :: \mathrm{t}$, and $\forall \Theta$ . $[\Theta(r)]_{\mathcal{S}} = [\Theta(r)]_{\mathcal{S}}$. And therefore by the induction hypothesis, for each recursive call $elab(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$ returning the environment $\mathcal{S}_g$, the lemma holds for the environment $\mathcal{S}_g$ returned by that recursive call. And $\mathcal{S}'$ is the union of all these environments, with the declaration of $\eta_f$ updated as defined by *elab*, and $\eta_f$ is not defined in $\mathcal{S}$, so the lemma holds in this case, which was the last to be proven. $\square$

**Lemma 4.29** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, if $elab(r_1, \mathcal{S}', \mathcal{R}', \mathcal{M}')$ and $elab(r_2, \mathcal{S}', \mathcal{R}', \mathcal{M}')$ are sibling recursive calls of $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ returning $(r'_1, \mathcal{S}_1)$ and $(r'_2, \mathcal{S}_2)$ respectively, then for all sorts $g$, if $\mathcal{S}_1 \vdash g :: \mathrm{t}$ then $\mathcal{S}_1 \cup \mathcal{S}_2 \vdash g :: \mathrm{t}$, and for all $\Theta$, $[\Theta(g)]_{\mathcal{S}_1} = [\Theta(g)]_{\mathcal{S}_1 \cup \mathcal{S}_2}$.

**Proof** If $\mathcal{S} \vdash g$, then the lemma follows follows by the previous lemma. If $\neg\mathcal{S} \vdash g$, then by the definition of *elab*, the definition for $g$ is added by some recursive call to $elab(r_1, \mathcal{S}', \mathcal{R}', \mathcal{M}')$. But since the store of fresh constructor symbols is assumed to be global, then any newly defined symbols in $\mathcal{S}_1$ will not be redefined or altered by any recursive call to $elab(r_2, \mathcal{S}', \mathcal{R}', \mathcal{M}')$. $\square$

**Definition 4.30** For all matchings $\mathcal{M}$ and environments $\mathcal{S}$, $erefi(\mathcal{M}, \mathcal{S})$ holds iff for all $(s_1, s_2) \in \mathcal{M}$, $\mathcal{S} \vdash s_1 :: \mathrm{t}$ and $\mathcal{S} \vdash s_2 :: \mathrm{t}$.

**Lemma 4.31** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ and $erefi(\mathcal{S}, \mathcal{M})$ hold, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_m, \zeta_m)\rangle$, if $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ returns $(s', \mathcal{S}')$ and $\mathcal{S} \vdash \Theta_1 \circ \cdots \circ \Theta_m(s) :: \mathrm{t}$, then $s'$ is an intermediate sort and $\mathcal{S}' \vdash s' :: \mathrm{t}$.

**Proof** By well-founded induction on $n$, the number of calls to *elab*.

Suppose that for some $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$, $evalid(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_m, \zeta_m)\rangle$, and suppose $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ returns $(s', \mathcal{S}')$ in $n$ recursive calls. Let $r = \Theta_1 \circ \cdots \circ \Theta_m(s)$, and suppose $\mathcal{S} \vdash r :: \mathrm{t}$. Then it suffices to show that $s'$ is an intermediate sort and $\mathcal{S}' \vdash s' :: \mathrm{t}$. By definition 2.7, the proof proceeds via the following cases:

*case(i)* $r = r\alpha$, $\mathrm{t} = \alpha$. In this case, by definition 4.10 there is no primary sort in $\Theta_1 \circ \cdots \circ \Theta_m(s)$, therefore by lemma 4.12, $s' = \Theta_1(\circ(\Theta_m(s)) \cdots)$ and $\mathcal{S}' = \mathcal{S}$. Then by proposition 3.12, $s' = r = \alpha$, and therefore by definition 3.26 $\mathcal{S}' \vdash r\alpha :: \alpha$, i.e. $\mathcal{S}' \vdash s' :: \mathrm{t}$, and clearly $s'$ is an intermediate sort in this case, by definition 3.25.

*case(ii)* $r = r_1 * \cdots * r_j$, $\mathrm{t} = \mathrm{t}_1 * \cdots \mathrm{t}_j$. In this case, by definition 4.10 there is a primary sort $r' = r'_1 * \cdots * r'_j$ in $\Theta_1 \circ \cdots \circ \Theta_m(s)$, and by lemma 4.12 $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M}) = elab(r', \mathcal{S}, \mathcal{R}', \mathcal{M})$, where $\mathcal{R}' = \langle(\Theta_1, \zeta_1, \ldots, (\Theta_{i-1}, \zeta_{i-1})\rangle$ for some $i \leq m$. Thus, $elab(r', \mathcal{S}, \mathcal{R}', \mathcal{M})$ returns $(s', \mathcal{S}')$, and since $r = \Theta_1 \circ \cdots \circ \Theta_{i-1}(r')$ by definition 4.10, therefore $\forall 0 < i \leq j$ . $\mathcal{S} \vdash \Theta_1 \circ \cdots \circ \Theta_{i-1}(r'_i) :: \mathrm{t}_i$

by definitions 3.3 and 2.7. By the definition of *elab*, $s' = s'_1 * \cdots * s'_j$, where $\forall 0 < i \leq j \,.\, (s'_i, \mathcal{S}_i) = elab(r'_i, \mathcal{S}, \mathcal{R}', \mathcal{M})$, and $\mathcal{S}' = \cup_{i=1}^{j} \mathcal{S}_i$. But for each $(r'_i, \mathcal{S}, \mathcal{R}', \mathcal{M})$, $evalid(r'_i, \mathcal{S}, \mathcal{R}', \mathcal{M}))$ holds by lemma 4.8, and $erefi(\mathcal{S}, \mathcal{M})$ holds by assumption. Therefore, since each $elab(r'_i, \mathcal{S}, \mathcal{R}', \mathcal{M})$ terminates in less than $n$ calls, $\forall 0 < i \leq j \,.\, \mathcal{S}' \vdash s'_i :: \mathrm{t}_i$ and $s'_i$ is an intermediate sort, by the induction hypothesis and proposition 4.29. Thus, by definition 3.25, $s'_1 * \cdots * s'_j$ is an intermediate sort, and by definition 3.26, $\mathcal{S}' \vdash s'_1 * \cdots * s'_j :: \mathrm{t}_1 * \cdots * \mathrm{t}_j$, so the theorem holds in this case.

*case (iii)* $r = (r_1, \ldots, r_j)\phi$, $\mathrm{t} = (\mathrm{t}_1, \ldots, \mathrm{t}_j)\mathrm{c}$. In this case, by definition 4.10 there is a primary sort $r' = (r'_1, \ldots, r'_j)\phi$ in $\Theta_1 \circ \cdots \circ \Theta_m(s)$, and by lemma 4.12 $elab(s, \mathcal{S}, \mathcal{R}, \mathcal{M}) = elab(r', \mathcal{S}, \mathcal{R}', \mathcal{M})$, where $\mathcal{R}' = \langle(\Theta_1, \zeta_1, \ldots, (\Theta_{i-1}, \zeta_{i-1})\rangle$ for some $i \leq m$. Thus, $elab(r', \mathcal{S}, \mathcal{R}', \mathcal{M})$ returns $(s', \mathcal{S}')$, and $r = \Theta_1 \circ \cdots \circ \Theta_{i-1}(r')$ by definition 4.10.

Suppose on the one hand that there exists a pair $(s_1, s_2)$ in $\mathcal{M}$ such that $s_1 = r$. Then by the definition of *elab*, $s' = s_2$ and $\mathcal{S}' = \mathcal{S}$, and since $erefi(\mathcal{S}, \mathcal{M})$ holds by assumption, therefore by definition 4.30 the lemma holds. Suppose on the other hand that $r$ is not a key in $\mathcal{M}$. Then by the definition of *elab*, $s' = (r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f$, where $(r\alpha_1^f, \ldots, r\alpha_k^f)$ are the variables in $r$, and the declaration $(r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \Omega$ is computed by *elab*. Let:

$$
\begin{aligned}
&\mathcal{S} \vdash ((r\alpha_1, \ldots, r\alpha_j)\phi = \Gamma) :: ((\alpha_1, \ldots, \alpha_j)\mathrm{c} = \mathrm{G}) \\
&\Theta' = \{r_1/r\alpha, \ldots, r_j/r\alpha_j\} \\
&\Xi = \{\mathrm{t}_1/\alpha, \ldots, \mathrm{t}_j/\alpha_j\} \\
&\Theta_f = \{r'_1/r\alpha, \ldots, r'_j/r\alpha_j\} \\
&\mathcal{R}_f = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_{i-1}, \zeta_{i-1}), (\Theta_f, \phi)\rangle \\
&\mathcal{S}_f = \mathcal{S} \cup \left\{(r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \circ\right\} \\
&\mathcal{M}_f = \mathcal{M} \cup \{(r, s')\}
\end{aligned}
$$

Then by the definition of *elab*, for each value constructor term $F(g') \in \Omega$, $(g', \mathcal{S}_{g'}) = elab(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$, where there exists a value contructor term $F(g) \in \Gamma$, and hence a term $F(\mathrm{g}) \in \mathrm{G}$ such that $\mathcal{S} \vdash g :: \mathrm{g}$ by definition 2.14. Thus, since $\forall 0 < i \leq j \,.\, \mathcal{S} \vdash r_i :: \mathrm{t}_i$, therefore $\mathcal{S} \vdash \Theta' :: \Xi$ by definition 3.6, and therefore $\mathcal{S} \vdash \Theta'(g) :: \Xi(\mathrm{g})$, by proposition 3.7.

Clearly, since $(r\alpha_1^f, \ldots, r\alpha_k^f)$ are the variables in $r$, and $r :: \mathrm{t}$, therefore $(\alpha_1^f, \ldots, \alpha_k^f)$ are the variables in t. Therefore $\mathcal{S}_f \vdash ((r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \circ) :: (\Xi, (\alpha_1, \ldots, \alpha_j)\mathrm{c} = \mathrm{G})$ by definition 3.27. Thus, since $erefi(\mathcal{S}, \mathcal{M})$ holds by assumption, $erefi(\mathcal{S}_f, \mathcal{M}_f)$ holds. And $evalid(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$ holds by proposition 4.8, and $elab(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$ terminates in less than $n$ calls; and since $\Theta'(g) = \Theta_1 \circ \cdots \circ \Theta_{i-1} \circ \Theta_f(g)$ by proposition 3.13, therefore by the induction hypothesis, $\mathcal{S}_{g'} \vdash g' :: \Xi(\mathrm{g})$. And since each constant $C \in \Omega$ is in $\Gamma$, and hence in G by definition 2.14, therefore $\mathcal{S}' \vdash \Omega :: (\Xi, \mathrm{G})$ by propositions 3.27 and 4.29.

Thus, by the preceding argument, $\mathcal{S}' \vdash \Omega :: (\Xi, \mathrm{G})$, and any sort variable appearing in a value constructor term in $\Omega$ is bound by one of $r\alpha_1^f, \ldots, r\alpha_k^f$, by proposition 3.28. Therefore, $(r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \Omega$ is a well-formed declaration and $\mathcal{S}' \vdash ((r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \Omega) :: (\Xi, (\alpha_1, \ldots, \alpha_j)\mathrm{c} = \mathrm{G})$ by definition 3.27, and clearly $(r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f$ is an intermediate sort, which covers the last case to be demonstrated. $\square$

## 4.4 Soundness of *elab*

Having established the previous results, it is relatively straightforward to give the necessary soundness proof for *elab*.

**Definition 4.32** For all environments $\mathcal{S}$ and matchings $\mathcal{M}$, $\mathit{submatch}(\mathcal{M}, \mathcal{S})$ holds iff $\forall (s_1, s_2) \in \mathcal{M}\ .\ \forall \Theta\ .\ [\Theta(s_2)]_{\mathcal{S}} \subseteq [\Theta(s_1)]_{\mathcal{S}}$.

**Lemma 4.33** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $\mathit{evalid}(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ and $\mathit{submatch}(\mathcal{M}, \mathcal{S})$ hold, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_m, \zeta_m)\rangle$, if $\mathit{elab}(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ returns $(s', \mathcal{S}')$ then $\forall \Theta\ .\ [\Theta(s')]_{\mathcal{S}'} \subseteq [\Theta \circ \Theta_1 \circ \cdots \circ \Theta_m(s)]_{\mathcal{S}}$.

**Proof** By well-founded induction on $n$, the number of recursive calls to *elab*.

Suppose there is some $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that $\mathit{evalid}(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ and $\mathit{submatch}(\mathcal{M}, \mathcal{S})$ hold, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_m, \zeta_m)\rangle$, and suppose $\mathit{elab}(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ returns $(s', \mathcal{S}')$ in $n$ recursive calls. Suppose that for some particular $\Theta$ there exists a value $\mathbf{c} \in [\Theta(s')]_{\mathcal{S}'}$, and let $\Theta' = \Theta_1 \circ \cdots \circ \Theta_m$ and let $r = \Theta'(s)$. Then it suffices to show that $\mathbf{c} \in [\Theta(r)]_{\mathcal{S}}$ to demonstrate the theorem.

**Induction Hypothesis** The lemma holds for all calls to *elab* which terminate in less than $n$ recursive calls.

*case(i)* $s = r\alpha_i$ and $m = 0$ (i.e., $\mathcal{R} = \langle\rangle$). Then by the definition of *elab*, $s' = s = r\alpha_i$ and $\mathcal{S}' = \mathcal{S}$, and therefore clearly $\mathbf{c} \in [\Theta(r)]_{\mathcal{S}}$ in this case.

*case(ii)* $s = r\alpha_i$ and $m > 0$. Let $\mathcal{R}' = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_{m-1}, \zeta_{m-1})\rangle$. Then by the definition of the algorithm, $\mathit{elab}(s, \mathcal{S}, \mathcal{R}, \mathcal{M}) = \mathit{elab}(\Theta_m(r\alpha_i), \mathcal{R}', \mathcal{S}, \mathcal{M})$, so that $\mathit{elab}(\Theta_m(r\alpha_i), \mathcal{R}', \mathcal{S}, \mathcal{M})$ returns $(s', \mathcal{S}')$. By lemma 4.8, $\mathit{evalid}(\Theta_m(r\alpha_i), \mathcal{S}, \mathcal{R}', \mathcal{S})$ holds, and $\mathit{submatch}(\mathcal{M}, \mathcal{S})$ holds and $\mathit{elab}(\Theta_m(r\alpha_i), \mathcal{R}', \mathcal{S}, \mathcal{M})$ terminates in $< n$ recursive calls by assumption. Therefore, by the induction hypothesis, $[\Theta(s')]_{\mathcal{S}'} \subseteq [\circ \cdots \circ \Theta_{m-1}(\Theta_m(s))]_{\mathcal{S}}$. But by theorem 3.12, $\Theta \circ \cdots \circ \Theta_{m-1}(\Theta_m(r\alpha_i)) = \Theta \circ \cdots \circ \Theta_{m-1} \circ \Theta_m(r\alpha_i) = \Theta(r)$, and hence $\mathbf{c} \in [\Theta(r)]_{\mathcal{S}}$ in this case.

*case(iii)* $s = s_1 * \cdots * s_j$. Then by the definition of *elab*, $s' = s'_1 * \cdots * s'_n$, such that $\forall 0 < i \leq j\ .\ (s'_i, \mathcal{S}_j) = \mathit{elab}(s_i, \mathcal{S}, \mathcal{R}, \mathcal{M})$ and $\mathcal{S}' = \cup_{i=1}^{j} \mathcal{S}_j$. Therefore by theorem 3.24 and lemma 4.29, $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_j)$ where $\forall 0 < i \leq j\ .\ \mathbf{c}_i \in [\Theta(s'_i)]_{\mathcal{S}'}$. But for all $0 < i \leq j$, $\mathit{evalid}(s_i, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds by lemma 4.8, and $\mathit{submatch}(\mathcal{M}, \mathcal{R})$ holds by assumption, hence by the induction hypothesis and proposition 4.29, $\forall 0 < i \leq j\ .\ [\Theta(s'_i)]_{\mathcal{S}'} \subseteq [\Theta \circ \Theta'(s_i)]_{\mathcal{S}}$, since each $\mathit{elab}(s_i, \mathcal{S}, \mathcal{R}, \mathcal{M})$ terminates in less than $n$ recursive calls. Therefore, $\forall 0 < i \leq j\ .\ \mathbf{c}_i \in [\Theta \circ \Theta'(s_i)]_{\mathcal{S}}$, so that $\mathbf{c} \in [\Theta \circ \Theta'(s)]_{\mathcal{S}}$ by theroem 3.24, and by theorem 3.10 $\Theta \circ \Theta'(s) = \Theta(\Theta'(s))$, hence $\mathbf{c} \in [\Theta(r)]_{\mathcal{S}}$ in this case.

*case(iv)* $s = (s_1, \ldots, s_j)\phi$, where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\phi = \Gamma$. By the definition of *elab*, there are two subcases:

*a)* Suppose $\exists k \,.\, (r,k) \in \mathcal{M}$. Then by the definition of *elab*, $\mathcal{S}' = \mathcal{S}$ and $s' = k$. But *submatch*$(\mathcal{M},\mathcal{S})$ holds by assumption, therefore $\mathbf{c} \in [\Theta(s)]_{\mathcal{S}}$ by definitions 4.32 and 3.22.

*b)* Suppose $\neg\exists k \,.\, (r,k) \in \mathcal{M}$. Then by the definition of *elab*, $s' = (r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f$, where $\mathcal{S}' \vdash (r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \Omega$ and where $\eta_f$ is a fresh constructor symbol. Therefore, if $\mathbf{c} \in [\Theta((r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f)]_{\mathcal{S}'}$, then by theorem 3.24, $\mathbf{c}$ is some constant $C$, or $\mathbf{c}$ is a value constructor term $F(\mathbf{d})$. If $\mathbf{c}$ is a constant $C$, then by theorem 3.24, $C \in \Omega$. But then $C \in \Gamma$ by the definition of *elab*, so that $\Theta(s) \Rightarrow_{\mathcal{S}} C$ by definition 3.14, and hence $\mathbf{c} \in [\Theta(s)]_{\mathcal{S}}$ by definition 3.22. Suppose on the other hand that $\mathbf{c} = F(\mathbf{d})$; then by definition 3.24 there exists a value constructor term $F(g') \in \Omega$ such that $\mathbf{d} \in [\Theta(g')]_{\mathcal{S}'}$, by proposition 3.30. Let:

$$\begin{aligned}
&\Theta_f = \{s_1/r\alpha_1, \ldots, s_j/r\alpha_j\}\\
&\mathcal{M}_f = \mathcal{M} \cup \left\{(\Theta_1 \circ \cdots \circ \Theta_n(s), (r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f)\right\}\\
&\mathcal{R}_f = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_n, \zeta_n), (\Theta_f, \phi)\rangle\\
&\mathcal{S}_f = \mathcal{S} \cup \left\{(r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f = \circ\right\}
\end{aligned}$$

Then by the definition of *elab*, $(g', \mathcal{S}_g) = \mathit{elab}(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$, where $F(g) \in \Gamma$. By lemma 4.8, *evalid*$(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$ holds. And since clearly $[\Theta((r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f)]_{\mathcal{S}_f} = \emptyset$, therefore it is the case that $[\Theta((r\alpha_1^f, \ldots, r\alpha_k^f)\eta_f)]_{\mathcal{S}_f} \subseteq [\Theta(r)]_{\mathcal{S}_f}$; and $\forall (r_1, r_2) \in \mathcal{M} \,.\, [\Theta(r_2)]_{\mathcal{S}_f} \subseteq [\Theta(r_1)]_{\mathcal{S}_f}$, since *submatch*$(\mathcal{M},\mathcal{S})$ holds by assumption, and since the declarations in $\mathcal{S}$ are unaltered in $\mathcal{S}_f$. Therefore, *submatch*$(\mathcal{M}_f,\mathcal{S}_f)$ holds. And therefore by the induction hypothesis and lemma 4.29, $[\Theta(g')]_{\mathcal{S}'} \subseteq [\Theta \circ \Theta' \circ \Theta_f(g)]_{\mathcal{S}}$. Thus, if $\mathbf{d} \in [\Theta(g')]_{\mathcal{S}'}$, then $\mathbf{d} \in [\Theta \circ \Theta' \circ \Theta_f(g)]_{\mathcal{S}}$. And, by proposition 3.13, $\Theta \circ \Theta' \circ \Theta_f(g) = \{\Theta \circ \Theta'(s_1)/r\alpha_1, \ldots, \Theta \circ \Theta'(s_j)/r\alpha_j\}\,(g)$, therefore by theorem 3.24, $F(\mathbf{d}) \in [\Theta \circ \Theta'((s_1, \ldots, s_j)\phi)]_{\mathcal{S}}$, i.e. $\mathbf{c} \in [\Theta(r)]_{\mathcal{S}}$. And this covers all the possibilities for this case, which was the last to be demonstrated. $\Box$

## 4.5 Completeness and full correctness of *elab*

The necessary completeness result cannot be established as easily as soundness. First, a generalized result must be obtained for a sort of derivation called an *end-term* derivation; this result is then combined with results from previous sections to give the full correctness proof for *elab*, given in theorem 4.46 below.

**Definition 4.34** For all sort symbols $s$ and environments $\mathcal{S}$, a derivation $s \Rightarrow_{\mathcal{S}}^* t$ is a *derivation in* a set of sorts $\Psi$ iff for all $r \in \Psi$, any occurence of $r$ in $s \Rightarrow_{\mathcal{S}}^* t$ is not reduced. The derivation $s \Rightarrow_{\mathcal{S}}^* t$ is an *end-term derivation in* a set of sorts $\Psi$ if it is a derivation in $\Psi$, and $t$ cannot be reduced, so that every subterm of $t$ must be a constant, a value constructor symbol, or a sort symbol in $\Psi$. It follows that if $\Psi = \emptyset$, then $t$ is a value.

**Example 4.35** Given $\mathcal{S}$ containing the declarations $\beta = A \mid B$, $(r\alpha)\phi = F((r\alpha)\phi * \beta * r\alpha) \mid C$ and $(r\alpha)\zeta = H((r\alpha)\phi) \mid G((r\alpha)\zeta * \beta)$, the following are end-term derivations in $\{\beta\}$:

$$(\varrho)\phi \Rightarrow_{\mathcal{S}} F((\varrho)\phi * \beta * \varrho) \Rightarrow_{\mathcal{S}} F((\varrho)\phi, \beta, \varrho) \Rightarrow_{\mathcal{S}} F(C, \beta, \varrho) \Rightarrow_{\mathcal{S}} F(C, \beta, O)$$

$$(\varrho)\zeta \Rightarrow_{\mathcal{S}} G((\varrho)\zeta * \beta) \Rightarrow_{\mathcal{S}} G((\varrho)\zeta, \beta) \Rightarrow_{\mathcal{S}} G(H((\varrho)\phi), \beta) \Rightarrow_{\mathcal{S}} G(H(C), \beta)$$

**Definition 4.36** In a derivation $s \Rightarrow^*_{\mathcal{S}} u$, any sort symbol $r$ occuring in $s$ is an *initial occurence* of $r$ in $s \Rightarrow^*_{\mathcal{S}} u$, and for any derivation step $t \Rightarrow_{\mathcal{S}} t'$ in $s \Rightarrow^*_{\mathcal{S}} u$, if the step $t \Rightarrow_{\mathcal{S}} t'$ takes a sort symbol $r$ in $t$ to a term $v$ in $t'$, then this derivation step is an *initial occurence* of the sort symbols occuring in $v$ in the derivation $s \Rightarrow^*_{\mathcal{S}} u$.

**Proposition 4.37** For all sorts $s$, sets of sorts $\Psi$ and environments $\mathcal{S}$, if $s \Rightarrow^*_{\mathcal{S}} t$ is an end term derivation in $\Psi$, and the sort $r$ has no initial occurences in $s \Rightarrow^*_{\mathcal{S}} t$, then $s \Rightarrow^*_{\mathcal{S}} t$ is an end-term derivation in $\Psi \cup \{r\}$.

**Proof** This follows immediately from definition 4.34, since if $r$ has no initial occurences in $s \Rightarrow^*_{\mathcal{S}} t$, then $r$ does not appear in $s \Rightarrow^*_{\mathcal{S}} t$. $\square$

**Definition 4.38** For all sort terms $t$ and $t'$, the equivalence of $t$ and $t'$ in a matching $\mathcal{M}$, written $t \approx_{\mathcal{M}} t'$, is inductively defined as follows:

1. $t = (t_1, \ldots, t_n)$, $t' = (t'_1, \ldots, t'_n)$, and $\forall 0 < i \leq n \,.\, t_i \approx_{\mathcal{M}} t'_i$.
2. $t = F(u)$ and $t' = F(u')$, and $u \approx_{\mathcal{M}} u'$.
3. $t = C$ and $t' = C$.
4. $t$ is a sort symbol, $t'$ is a sort symbol, and $(t, t') \in \mathcal{M}$.

**Example 4.39** The following terms are equivalent in a matching $\mathcal{M} = \{((\varrho)\gamma, (\varrho)\eta_1), ((\varrho)\zeta, (\varrho)\eta_2)\}$:

$$F((\varrho)\zeta, G) \approx_{\mathcal{M}} F((\varrho)\eta_2, G)$$

$$(H((\varrho)\gamma, (\varrho)\gamma)), G) \approx_{\mathcal{M}} (H((\varrho)\eta_1, (\varrho)\eta_1)), G)$$

**Definition 4.40** For all terms $t_1$, $t_2$ such that $t_1 \approx_{\mathcal{M}} t_2$, the *corresponding occurences* of sort symbols $(s_1, s_2)$ in $(t_1, t_2)$ are defined as follows:

1. If $t_1 = s_1$ and $t_2 = s_2$ then there is one corresponding occurence of $(s_1, s_2)$ in $(t_1, t_2)$, namely $(t_1, t_2)$.
2. If $t_1 = (u_1, \ldots, u_j)$ and $t_2 = (v_1, \ldots, v_j)$, then for all $0 < i < j$, every corresponding occurence of $(s_1, s_2)$ in $(u_i, v_i)$ is a corresponding occurence of $(s_1, s_2)$ in $(t_1, t_2)$.
3. If $t_1 = F(u)$ and $t_2 = F(u')$, then every corresponding occurence of $(s_1, s_2)$ in $(u, u')$ is a corresponding occurence of $(s_1, s_2)$ in $(t_1, t_2)$.
4. Otherwise (i.e., if $t_1$ and $t_2$ are sort symbols and $t_1 \neq s_1$ and $t_2 \neq s_2$), there are no corresponding occurences of $(s_1, s_2)$ in $(t_1, t_2)$.

**Proposition 4.41** For all terms $t$ and $t'$ and matchings $\mathcal{M}$, if $t \approx_{\mathcal{M}'} t'$ and $\exists (s_1, s_2) \in \mathcal{M}$ such that $s_1$ does not occur in $t$, then $t \approx_{\mathcal{M}'} t'$, where $\mathcal{M}' = \mathcal{M} - \{(s_1, s_2)\}$.

**Proof** Since $s_1$ does not occur in $t$, then each constructor term $r_1$ which does appear in $t$, and the corresponding constructor term $r_2$ in $t'$, has a matching $(r_1, r_2)$ in $\mathcal{M} - \{(s_1, s_2)\}$, so by definition 4.38, $t \approx_{\mathcal{M}'} t'$. $\square$

**Proposition 4.42** Let $t \approx_{\mathcal{M}} t'$, and let $\mathcal{M}' = \mathcal{M} - \{(s_1, s_2)\}$. Let $u$ and $u'$ be the terms obtained by replacing each corresponding occurence $(s_1, s_2)$ in $(t, t')$ by terms $(p, p')$ such that $p \approx_{\mathcal{M}'} p'$. Then, $u \approx_{\mathcal{M}'} u'$.

**Proof** By well-founded structural induction on $t$ and $t'$.

*case(i)* If $t = (u_1, \ldots, u_n)$ and $t' = (v_1, \ldots, v_n)$, then $\forall 0 < i \leq n \,.\, u_i \approx_{\mathcal{M}} v_i'$ by definition 4.38. For all $0 < i \leq n$, let $u_i'$ and $v_i'$ be the terms obtained by replacing each corresponding occurence of $(s_1, s_2)$ in $(u_i, v_i)$ by terms $(p, p')$ such that $p \approx_{\mathcal{M}'} p'$. Then, for all $0 < i \leq n$, $u_i' \approx_{\mathcal{M}'} v_i'$ by the induction hypothesis, so $(u_1', \ldots, u_n') \approx_{\mathcal{M}'} (v_1', \ldots, v_n')$ by definition 4.38.

*case(ii)* If $t = F(u)$ and $t' = F(u')$, then $u \approx_{\mathcal{M}} u'$ by definition 4.38. Let $v$ and $v'$ be the terms obtained by replacing each corresponding occurence of $(s_1, s_2)$ in $(u, u')$ by $p$ and $p'$ such that $p \approx_{\mathcal{M}'} p'$. Therefore by the induction hypothesis, $v \approx_{\mathcal{M}'} v'$, so that by definition 4.38, $F(v) \approx_{\mathcal{M}'} F(v')$.

*case(iii)* if $(t, t') = (s_1, s_2)$, then the proposition holds by definition.

*case(iv)* Otherwise, there are no corresponding occurences of $(s_1, s_2)$ in $(t_1, t_2)$, so this case holds by proposition 4.41. $\square$

The following predicate definition will assist in the proof of completeness for *elab*; the proposition immediately following the definition establishes a necessary result, relevant to the predicate definition, for the purposes of induction in the completeness proof for *elab*.

**Definition 4.43** For all substitutions $\Theta$, sorts $s$ and matchings $\mathcal{M}$, where $\Psi$ are the keys in $\mathcal{M}$, *subfilt*$(\Theta, s, \mathcal{M})$ holds just in case if $s \notin \Psi$, then $\Theta(s) \notin \Theta(\Psi)$.

**Proposition 4.44** For all sorts $s_1$ and $s_2$, matchings $\mathcal{M}$ and substitutions $\Theta$ such that *subfilt*$(\Theta, s_1, \mathcal{M})$ holds, if for all derivations $\Theta(s_1) \Rightarrow_{\mathcal{S}}^* t_1$ there exists a derivation $\Theta(s_2) \Rightarrow_{\mathcal{S}}^* t_2$ such that $t_1 \approx_{\Theta(\mathcal{M})} t_2$, then for all substitutions $\Theta$ such that *subfilt*$(\Theta, s_1, \mathcal{M}')$ holds, for all derivations $\Theta(s_1) \Rightarrow_{\mathcal{S}}^* t_1$ there exists a derivation $\Theta(s_2) \Rightarrow_{\mathcal{S}}^* t_2$ such that $t_1 \approx_{\Theta(\mathcal{M}')} t_2$, where $\mathcal{M}' = \mathcal{M} - \{(s_1, s_2)\}$.

**Proof** This follows from definition 4.43 and induction on the length of the derivation. $\square$

Having laid the preceding groundwork, and recalling the meaning of "application set" from definition 3.5, the generalized completeness result can be given for *elab* with the following lemma.

**Lemma 4.45** For all $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ such that *evalid*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds, if *elab*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ returns $(s', \mathcal{S}')$, then for all substitutions $\Theta$ such that *subfilt*$(\Theta, s, \mathcal{M})$ holds, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_m, \zeta_m)\rangle$ and $\mathcal{M}' = \Theta(\mathcal{M})$, if there exists an end-term derivation $\Theta \circ \Theta_1 \circ \cdots \circ \Theta_m(s) \Rightarrow_{\mathcal{S}}^n t$ in the application set of $\Theta$ and the keys in $\mathcal{M}$, then there exists a derivation $\Theta(s') \Rightarrow_{\mathcal{S}'}^* t'$, such that $t' \approx_{\mathcal{M}'} t$.

**Proof** By well-founded induction on $n$, the length of the end-term derivation.

**Induction Hypothesis** The lemma holds for all $j \leq n$.

Suppose *evalid*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds for some $(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$, where $\mathcal{R} = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_m, \zeta_m)\rangle$, and let it return $(s', \mathcal{S}')$. Let $r = \Theta_1 \circ \cdots \circ \Theta_m(s)$, and for all $\Theta$ such that *subfilt*$(\Theta, s, \mathcal{M})$ holds, let $\Psi$ be the application set of $\Theta$ and the keys in $\mathcal{M}$, and let $\mathcal{M}' = \Theta(\mathcal{M})$.

Now, it either is or is not the case that there exists a primary sort in $\Theta_1 \circ \cdots \circ \Theta_m(s)$. Suppose on the one hand that there is not a primary sort in $\Theta_1 \circ \cdots \circ \Theta_m(s)$, so that $r$ is a sort variable by definition 4.10; then by lemma 4.12, *elab*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ returns $(\Theta_1(\cdots(\Theta_m(s))\cdots), \mathcal{S})$, so that $\mathcal{S}' = \mathcal{S}$ and $\Theta(r) = \Theta(s')$ by proposition 3.12, and therefore the lemma clearly holds in this case.

Suppose on the other hand that $\exists 0 < i \leq m$ . $\Theta_i \circ \cdots \circ \Theta_m(s)$ is a primary sort in $\Theta_1 \circ \cdots \circ \Theta_m(s)$. Let $r' = \Theta_i \circ \cdots \circ \Theta_m(s)$ be the primary sort of $\Theta_1 \circ \cdots \circ \Theta_m(s)$. Then by lemma 4.12, *elab*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ = *elab*$(r', \mathcal{S}, \mathcal{R}', \mathcal{M})$, where $\mathcal{R}' = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_{i-1}, \zeta_{i-1})\rangle$, so that *elab*$(r', \mathcal{S}, \mathcal{R}', \mathcal{M})$ returns $(s', \mathcal{S}')$. It must be the case that $r$ either is or is not in a key in $\mathcal{M}$, so the proof then proceeds by cases:

*case (i)* Suppose that $r$ is a key in $\mathcal{M}$. Then for all $\Theta$ such that *subfilt*$(\Theta, s, \mathcal{M})$ holds, $\Theta(r) \in \Psi$, therefore by definition 4.34, for all such $\Theta$ there exists only one end-term derivation $\Theta(r) \Rightarrow^n_{\mathcal{S}} t$ in $\Psi$, namely the identity derivation $\Theta(r) \Rightarrow^0_{\mathcal{S}} \Theta(r)$. Since *evalid*$(s, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds by assumption, any key in $\mathcal{M}$ is a constructed sort, hence $r$ is a constructed sort; therefore by definition 4.10, $r'$ is a constructor term, and thus *elab*$(r', \mathcal{S}, \mathcal{R}', \mathcal{M})$ is a constructor call. And since $\Theta_1 \circ \cdots \circ \Theta_{i-1}(r') = r$ by definition 4.10, therefore by the definition of the algorithm, *elab* returns $((r\alpha_1, \ldots, r\alpha_x)\eta_f, \mathcal{S})$, where $(r\alpha_1, \ldots, r\alpha_x)\eta_f$ is the matching for $r$ in $\mathcal{M}$. Thus, $s' = (r\alpha_1, \ldots, r\alpha_x)\eta_f$ and $\mathcal{S}' = \mathcal{S}$. And since by the definition of *elab* the parameter sequence $(r\alpha_1, \ldots, r\alpha_x)$ consists only of the sort variables in $r$, therefore $\Theta$ is a ground substitution for $s'$, since by assumption $\Theta$ is a ground substitution for $r$. And therefore $\Theta(s') \Rightarrow^0_{\mathcal{S}'} \Theta(s')$, and $\Theta(r) \approx_{\mathcal{M}'} \Theta(s')$ by definition 4.38, since $(r, s') \in \mathcal{M}$.

*case (ii)* Suppose that $r$ is not a key in $\mathcal{M}$. And suppose that for some particular $\Theta$ such that *subfilt*$(\Theta, s, \mathcal{M})$ holds, there exists an end-term derivation $\Theta(r) \Rightarrow^n_{\mathcal{S}} t$ in $\Psi$. Then to prove the theorem it is sufficient to show that $\Theta(s') \Rightarrow^*_{\mathcal{S}'} t'$, such that $t' \approx_{\mathcal{M}'} t$. Since $\Theta(r)$ is either a product sort or a constructed sort, the proof then proceeds by subcases:

*a)* $\Theta(r) = r_1 * \cdots * r_k$, so that by definition 4.10, $r'$ is a product sort. Let $r' = r'_1 * \cdots * r'_k$. Then by the definition of *elab*, $s' = s'_1 * \cdots * s'_k$ and $\mathcal{S}' = \cup^k_{i=1} \mathcal{S}_i$, where $\forall 0 < i \leq k$ . $(s'_i, \mathcal{S}_i)$ = *elab*$(r'_i, \mathcal{S}, \mathcal{R}, \mathcal{M})$, and *evalid*$(r'_i, \mathcal{S}, \mathcal{R}, \mathcal{M})$ holds by lemma 4.8. Now, since $\Theta(r) = r_1 * \cdots * r_k$, therefore $t = (t_1, \ldots, t_k)$ by definition 4.34. And by theorem 3.20, for all $0 < i \leq k$, there exists a subderivation $\Theta(r_i) \Rightarrow^{j_i}_{\mathcal{S}} t_i$ of $\Theta(r) \Rightarrow^n_{\mathcal{S}} t$, with $j_i < n$. But clearly each subderivation $\Theta(r_i) \Rightarrow^{j_i}_{\mathcal{S}} t_i$ is an end-term derivation in $\Psi$ by definition 4.34, therefore by the induction hypothesis and proposition 4.29, $\forall 0 < y \leq k$ . $\Theta(s'_y) \Rightarrow^*_{\mathcal{S}'} t'_y$, such that $t'_y \approx_{\mathcal{M}'} t_y$. Let $t' = (t'_1, \ldots, t'_k)$.

Thus, there exists a derivation $\Theta(s') \Rightarrow^*_{\mathcal{S}'} t'$ by theorem 3.20, and $t \approx_{\mathcal{M}'} t'$ by definition 4.38, so the lemma holds in this case.

*b)* $\Theta(r) = (r_1, \ldots, r_k)\phi$, so that by definition 4.10 $r'$ is a sort constructor term $(r'_1, \ldots, r'_k)\phi$. And clearly, since $r$ is not a key in $\mathcal{M}$ by assumption, therefore $\Theta(r) \notin \Psi$ by definition 4.43, since *subfilt*$(\Theta, s, \mathcal{M})$ holds by assumption. Hence, in the call *elab*$(r', \mathcal{S}, \mathcal{R}', \mathcal{M})$, the definition of a new intermediate sort $(r\alpha^f_1, \ldots, r\alpha^f_x)\eta_f$ is computed, and $s' = (r\alpha^f_1, \ldots, r\alpha^f_x)\eta_f$ and $\mathcal{S}' \vdash (r\alpha_1, \ldots, r\alpha_x)\eta_f = \Omega$, where $\Omega$ is the definition of $\eta_f$ as computed by *elab* and $r\alpha^f_1, \ldots, r\alpha^f_x$ are the sort variables in $r$. Now, by assumption in this subcase $(r_1, \ldots, r_k)\phi \Rightarrow^n_{\mathcal{S}} t$, which is an end-term derivation in $\Psi$. The proof then proceeds via the following claim:

***claim:*** $\Theta((r\alpha_1, \ldots, r\alpha_x)\eta_f) \Rightarrow^*_{\mathcal{S}'} t'$, such that $t \approx_{\mathcal{M}'} t'$.

**proof** By induction on $z$, the number of initial occurrences of $\Theta(r)$ in $\Theta(r) \Rightarrow^n_{\mathcal{S}} t$.

Let:

$$
\begin{aligned}
&\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_k)\phi = \Gamma \\
&\Theta_f = \{r'_1/r\alpha_1, \ldots, r'_k/r\alpha_k\} \\
&\mathcal{R}_f = \langle(\Theta_1, \zeta_1), \ldots, (\Theta_{i-1}, \zeta_{i-1}), (\Theta_f, \phi)\rangle \\
&\mathcal{M}_f = \mathcal{M} \cup (r, s') \\
&\mathcal{S}_f = \mathcal{S} \cup \{(r\alpha_1, \ldots, r\alpha_x)\eta_f = \circ\}
\end{aligned}
$$

**basis** By definitions 4.34 and 3.14, $t$ is either a constant $C$ or a value constructor term $F(u)$. Suppose on the one hand that $t = C$. Then by definition 3.14, $C \in \Gamma$. But by the definition of *elab*, if $C \in \Gamma$ then $C \in \Omega$, hence by definition 3.14, $\Theta((r\alpha_1, \ldots, r\alpha_x)\eta_f) \Rightarrow_{\mathcal{S}'} C$.

Suppose on the other hand that $t = F(u)$, so that by theorem 3.20 there exists a subderivation $\Theta \circ \Theta_1 \circ \cdots \circ \Theta_{i-1} \circ \Theta_f(g) \Rightarrow^j_{\mathcal{S}} u$ of the derivation $\Theta(r) \Rightarrow^n_{\mathcal{S}} t$, where $F(g)$ is a value constructor term in $\Gamma$. And clearly, $\Theta \circ \Theta_1 \circ \cdots \circ \Theta_{i-1} \circ \Theta_f(g) \Rightarrow^j_{\mathcal{S}} u$ is an end-term derivation in $\Psi$, by definition 4.34. Let $g_r = \Theta \circ \Theta_1 \circ \cdots \circ \Theta_{i-1} \circ \Theta_f(g)$, and let $\Psi'$ be the application set of $\Theta$ and the keys in $\mathcal{M}_f$. Thus, $\Theta(g_r) \Rightarrow^j_{\mathcal{S}} u$ is an end-term derivation in $\Psi$ and, by assumption, $\Theta(r)$ does not occur in $\Theta(g_r) \Rightarrow^j_{\mathcal{S}} u$, therefore by proposition 4.37, $\Theta(g_r) \Rightarrow^j_{\mathcal{S}} u$ is an end-term derivation in $\Psi'$, where clearly $j < n$.

Now, by the definition of *elab*, there is a value constructor term $F(g') \in \Omega$, such that $(g', \mathcal{S}'_g) =$ *elab*$(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$, where *evalid*$(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$ holds by lemma 4.8. Let $\mathcal{M}'_f = \Theta(\mathcal{M}_f)$. Therefore, by the induction hypothesis of the current lemma and propositions 4.44 and 4.29, there exists a derivation $\Theta(g') \Rightarrow^*_{\mathcal{S}'} u'$ such that $u \approx_{\mathcal{M}'_f} u'$. But $\Theta(r)$ does not occur in $u$ by definition 3.19, since $\Theta(r)$ does not occur in $\Theta(g_r) \Rightarrow^j_{\mathcal{S}} u$ by assumption, therefore by proposition 4.41, $u \approx_{\mathcal{M}'} u'$. And therefore by proposition 3.30, $\Theta((r\alpha_1, \ldots, r\alpha_x)\eta_f) \Rightarrow^*_{\mathcal{S}'} F(u')$, and $F(u') \approx_{\mathcal{M}'} F(u)$ by the preceding and definition 4.38, therefore the basis holds.

**induction hypothesis** For all end-term derivations $\Theta(r) \Rightarrow_{\mathcal{S}}^{m} v$ with $m < n$ and fewer than $z$ initial occurences of $\Theta(r)$, there exists a derivation $\Theta(s') \Rightarrow_{\mathcal{S}'}^{*} v'$ such that $v \approx_{\mathcal{M}'} v'$.

**induction step** By definition 4.34, since $\Theta(r) \notin \Psi$, and since by assumption in this case $\Theta(r)$ has more than one initial occurence in $\Theta(r) \Rightarrow_{\mathcal{S}}^{n} t$, therefore $t$ must be a value constructor term $F(u)$. Therefore, by definition 4.34 and proposition 3.13 there exists a value constructor term $F(g) \in \Gamma$ such that $\Theta \circ \Theta_1 \circ \cdots \circ \Theta_{i-1} \circ \Theta_f(g) \Rightarrow_{\mathcal{S}}^{j} u$, which is an end-term derivation in $\Psi$. Let $g_r = \Theta_1 \circ \Theta_{i-1} \circ \Theta_f(g)$. Thus, $\Theta(g_r) \Rightarrow_{\mathcal{S}}^{j} u$ is an end-term derivation in $\Psi$ and, by assumption, $\Theta(r)$ has at least one initial occurence in the derivation $\Theta(g_r) \Rightarrow_{\mathcal{S}}^{j} u$.

Let $\Theta(g_r) \Rightarrow_{\mathcal{S}}^{m} w$ be the derivation which is identical to $\Theta(g_r) \Rightarrow_{\mathcal{S}}^{j} u$, except that each initial occurence of $\Theta(r)$ is not reduced. Let $\Psi'$ be the application set of $\Theta$ and the keys in $\mathcal{M}_f$. Then clearly, $\Theta(g_r) \Rightarrow_{\mathcal{S}}^{j} w$ is an end-term derivation in $\Psi'$, and $m < n$, since $j < n$ by assumption. Now, by the definition of *elab*, there is a value constructor term $F(g') \in \Omega$, such that $(g', \mathcal{S}'_g) = elab(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$. Let $\mathcal{M}'_f = \Theta(\mathcal{M}_f)$. Since $evalid(g, \mathcal{S}_f, \mathcal{R}_f, \mathcal{M}_f)$ holds by lemma 4.8, therefore by propositions 4.29 and 4.44 and the induction hypothesis of the current lemma, there exists a derivation $\Theta(g') \Rightarrow_{\mathcal{S}'}^{*} w'$ such that $w \approx_{\mathcal{M}'_f} w'$.

Furthermore, each initial occurence of $\Theta(r)$ in $\Theta(g_r) \Rightarrow_{\mathcal{S}}^{j} u$ must be reduced to some term $p$, a subterm of $u$, by a subderivation $\Theta(r) \Rightarrow_{\mathcal{S}}^{x} p$. But clearly $x < n$ for each such subderivation, and each contains fewer than $z$ initial occurences of $\Theta(r)$. Therefore, by the induction hypothesis of the claim, there exists a derivation $\Theta(s') \Rightarrow_{\mathcal{S}'}^{*} p'$ such that $p \approx_{\mathcal{M}'} p'$.

By replacing each occurence of $\Theta(r)$ in $w$ with the term $p$ which that same occurence is reduced to in the derivation $\Theta(g_r) \Rightarrow_{\mathcal{S}}^{j} u$, clearly the term $u$ is obtained. Let $u'$ be the term obtained by replacing each occurence of $\Theta(s')$ in $w'$ with some $p'$ such that $p \approx_{\mathcal{M}'} p'$ and $\Theta(s') \Rightarrow_{\mathcal{S}}^{*} p'$; then by proposition 4.42, $u \approx_{\mathcal{M}'} u'$. And since $\Theta(g') \Rightarrow_{\mathcal{S}'}^{*} w'$, and since for each occurence of $\Theta(s')$ in $w'$ which is replaced by a term $p'$ to obtain $u'$ there exists a derivation $\Theta(s') \Rightarrow_{\mathcal{S}'}^{*} p'$, therefore there exists a derivation $\Theta(g') \Rightarrow_{\mathcal{S}'}^{*} u'$. And therefore, by proposition 3.30 and definition 4.38, there exists a derivation $\Theta(s') \Rightarrow_{\mathcal{S}'}^{*} F(u')$ such that $F(u) \approx_{\mathcal{M}'} F(u')$, so the claim holds.

And since the claim holds, therefore this subcase holds, which was the last subcase of the last case to be demonstrated, so the theorem holds. $\square$

Finally, the full correctness proof for *elab* can be given easily with the following theorem.

**Theorem 4.46** *For all environments $s$ and sorts $\mathcal{S}$ such that $\mathcal{S} \vdash s :: \mathrm{t}$ for some type* t*, $elab(s, \mathcal{S}, \langle\rangle, \emptyset)$ terminates, leaving all pre-existing definitions unchanged, and returns $(s', \mathcal{S}')$ such that $s'$ is an intermediate sort, $\mathcal{S}' \vdash s' :: \mathrm{t}$ and for all substitutions $\Theta$, $[\Theta(s)]_{\mathcal{S}} = [\Theta(s')]_{\mathcal{S}'}$.*

**Proof** Suppose that for some $s$ and $\mathcal{S}$, $\mathcal{S} \vdash s :: \mathrm{t}$. By definition 4.5, $evalid(s, \mathcal{S}, \langle\rangle, \emptyset)$ holds, therefore by lemma 4.27, $elab(s, \mathcal{S}, \langle\rangle, \emptyset)$ terminates and returns a pair $(s', \mathcal{S}')$, leaving all pre-existing definitions unchanged by lemma 4.28. Since $erefi(\mathcal{S}, \emptyset)$ and $submatch(\emptyset, \mathcal{S})$ hold by definitions 4.30 and 4.32, therefore by lemma 4.31, $\mathcal{S}' \vdash s' :: \mathrm{t}$, and for all $\Theta$, $[\Theta(s')]_{\mathcal{S}'} \subseteq [\Theta(s)]_{\mathcal{S}}$ by lemma

4.33.

Now, suppose that for some substitution $\Theta$, a value $\mathbf{c}$ is in $[\Theta(s)]_{\mathcal{S}}$. By definitions 3.22 and 4.34, if $\mathbf{c} \in [\Theta(s)]_{\mathcal{S}}$ then there exists an end-term derivation $\Theta(s) \Rightarrow^*_{\mathcal{S}} \mathbf{c}$ in $\emptyset$, since an end-term derivation in $\emptyset$ is just a normal-form derivation of a value. Thus, since $\mathit{subfilt}(\Theta, s, \emptyset)$ holds by definition 4.43, there exists a derivation $\Theta(s) \Rightarrow^*_{\mathcal{S}'} \mathbf{c}$ by lemma 4.45, therefore $\mathbf{c} \in [\Theta(s')]_{\mathcal{S}'}$ by definition 3.22. And therefore, $[\Theta(s)]_{\mathcal{S}} \subseteq [\Theta(s')]_{\mathcal{S}'}$. Thus, $[\Theta(s)]_{\mathcal{S}} = [\Theta(s')]_{\mathcal{S}'}$, so the theorem holds. $\square$

# 5 Emptiness of sorts

Having established the desired result for *elab*, we now turn to the first of the decision problems to be addressed, namely sort emptiness, or whether a sort corresponds to an empty set of values. Again, since we're concerned with parametric sorts, emptiness is defined in terms of ground substitutions– that is, a given sort $s$ as defined in an environment $\mathcal{S}$ is empty iff for all ground substitutions $\Theta$, $[\Theta(s)]_{\mathcal{S}} = \emptyset$. The *emptysort* algorithm given in figure 2 should return *true* if this is the case, and *false* otherwise.

The real work of the computation is done by the *empty* algorithm, which is also given in figure 2, and which determines emptiness of the intermediate sort yielded by a call to *elab*. The use of intermediate sorts allow the *empty* algorithm to be fairly simple, and relatively easy to prove correct. Indeed, comparison will reveal that the *empty* algorithm is almost identical to the algorithm given in [DZ92] for computing emptiness of non-parametric types, the polymorphic nature of intermediate sorts notwithstanding.

The *empty* algorithm works by essentially following the different possible derivation paths from a given sort, returning *false* if some path can constitute a *simple* derivation, a term defined in section 5.2. The desired result follows.

**Example 5.1** Suppose an environment $\mathcal{S}$ contains the declarations $(r\alpha)\phi = N \mid C(r\alpha * (r\alpha)\phi)$ and $(r\alpha)\nu = C(r\alpha * (r\alpha)\phi)$ (which are essentially declarations of list and "non-empty" list sorts), and also the declaration $(r\alpha)\varepsilon = F(r\alpha * (r\alpha)\varepsilon)$. Note that for all $\Theta$, $\Theta((r\alpha)\varepsilon)$ is empty. Thus, it is also the case that for all $\Theta$, $\Theta(((r\alpha)\varepsilon)\nu)$ is empty. By the definition of the algorithm, the call $\mathit{emptysort}(((r\alpha)\varepsilon)\nu, \mathcal{S})$ first makes a call to *elab*; suppose this call returns $((r\alpha)\eta_{f_1}, \mathcal{S}')$, where $\mathcal{S}'$ contains the declarations $(r\alpha)\eta_{f_1} = C((r\alpha)\eta_{f_3} * (r\alpha)\eta_{f_2})$, $(r\alpha)\eta_{f_2} = N \mid C((r\alpha)\eta_{f_3} * (r\alpha)\eta_{f_2})$ and $(r\alpha)\eta_{f_3} = F(r\alpha * (r\alpha)\eta_{f_3})$. Thus, the call $\mathit{emptysort}(((r\alpha)\varepsilon)\nu, \mathcal{S})$ returns the result of $\mathit{empty}((r\alpha)\eta_{f_1}, \mathcal{S}', \emptyset)$:

$$\begin{aligned}
\mathit{empty}((r\alpha)\eta_{f_1}, \mathcal{S}', \emptyset) = \quad & \mathit{empty}((r\alpha)\eta_{f_3} * (r\alpha)\eta_{f_2}, \mathcal{S}', \{(r\alpha)\eta_{f_1}\}) \\
= \quad & \mathit{empty}((r\alpha)\eta_{f_3}, \mathcal{S}', \{(r\alpha)\eta_{f_1}\}) \\
& \vee\ \mathit{empty}((r\alpha)\eta_{f_2}, \mathcal{S}', \{(r\alpha)\eta_{f_1}\}) \\
= \quad & \mathit{empty}(r\alpha * (r\alpha)\eta_{f_3}, \mathcal{S}', \{(r\alpha)\eta_{f_1}, (r\alpha)\eta_{f_3}\}) \\
& \vee\ \mathit{false}
\end{aligned}$$

$emptysort(s, \mathcal{S}) =$

let $(s', \mathcal{S}') = elab(s, \mathcal{S}, \langle\rangle, \emptyset)$

return $empty(s', \mathcal{S}', \{\})$

$empty(s, \mathcal{S}, \mathcal{E}) =$

- if $s$ is a sort variable then *false*
- if $s = s_1 * \cdots * s_j$ then $\exists 0 < i \leq j \ . \ empty(s_i, \mathcal{S}, \mathcal{E})$
- if $s = (r\alpha_1, \ldots, r\alpha_j)\eta$ then:
  - if $(r\alpha_1, \ldots, r\alpha_j)\eta \in \mathcal{E}$ then *true*
  - otherwise, where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta = \Omega$, if $\exists C \in \Omega$ or $\exists F(g) \in \Omega \ . \ \neg empty(g, \mathcal{S}, \mathcal{E} \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta\})$ then *false*, else *true*

**Figure 2:** The *empty* and *emptysort* algorithms

**Example 5.1** ***(continued)***

$$\begin{aligned}
&= \ empty(r\alpha, \mathcal{S}', \{(r\alpha)\eta_{f_1}, (r\alpha)\eta_{f_3}\}) \\
&\quad \vee \ empty((r\alpha)\eta_{f_3}, \mathcal{S}', \{(r\alpha)\eta_{f_1}, (r\alpha)\eta_{f_3}\}) \\
&\quad \vee \ false \\
&= \ false \vee true \vee false \\
&= \ true
\end{aligned}$$

This is clearly the correct result.

## 5.1 Invariants and termination of *empty*

The following definition and lemma establish the necessary representation invariant for *empty*, and demonstrates that this invariant is maintained throughout computation.

**Definition 5.2** For all $(s, \mathcal{S}, \mathcal{E})$, the relation $emvalid(s, \mathcal{S}, \mathcal{E})$ holds iff the following conditions are satisfied:

1. $s$ is an intermediate sort, $\mathcal{S}$ is an environment, and $\mathcal{E}$ is a set of intermediate constructed sorts.
2. $\mathcal{S} \vdash s$, and $\forall r \in \mathcal{E} \ . \ \mathcal{S} \vdash r$.

**Lemma 5.3** For all $(s, \mathcal{S}, \mathcal{E})$ such that $emvalid(s, \mathcal{S}, \mathcal{E})$ holds, if $empty(s', \mathcal{S}, \mathcal{E}')$ is an immediate recursive call of $empty(s, \mathcal{S}, \mathcal{E})$, then $emvalid(s', \mathcal{S}, \mathcal{E}')$ holds.

**Proof** Suppose $emvalid(s, \mathcal{S}, \mathcal{E})$ holds for some $(s, \mathcal{S}, \mathcal{E})$. Then there are two cases in which a recursive call to $empty(s, \mathcal{S}, \mathcal{E})$ might occur:

*case (i)* $s$ is a product sort $s_1 * \cdots * s_j$, so that for all $0 < i \le j$ the recursive call *empty*$(s_i, \mathcal{S}, \mathcal{E})$ occurs. By definitions 3.26, for all $0 < i \le j$, $\mathcal{S} \vdash s_i$ and $s_i$ is an intermediate sort; and since *emvalid*$(s, \mathcal{S}, \mathcal{E})$ holds by assumption, therefore for all $0 < i \le j$, *emvalid*$(s_i, \mathcal{S}, \mathcal{E})$ holds.

*case (ii)* $s$ is an intermediate constructed sort $(r\alpha_1, \ldots, r\alpha_j)\eta$, and $s \notin \mathcal{E}$. Let $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta = \Omega$. Then for all $F(g) \in \Omega$, the recursive call *empty*$(g, \mathcal{S}, \mathcal{E} \cup \{s\})$ occurs. By definitions 3.27 and 3.26, $g$ is an intermediate sort and $\mathcal{S} \vdash g$, and since by assumption *emvalid*$(s, \mathcal{S}, \mathcal{E})$ holds, and thus $\mathcal{S} \vdash s$ and $\forall r \in \mathcal{E}\ .\ \mathcal{S} \vdash r$, therefore $\forall r \in \mathcal{E} \cup \{s\}\ .\ \mathcal{S} \vdash r$. Hence *emvalid*$(s', \mathcal{S}, \mathcal{E}')$ in this case. □

The following lemma establishes termination of *empty*. It is similar to the termination proof for *elab*, in that a well-founded ordering is defined, and *empty* is shown to be a well-founded recursion in this ordering.

**Lemma 5.4** For all $(s, \mathcal{S}, \mathcal{E})$ such that *emvalid*$(s, \mathcal{S}, \mathcal{E})$ holds, *empty*$(s, \mathcal{S}, \mathcal{E})$ terminates.

**Proof** Let $\Psi$ be the set of intermediate sort constructors declared in $\mathcal{S}$, and for all $\Upsilon \subseteq \Psi$ let $\overline{\Upsilon}$ be the complement of $\Upsilon$. Define the relation $\prec_s$ on (sort,$\Upsilon \subseteq \Psi$) pairs as follows: $(s_1, \Upsilon_1) \prec_s (s_2, \Upsilon_2) \leftrightarrow |\overline{\Upsilon_1}| < |\overline{\Upsilon_2}|$ or $|\overline{\Upsilon_1}| = |\overline{\Upsilon_2}|$ and $s_1$ is a subterm of $s_2$. Since all environments contain a finite number of declarations, and all sorts are finite, therefore the relation $\prec_s$ is a well-founded lexicographic ordering. The proof then proceeds via the following claim:

***claim:*** For all sets of sorts $\mathcal{E}$, intermediate sorts $s$ and environments $\mathcal{S}$ such that $\mathcal{S} \vdash s$ and $\forall r \in \mathcal{E}\ .\ \mathcal{S} \vdash r$, if there exists an immediate recursive call *empty*$(s', \mathcal{S}, \mathcal{E}')$ of *empty*$(s, \mathcal{S}, \mathcal{E})$, then $(s', \mathcal{E}') \prec_s (s, \mathcal{E})$.

**proof:** Suppose *emvalid*$(s, \mathcal{S}, \mathcal{E})$ holds for some $(s, \mathcal{S}, \mathcal{E})$. Then there are two cases in which a recursive call to *empty*$(s, \mathcal{S}, \mathcal{E})$ might occur:

*case (i)* $s$ is a product sort $s_1 * \cdots * s_j$, so that for all $0 < i \le j$ the recursive call *empty*$(s_i, \mathcal{S}, \mathcal{E})$ occurs, and clearly for all $0 < i \le j$, $(s_i, \mathcal{E}) \prec_s (s, \mathcal{E})$, therefore the claim holds in this case.

*case (ii)* $s$ is a constructed sort $(r\alpha_1, \ldots, r\alpha_j)\eta$, and $s \notin \mathcal{E}$. Let $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta = \Omega$. Then for all $F(g) \in \Omega$, the recursive call *empty*$(g, \mathcal{S}, \mathcal{E} \cup \{s\})$ occurs. But clearly, $(g, \mathcal{E} \cup \{s\}) \prec_s (s, \mathcal{E})$, which demonstrates this case, therefore the claim holds.

Thus, by the above claim, for all $(s, \mathcal{S}, \mathcal{E})$ such that *emvalid*$(s, \mathcal{S}, \mathcal{E})$ holds, for any recursive call *empty*$(s', \mathcal{S}, \mathcal{E}')$ of *empty*$(s, \mathcal{S}, \mathcal{E})$, $(s', \mathcal{E}') \prec_s (s, \mathcal{E})$, and since $\prec_s$ is a lexicographic ordering, therefore *empty* is a well-founded recursion, hence *empty*$(s, \mathcal{S}, \mathcal{E})$ terminates. □

## 5.2 Soundness of *empty*

The soundness of *empty* is addressed in this section. Also, the previously mentioned *simple* derivation is defined, as follows.

**Definition 5.5** The derivation $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}$ is a *simple derivation* in a set of sort symbols $\Psi$ iff $s \notin \Psi$ and one of the following conditions holds:

1. $\mathbf{c}$ is a constant

2. $s = s_1 * \cdots s_j$, $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_j)$, and for all $0 < i \leq j$, the subderivation $s_i \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}_i$ of $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}$ is a simple derivation in $\Psi$

3. $s = (s_1, \ldots, s_j)\zeta$, $\mathbf{c} = F(\mathbf{d})$, and where $(s_1, \ldots, s_j)\zeta \Rightarrow_{\mathcal{S}} F(g)$ is the first derivation step in $s \Rightarrow_{\mathcal{S}}^{*} t$, the subderivation $g \Rightarrow_{\mathcal{S}}^{*} \mathbf{d}$ of $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}$ is a simple derivation in $\Psi \cup \{s\}$.

Note that this is a recursive definition.

The following two propositions demonstrate some characteristics of simple derivations, and will be helpful in later proofs.

**Proposition 5.6** For all environments $\mathcal{S}$, sorts $s$ and terms $t$, if $s \Rightarrow_{\mathcal{S}}^{*} t$ is a simple derivation in $\Psi$, and the sort symbol $r$ does not occur in $s \Rightarrow_{\mathcal{S}}^{*} t$, then $s \Rightarrow_{\mathcal{S}}^{*} t$ is a simple derivation in $\Psi \cup \{r\}$.

**Proof** This clearly holds by definition 5.5. $\square$

**Proposition 5.7** For all environments $\mathcal{S}$, sorts $s$ and terms $t$, if $s \Rightarrow_{\mathcal{S}}^{*} t$ is a simple derivation in $\Psi$, then every subderivation $s' \Rightarrow_{\mathcal{S}}^{*} t'$ of $s \Rightarrow_{\mathcal{S}}^{*} t$ is a simple derivation in $\Psi$.

**Proof** By well-founded induction on $n$, the length of the derivation $s \Rightarrow_{\mathcal{S}}^{*} t$.

**Induction Hypothesis** The proposition holds for all $k < n$.

*case(i)* $s = s_1 * \cdots * s_j$, so that $t = (t_1, \ldots, t_j)$ by definition 3.14. By definition 5.5, for all $0 < i \leq j$, the subderivation $s \Rightarrow_{\mathcal{S}}^{k_i} t_i$ is a simple derivation in $\Psi$, with $k_i < n$. Therefore by the induction hypothesis, every subderivation of $s \Rightarrow_{\mathcal{S}}^{k_i} t_i$ is a simple derivation in $\Psi$. But taken together, these comprise the subderivations in $s \Rightarrow_{\mathcal{S}}^{*} t$, so the proposition holds in this case.

*case(ii)* $s = (s_1, \ldots, s_j)\zeta$, so that $t$ is either a constant $C$ or a value constructor term $F(u)$, by definition 3.14. If $t = C$, then the only subderivation of $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}$ is itself, so the proposition holds in this case. If $t = F(u)$, then by definition 5.5, where $(s_1, \ldots, s_j)\zeta \Rightarrow_{\mathcal{S}} F(g)$ is the first step in the derivation $(s_1, \ldots, s_j)\zeta \Rightarrow_{\mathcal{S}}^{*} F(u)$, the derivation $g \Rightarrow_{\mathcal{S}}^{k} u$ is a simple derivation in $\Psi \cup \{s\}$, with $k < n$. Therefore by the induction hypothesis, every subderivation of $g \Rightarrow_{\mathcal{S}}^{k} u$ is a simple derivation in $\Psi \cup \{s\}$. But by definition 5.5, clearly any derivation in $\Psi \cup \{s\}$ is a simple derivation in $\Psi$; and since $g \Rightarrow_{\mathcal{S}}^{k} u$ and its subderivations comprise the subderivations of $(s_1, \ldots, s_j)\zeta \Rightarrow_{\mathcal{S}}^{*} F(u)$, thereofore this case holds, which was the last to be proven. $\square$

Intuitively, any derivation of a value must contain a simple derivation, since if any sort symbol $s$ appears in a derivation, then there must be some subderivation which takes a (possibly recursive) appearance of $s$ to a ground term with no recurrences of $s$ (otherwise, the derivation would be infinite). The following proposition demonstrates this formally.

**Proposition 5.8** For all environments $\mathcal{S}$ and sorts $s$, there exists a derivation $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}$ iff there exists a simple derivation $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}'$ in $\emptyset$.

Suppose on the one hand that there exists a simple derivation $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}'$ in $\emptyset$. Then by definition 5.5, $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}'$ is itself a valid derivation.

Suppose on the other hand that there exists a derivation $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}$. Then the proof proceeds by the following claim:

***claim:*** There exists a simple derivation $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}'$ in $\emptyset$.

**Proof** By well-founded induction on $n$, the length of the derivation $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}$.

**Induction Hypothesis** The proposition holds for all $k < n$.

*case(i)* $s = s_1 * \cdots * s_j$, so that $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_j)$ by definition 3.14, and for all $0 < i \leq j$, there exists a derivation $s_i \Rightarrow_{\mathcal{S}}^{k_i} \mathbf{c}_i$ where $k_i < n$, by theorem 3.20. But then by the induction hypothesis, for all $0 < i \leq j$, there exists a simple derivation $s_i \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}_i'$ in $\emptyset$, and therefore by definition 5.5, there exists a simple derivation $s_1 * \cdots * s_j \Rightarrow_{\mathcal{S}}^{*} (\mathbf{c}_1', \ldots, \mathbf{c}_j)$ in $\emptyset$, so that the claim holds in this case.

*case(ii)* $s = (s_1, \ldots, s_j)\zeta$, so that $\mathbf{c}$ is either a constant $C$ or a value constructor term $F(\mathbf{d})$, by definition 3.14. If $\mathbf{c} = C$, then $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}$ is itself a simple derivation in $\emptyset$, by definition 5.5. If $\mathbf{c} = F(\mathbf{d})$ then, where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\zeta = \Gamma$ and $\Theta = \{s_1/r\alpha_1, \ldots, s_j/r\alpha_j\}$, for some $F(g) \in \Gamma$ there exists a derivation $\Theta(g) \Rightarrow_{\mathcal{S}}^{k} \mathbf{d}$ where $k < n$, by theorem 3.20. But then by the induction hypothesis, there exists a simple derivation $\Theta(g) \Rightarrow_{\mathcal{S}}^{*} \mathbf{d}'$ in $\emptyset$.

Suppose on the one hand that $s$ does not occur in $\Theta(g) \Rightarrow_{\mathcal{S}}^{*} \mathbf{d}'$. Then by proposition 5.6, $\Theta(g) \Rightarrow_{\mathcal{S}}^{*} \mathbf{d}'$ is a simple derivation in $\{s\}$, so that by definitions 3.14 and 5.5, $s \Rightarrow_{\mathcal{S}}^{*} F(\mathbf{d}')$ is a simple derivation in $\emptyset$.

Suppose on the other hand that $s$ does occur in $\Theta(g) \Rightarrow_{\mathcal{S}}^{*} \mathbf{d}'$, and is reduced to a subterm $\mathbf{d}''$ of $\mathbf{d}'$ in $\Theta(g) \Rightarrow_{\mathcal{S}}^{*} \mathbf{d}'$. Then by proposition 5.7, there exists a simple derivation $s \Rightarrow_{\mathcal{S}}^{*} \mathbf{d}''$ in $\emptyset$.

And this covers case *(ii)*, which was the last to be proven, so the claim holds. And since the claim holds, therefore the proposition holds. □

Recalling the definition of $\varrho$ from 7, and the definition of $\Theta_\varrho$ from page 9, and also the fact that the non-empty sort $\varrho$ exists in every environment by definition 2.18, the soundnesss proof is simplified considerably by the following proposition.

**Proposition 5.9** For all intermediate sorts $s$ and environments $\mathcal{S}$, it is not the case that for all $\Theta$, $[\Theta(s)]_{\mathcal{S}} = \emptyset$ iff there exists a derivation $\Theta_\varrho(s) \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}$.

**Proof** Let there be $s$ and $\mathcal{S}$ such that $\mathcal{S} \vdash s$, and suppose on the one hand that there exists a derivation $\Theta_\varrho(s) \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}$. Then by definition 3.22, $[\Theta_\varrho(s)]_{\mathcal{S}} \neq \emptyset$, so that if there exists a derivation $\Theta_\varrho(s) \Rightarrow_{\mathcal{S}}^{*} \mathbf{c}$, then $\neg\forall\Theta \,.\, [\Theta(s)]_{\mathcal{S}} = \emptyset$.

Suppose on the other hand that $\neg\forall\Theta \,.\, [\Theta(s)]_{\mathcal{S}} = \emptyset$. Then by definition 3.22, for some $\Theta$ there exists a derivation $\Theta(s) \Rightarrow^*_{\mathcal{S}} \mathbf{c}'$. The proof then proceeds by the following claim:

***claim:*** There exists a derivation $\Theta_\varrho(s) \Rightarrow^*_{\mathcal{S}} \mathbf{c}$.

**proof** By well-founded induction on $n$, the length of the derivation $\Theta(s) \Rightarrow^*_{\mathcal{S}} \mathbf{c}'$.

**induction hypothesis** For all $k < n$, if there exists a derivation $\Theta(r) \Rightarrow^k_{\mathcal{S}} \mathbf{d}'$ then there exists a derivation $\Theta_\varrho(r) \Rightarrow^*_{\mathcal{S}} \mathbf{d}$.

*case (i)* $s$ is a sort variable $r\alpha$. But then the derivation $\Theta_\varrho(r\alpha) \Rightarrow^*_{\mathcal{S}} G$ exists, since $\Theta_\varrho(r\alpha) = \varrho$ by definition 3.9, therefore the claim holds in this case.

*case (ii)* $s$ is a product sort $s_1 * \cdots * s_j$. Then $\mathbf{c}' = (\mathbf{c}'_1, \ldots, \mathbf{c}'_j)$, and for all $0 < i \leq j$, there exists a derivation $\Theta(s_1) \Rightarrow^{k_i}_{\mathcal{S}} \mathbf{c}'_i$ with $k_i < n$, by theorem 3.20. Therefore by the induction hypothesis, for all $0 < i \leq j$, there exists a derivation $\Theta_\varrho(s_1) \Rightarrow^{k_i}_{\mathcal{S}} \mathbf{c}_i$, so that there exists a derivation $\Theta_\varrho(s) \Rightarrow^*_{\mathcal{S}} (\mathbf{c}_1, \ldots, \mathbf{c}_j)$ by theorem 3.20, thus the claim holds in this case.

*case (iii)* $s$ is an intermediate constructed sort $(r\alpha_1, \ldots, r\alpha_j)\eta$, where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta = \Omega$. Then $\mathbf{c}'$ is a constant $C \in \Omega$ or a value constructor term $F(\mathbf{d}')$, where $F(g) \in \Omega$, by definitions 3.24 and 3.22. If $\mathbf{c}' = C$, then $\Theta_\varrho(s) \Rightarrow_{\mathcal{S}} C$ by definition 3.14. If $\mathbf{c}' = F(\mathbf{d}')$, then there exists a derivation $\Theta(g) \Rightarrow^k_{\mathcal{S}} \mathbf{d}'$ with $k < n$, by proposition 3.30. Therefore by the induction hypothesis there exists a derivation $\Theta_\varrho(g) \Rightarrow^*_{\mathcal{S}} \mathbf{d}$. And therefore, by proposition 3.30, there exists the derivation $\Theta_\varrho(s) \Rightarrow^*_{\mathcal{S}} F(\mathbf{d})$. Therefore, the claim holds in this case, which was the last to be proven. And since the claim holds, therefore clearly if $\neg\forall\Theta \,.\, [\Theta(s)]_{\mathcal{S}} = \emptyset$ then there exists a derivation $\Theta_\varrho(s) \Rightarrow^*_{\mathcal{S}} \mathbf{c}$. Therefore, the proposition holds. $\square$

**Definition 5.10** A derivation $s \Rightarrow^*_{\mathcal{S}} t$ *contains* a sort symbol $r$ iff $r$ occurs in $s \Rightarrow^*_{\mathcal{S}} t$.

Given the foregoing definitions and propositions, the necessary soundness result can easily be established as follows.

**Lemma 5.11** For all $(s, \mathcal{S}, \mathcal{E})$ such that *emvalid*$(s, \mathcal{S}, \mathcal{E})$ holds and $\varrho \notin \mathcal{E}$, if *empty*$(s, \mathcal{S}, \mathcal{E})$ returns *false* then there exists a simple derivation $\Theta_\varrho(s) \Rightarrow^*_{\mathcal{S}} \mathbf{c}$ in $\Theta_\varrho(\mathcal{E})$.

**Proof** By well-founded induction on $n$, the number of calls to *empty*.

**Induction Hypothesis** The lemma holds for all $m < n$.

Suppose that for some $(s, \mathcal{S}, \mathcal{E})$ such that *emvalid*$(s, \mathcal{S}, \mathcal{E})$ holds and $\varrho \notin \mathcal{E}$, *empty*$(s, \mathcal{S}, \mathcal{E})$ returns *false*. Let $\Psi = \Theta_\varrho(\mathcal{E})$. The proof then proceeds by a consideration of those cases in which *elab* returns *false*:

*case (i)* $s$ is a sort variable $r\alpha$. If $s = r\alpha$, then $\Theta_\varrho(s) = \varrho$ by definition 3.9, so that $\Theta_\varrho(s) \Rightarrow_{\mathcal{S}} G$ is a simple derivation not containing any symbols in $\Psi$, therefore the lemma holds in this case.

*case (ii)* $s$ is a product sort $s_1 * \cdots * s_j$, and for all $0 < i \leq j$, $(s_i, \mathcal{S}, \mathcal{E})$ returns *false*. For all $0 < i \leq j$, $emvalid(s_i, \mathcal{S}, \mathcal{E})$ holds by lemma 5.3. Therefore by the induction hypothesis, for all $0 < i \leq j$ there exists a simple derivation $\Theta_\varrho(s_i) \Rightarrow_S^* \mathbf{c}_i$ in $\Psi$. But then by proposition 3.20 and definition 5.5, there exists the derivation $\Theta_\varrho(s) \Rightarrow_S^* (\mathbf{c}_i, \ldots, \mathbf{c}_j)$ which is a simple derivation in $\Psi$, so the lemma holds in this case.

*case (iii)* $s$ is an intermediate constructed sort $(r\alpha_1, \ldots, r\alpha_j)\eta$, $(r\alpha_1, \ldots, r\alpha_j)\eta \notin \mathcal{E}$, and where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta = \Omega$, there exists a constant $C \in \Omega$, or a value constructor term $F(g) \in \Omega$ such that $empty(g, \mathcal{S}, \mathcal{E} \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta\})$ returns *false*. Now clearly, $(r\alpha_1, \ldots, r\alpha_j)\eta \in \mathcal{E}$ iff $\Theta_\varrho((r\alpha_1, \ldots, r\alpha_j)\eta) \in \Psi$, so $\Theta_\varrho(s) \notin \Psi$. And, if $\exists C \in \Omega$, then by definition 5.5, $\Theta_\varrho(s) \Rightarrow_S^* C$ is a simple derivation in $\Psi$, so the lemma holds in this case.

Suppose on the other hand that there does not exist a constant $C \in \Omega$, but there does exist a value constructor term $F(g) \in \Omega$, such that $empty(g, \mathcal{S}, \mathcal{E} \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta\})$ returns *false*. By lemma 5.3, $emvalid(g, \mathcal{S}, \mathcal{E} \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta\})$ holds, and clearly $(r\alpha_1, \ldots, r\alpha_j)\eta \neq \varrho$, since there are no value constructor terms in the definition of $\varrho$. Therefore, since $empty(g, \mathcal{S}, \mathcal{E} \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta\})$ terminates in less than $n$ recursive calls, by the induction hypothesis there exists a simple derivation $\Theta_\varrho(g) \Rightarrow_S^* \mathbf{d}$ in $\Psi \cup \{\Theta_\varrho((r\alpha_1, \ldots, r\alpha_j)\eta)\}$ . But then by proposition 3.30 and definition 5.5, there exists a derivation $\Theta_\varrho((r\alpha_1, \ldots, r\alpha_j)\eta) \Rightarrow_S^* F(\mathbf{d})$ which is a simple derivation in $\Psi$; and therefore this case holds, which was the last case of the lemma to be proven. $\square$

## 5.3 Completeness of *empty*, correctness of *emptysort*

A generalized completeness result for *empty* is also obtained using simple derivations; this result is then combined with previous results to give the full correctness proof for *emptysort*, given in theorem 5.13.

**Lemma 5.12** For all $(s, \mathcal{S}, \mathcal{E})$ such that $emvalid(s, \mathcal{S}, \mathcal{E})$ holds and $\varrho \notin \mathcal{E}$, if there exists a simple derivation $\Theta_\varrho(s) \Rightarrow_S^n \mathbf{c}$ in $\Theta_\varrho(\mathcal{E})$, then $empty(s, \mathcal{S}, \mathcal{E})$ returns *false*.

**Proof** By well-founded induction on $n$, the length of the simple derivation.

**Induction Hypothesis** The lemma holds for all $m < n$.

Suppose for some $s$, $\mathcal{S}$ and $\mathcal{E}$ there exists a simple derivation $\Theta_\varrho(s) \Rightarrow_S^n \mathbf{c}$ in $\Theta_\varrho(\mathcal{E})$. Let $\Psi = \Theta_\varrho(\mathcal{E})$. The proof then proceeds by cases:

*case (i)* $s$ is a sort variable $r\alpha$, so that $\Theta_\varrho(r\alpha) = \varrho$ by definition 3.9, and therefore there exists the simple derivation $\Theta_\varrho(r\alpha) \Rightarrow_S G$, by definition 3.9. But by the definition of *empty*, in this case $empty(s, \mathcal{S}, \mathcal{E})$ returns *false*.

*case (ii)* $s = s_1 * \cdots * s_j$, so that $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_j)$, and for all $0 < i \leq j$ there exists a simple derivation $\Theta_\varrho(s_i) \Rightarrow_S^{m_i} \mathbf{c}_i$ in $\Psi$ by definition 5.5. But for all $0 < i \leq j$, $emvalid(s_i, \mathcal{S}, \mathcal{E})$ holds by lemma 5.3, and $\varrho \notin \mathcal{E}$ by assumption, and $m_i < n$ by proposition 3.20. Therefore by the induction hypothesis, for all $0 < i \leq j$, $\mathit{empty}(s_i, \mathcal{S}, \mathcal{E})$ returns *false*, so by the definition of *empty*, $\mathit{empty}(s, \mathcal{S}, \mathcal{E})$ returns *false* in this case.

*case (iii)* $s = (r\alpha_1, \ldots, r\alpha_j)\eta$, where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta = \Omega$, and $\Theta_\varrho(s) \notin \Psi$ by definition 5.5, since $\Theta_\varrho(s) \Rightarrow^*_\mathcal{S} \mathbf{c}$ is a simple derivation in $\Psi$ by assumption. In this case, $\mathbf{c}$ is a constant $C$ or a value constructor term $F(\mathbf{d})$, by definition 3.14.

Suppose on the one hand that $\mathbf{c} = C$. Since $\Theta_\varrho(s) \notin \Psi$, therefore $s \notin \mathcal{E}$, since $s$ is an intermediate sort and so clearly $\Theta_\varrho(s) \in \Psi \leftrightarrow s \in \mathcal{E}$. And since $C \in \Omega$ by definition 3.14, therefore $\mathit{empty}(s, \mathcal{S}, \mathcal{E})$ returns *false* in this case by the definition of *empty*. Suppose on the other hand that $\mathbf{c} = F(\mathbf{d})$, so that for some $F(g) \in \Omega$, there exists a simple derivation $\Theta_\varrho(g) \Rightarrow^m_\mathcal{S} \mathbf{d}$ in $\Psi \cup \{\Theta_\varrho(s)\}$, by definition 5.5. But $\mathit{emvalid}(g, \mathcal{S}, \mathcal{E} \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta\})$ holds by lemma 5.3, and surely $(r\alpha_1, \ldots, r\alpha_j)\eta \neq \varrho$, since there are no value constructor terms in $\varrho$'s definition. And since $m < n$ by proposition 3.20, therefore by the induction hypothesis and definition 3.5, $\mathit{empty}(g, \mathcal{S}, \mathcal{E} \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta\})$ returns *false*; therefore by the definition of *empty*, $\mathit{empty}(s, \mathcal{S}, \mathcal{E})$ returns *false* in this case. And this was the last case to proven, therefore the lemma holds. $\Box$

**Theorem 5.13** *For all sorts $s$ and environments $\mathcal{S}$ such that $\mathcal{S} \vdash s$, $\mathit{emptysort}(s, \mathcal{S})$ terminates, returning true iff for all $\Theta$, $[\Theta(s)]_\mathcal{S} = \emptyset$.*

**Proof** Let there be $s$ and $\mathcal{S}$ such that $\mathcal{S} \vdash s$. Then, by the definition of the algorithm, $\mathit{emptysort}(s, \mathcal{S})$ returns the result of $\mathit{empty}(s', \mathcal{S}', \emptyset)$, where $(s', \mathcal{S}')$ is the result of the call $\mathit{elab}(s, \mathcal{S}, \langle\rangle, \emptyset)$. Now, since $\mathit{evalid}(s, \mathcal{S}, \langle\rangle, \emptyset)$ holds by definition 4.5, therefore by theorem 4.46, $\mathit{elab}(s, \mathcal{S}, \langle\rangle, \emptyset)$ terminates, and $s'$ is an intermediate sort defined in $\mathcal{S}'$ and for all $\Theta$, $[\Theta(s)]_\mathcal{S} = [\Theta(s')]_{\mathcal{S}'}$.

Thus, $\mathit{emvalid}(s', \mathcal{S}', \emptyset)$ holds by definition 5.2, and clearly $\varrho \notin \emptyset$, so by lemmas 5.4, 5.11 and 5.12, $\mathit{empty}(s', \mathcal{S}', \emptyset)$ terminates, returning *false* iff there exists a simple derivation $\Theta_\varrho(s') \Rightarrow^*_{\mathcal{S}'} \mathbf{c}$ in $\emptyset$. But then by proposition 5.8, $\mathit{empty}(s', \mathcal{S}', \emptyset)$ returns *false* iff there exists a derivation $\Theta_\varrho(s') \Rightarrow^*_{\mathcal{S}'} \mathbf{c}'$. And therefore by proposition 5.9, $\mathit{empty}(s', \mathcal{S}', \emptyset)$ returns *false* iff $\neg\forall\Theta\ .\ [\Theta(s')]_{\mathcal{S}'} = \emptyset$, otherwise $\mathit{empty}(s', \mathcal{S}', \emptyset)$ returns *true* by the definition of *empty*. And therefore by contraposition, $\mathit{empty}(s', \mathcal{S}', \emptyset)$ returns *true* iff $\forall\Theta\ .\ [\Theta(s')]_{\mathcal{S}'} = \emptyset$.

But as demonstrated above, for all $\Theta$, $[\Theta(s)]_\mathcal{S} = [\Theta(s')]_{\mathcal{S}'}$, and $\mathit{emptysort}(s, \mathcal{S})$ returns the result of $\mathit{empty}(s', \mathcal{S}', \emptyset)$, therefore the theorem holds. $\Box$

# 6 Intersection of sorts

The intersection problem for sorts is addressed in this section. It is important to recall that intersection in this case is restricted to sorts which refine the same type; this will have substantial effects on the decidability of the intersection problem. For example, the refinement type relation ensures that if $s_1$ and $s_2$ refine the same type, and one of $s_1$ or $s_2$ is a sort variable $r\alpha$, then both are the same sort variable $r\alpha$. Thus, we will never need to consider intersecting a sort variable with anything other than itself.

The *intersection* algorithm given in figure 3 decides the intersection problem for sorts $s_1$ and $s_2$ by computing the definition of a (possibly) new intermediate sort which corresponds to the intersection of $s_1$ and $s_2$. Again, we are concerned with polymorphic sorts, so the desired result, more specifically, is that for all sorts $s_1$ and $s_2$ defined in an environment $\mathcal{S}$, $intersection(s_1, s_2, \mathcal{S})$ terminates and returns $(s', \mathcal{S}')$, such that for all $\Theta$, $[\Theta(s')]_{\mathcal{S}'} = [\Theta(s_1)]_{\mathcal{S}} \cap [\Theta(s_2)]_{\mathcal{S}}$. However, most of the work of the computation is actually done by the *intersect* algorithm, also given in figure 3, which intersects the elaborations of $s_1$ and $s_2$. Thus, the majority of the following proofs, which lead to the desired result for *intersection*, must deal specifically with *intersect*.

**Example 6.1** Suppose an environment $\mathcal{S}$ contains the declaration $\iota = Z \mid S(\iota)$, which corresponds to the ML sort declaration `datasort nat = ZERO | SUC of nat`; i.e., $\iota$ is a representation of the natural numbers. Suppose that $\mathcal{S}$ also contains the declaration $\rho = S(\iota)$ (the positive natural numbers), and also the mutually recursive declarations $\varepsilon = Z \mid S(o)$ and $o = S(\varepsilon)$ (evens and odds). Suppose the call $intersection(\rho, \varepsilon, \mathcal{S})$ is made. By the definition of the algorithm, $intersection(\rho, \varepsilon, \mathcal{S})$ first makes two calls to *elab*; suppose these calls return $(\eta_{f_1}, \mathcal{S}_1)$ and $(\eta_{f_2}, \mathcal{S}_2)$, where $\mathcal{S}_2$ contains the declarations $\eta_{f_4} = Z \mid S(\eta_{f_4})$, $\eta_{f_3} = S(\eta_{f_4})$, $\eta_{f_1} = Z \mid S(\eta_{f_2})$ and $\eta_{f_2} = S(\eta_{f_1})$. Then, $intersection(\rho, \epsilon, \mathcal{S})$ returns the result of $intersect(\eta_{f_1}, \eta_{f_3}, \mathcal{S}_2, \emptyset)$. The recursive call tree resulting from this initial call is represented by the following:

$$\begin{array}{c}
intersect(\eta_{f_1}, \eta_{f_3}, \mathcal{S}_2, \emptyset) \\
\downarrow \\
intersect(\eta_{f_2}, \eta_{f_4}, \mathcal{S}_2 \cup \{\eta_{f_5} = \circ\}, \{\langle(\eta_{f_1}, \eta_{f_3}), \eta_{f_5}\rangle\}) \\
\downarrow \\
intersect(\eta_{f_1}, \eta_{f_4}, \mathcal{S}_2 \cup \{\eta_{f_5} = \circ, \eta_{f_6} = \circ\}, \{\langle(\eta_{f_1}, \eta_{f_3}), \eta_{f_5}\rangle, \langle(\eta_{f_2}, \eta_{f_4}), \eta_{f_6}\rangle\}) \\
\downarrow \\
intersect(\eta_{f_2}, \eta_{f_4}, \mathcal{S}_2 \cup \{\eta_{f_5} = \circ, \eta_{f_6} = \circ, \eta_{f_7} = \circ\}, \\
\qquad \{\langle(\eta_{f_1}, \eta_{f_3}), \eta_{f_5}\rangle, \langle(\eta_{f_2}, \eta_{f_4}), \eta_{f_6}\rangle, \langle(\eta_{f_1}, \eta_{f_4}), \eta_{f_7}\rangle\})
\end{array}$$

The respective values returned by these recursive calls are:

$$\begin{array}{c}
(\eta_{f_5}, \mathcal{S}_2 \cup \{\eta_{f_5} = S(\eta_{f_6}),\ \eta_{f_6} = S(\eta_{f_7}),\ \eta_{f_7} = Z \mid S(\eta_{f_6})\}) \\
| \\
(\eta_{f_6}, \mathcal{S}_2 \cup \{\eta_{f_5} = \circ,\ \eta_{f_6} = S(\eta_{f_7}),\ \eta_{f_7} = Z \mid S(\eta_{f_6})\}) \\
| \\
(\eta_{f_7}, \mathcal{S}_2 \cup \{\eta_{f_5} = \circ,\ \eta_{f_6} = \circ,\ \eta_{f_7} = Z \mid S(\eta_{f_6})\}) \\
| \\
(\eta_{f_6}, \mathcal{S}_2 \cup \{\eta_{f_5} = \circ,\ \eta_{f_6} = \circ,\ \eta_{f_7} = \circ\})
\end{array}$$

It is easily verified that $\eta_{f_5}$ has the desired structure (i.e., corresponding to even non-zero natural numbers).

## 6.1 Invariants and termination of *intersect*

The following definition and proposition state the necessary representation invariant for *intersect*, and show that this invariant is maintained throughout computation.

$intersection(s_1, s_2, \mathcal{S}) =$

let $(s_1', \mathcal{S}_1) = elab(s_1, \mathcal{S}, \langle\rangle, \emptyset)$, $(s_2', \mathcal{S}_2) = elab(s_2, \mathcal{S}_1, \langle\rangle, \emptyset)$

return $intersect(s_1', s_2', \mathcal{S}_2, \emptyset)$

$intersect(s_1, s_2, \mathcal{S}, \mathcal{I}) =$

- if $s_1 = s_2$ then return $(s_1, \mathcal{S})$
- if $s_1 = s_1^1 * \cdots * s_j^1$ and $s_2 = s_1^2 * \cdots * s_j^2$ then $\forall 0 < i \leq j$ . let $(s_i', \mathcal{S}_i) = intersect(s_i^1, s_i^2, \mathcal{S}, \mathcal{I})$, return $(s_1' * \cdots * s_j', \cup_{i=1}^{j} \mathcal{S}_i)$
- if $s_1 = (r\alpha_1^1, \ldots, r\alpha_j^1)\eta_1$ and $s_2 = (r\alpha_1^2, \ldots, r\alpha_j^2)\eta_2$, then:
  - if $\exists s'$ . $\langle (s_1, s_2), s' \rangle \in \mathcal{I}$ then return $(s', \mathcal{S})$
  - otherwise, where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_1 = \Omega_1$ and $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_2 = \Omega_2$:

    let $\mathcal{S}_f = \mathcal{S} \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta_f = \circ\}$
    let $\Omega = C_1 \mid \cdots \mid C_x$, where for all $0 < i \leq x$, $C_i \in \Omega_1$ and $C_i \in \Omega_2$
    let $\mathcal{I}_f = \mathcal{I} \cup \{\langle (s_1, s_2), (r\alpha_1, \ldots, r\alpha_j)\eta_f \rangle\}$

    for all $F_1(g_1) \in \Omega_1$
    for all $F_2(g_2) \in \Omega_2$ such that $F_1 = F_2$
    let $(g', \mathcal{S}_{g'}) = intersect(g_1, g_2, \mathcal{S}_f, \mathcal{I}_f)$
    let $\Omega = \Omega \mid F_1(g')$

    let $\mathcal{S}' = \bigcup_{F(g') \in \Omega} \mathcal{S}_{g'} - \{(r\alpha_1, \ldots, r\alpha_j)\eta_f = \circ\}$
    let $\mathcal{S}'' = \mathcal{S} \cup \mathcal{S}' \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta_f = \Omega\}$
    return $((r\alpha_1, \ldots, r\alpha_j)\eta_f, \mathcal{S}'')$

**Figure 3:** The *intersect* and *intersection* algorithms

**Definition 6.2** A *co-matching* $\mathcal{I}$ is a set of elements of the form $\langle (s_1, s_2), s_3 \rangle$, where each of $s_1$, $s_2$ and $s_3$ are intermediate constructed sorts. For each element $\langle (s_1, s_2), s_3 \rangle \in \mathcal{I}$, $(s_1, s_2)$ is called a *key* in $\mathcal{I}$, and $s_3$ is a *matching for* $(s_1, s_2)$ in $\mathcal{I}$.

**Definition 6.3** For all $(s_1, s_2, \mathcal{S}, \mathcal{I})$, the relation $ivalid(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds iff the following conditions are satisfied:

1. $s_1$ and $s_2$ are intermediate sorts, $\mathcal{S}$ is an environment and $\mathcal{I}$ is a co-matching.
2. For some type t, $\mathcal{S} \vdash s_1 :: \mathrm{t}$ and $\mathcal{S} \vdash s_2 :: \mathrm{t}$.
3. For all $\langle (r_1, r_2), r_3 \rangle \in \mathcal{I}$, $\mathcal{S} \vdash r_1$, $\mathcal{S} \vdash r_2$, $\mathcal{S} \vdash r_3$, and each of $r_1$, $r_2$ and $r_3$ are intermediate constructed sorts.

**Lemma 6.4** For all $(s_1, s_2, \mathcal{S}, \mathcal{I})$ such that $ivalid(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds, for any immediate recursive call $intersect(s_1', s_2', \mathcal{S}', \mathcal{I}')$ of $intersect(s_1, s_2, \mathcal{S}, \mathcal{I})$, $ivalid(s_1', s_2', \mathcal{S}', \mathcal{I}')$ holds.

**Proof** Suppose there is some $(s_1, s_2, \mathcal{S}, \mathcal{I})$ such that $ivalid(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds. Then by the definition of *intersect*, there ar two cases in which a recursive call might occur:

*case (i)* $s_1 = s_1^1 * \cdots * s_j^1$, $s_2 = s_1^2 * \cdots * s_j^2$, and for all $0 < i \leq j$ the recursive call $intersect(s_i^1, s_i^2, \mathcal{S}, \mathcal{I})$ occurs. Let $\mathcal{S} \vdash s_1 :: \mathrm{t}$ and $\mathcal{S} \vdash s_2 :: \mathrm{t}$ for some $\mathrm{t} = \mathrm{t}_1 * \cdots * \mathrm{t}_j$. Then for all $0 < i \leq j$, $s_i^1$ and $s_i^2$ are intermediate sorts by definition 3.25, and $\mathcal{S} \vdash s_i^1 :: \mathrm{t}_i$ and $\mathcal{S} \vdash s_i^2 :: \mathrm{t}_i$ by definition 3.26, so conditions 1 and 2 of definition 6.3 are satisfied. And since $ivalid(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds by assumption, therefore condition 3 is satisfied by $\mathcal{I}$, hence for all $0 < i \leq j$, $ivalid(s_i^1, s_i^2, \mathcal{S}, \mathcal{I})$ holds.

*case (ii)* $s_1' = (r\alpha_1, \ldots, r\alpha_j)\eta_1$, $s_2' = (r\alpha_1, \ldots, r\alpha_j)\eta_2$, and $(s_1', s_2')$ is not a key in $\mathcal{I}'$. Suppose that $\mathcal{S} \vdash s_1 :: \mathrm{t}$ and $\mathcal{S} \vdash s_2 :: \mathrm{t}$ for some type t. Let:

$$
\begin{aligned}
&\mathrm{t} = (\mathrm{t}_1, \ldots, \mathrm{t}_k)\mathrm{c} \\
&\mathcal{S} \vdash (\alpha_1^c, \ldots, \alpha_k^c)\mathrm{c} = \mathrm{G} \\
&\Xi = \{\mathrm{t}_1/\alpha_1^c, \ldots, \mathrm{t}_k/\alpha_k^c\} \\
&\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_1 = \Omega_1 \\
&\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_2 = \Omega_2 \\
&\mathcal{S}_f = \mathcal{S} \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta_f = \circ\}, \text{ where } \eta_f \text{ is fresh} \\
&\mathcal{I}_f = \mathcal{I} \cup \{\langle (s_1, s_2), (r\alpha_1, \ldots, r\alpha_j)\eta_f \rangle\}
\end{aligned}
$$

Then for each $F_1(g_1) \in \Omega_1$ and $F_2(g_2) \in \Omega_2$ such that $F_1 = F_2$, the recursive call $intersect(g_1, g_2, \mathcal{S}_f, \mathcal{I}_f)$ occurs. By definition 3.27, $g_1$ and $g_2$ are intermediate sorts, and there exists a value constructor term $F(\mathrm{g}) \in \mathrm{G}$ such that $\mathcal{S} \vdash g_1 :: \Xi(\mathrm{g})$ and $\mathcal{S} \vdash g_2 :: \Xi(\mathrm{g})$. But then clearly $\mathcal{S}_f \vdash g_1 :: \Xi(\mathrm{g})$ and $\mathcal{S}_f \vdash g_2 :: \Xi(\mathrm{g})$, so conditions 1 and 2 of definition 6.3 are satisfied by $g_1$, $g_2$ and $\mathcal{S}_f$. And since also clearly $\mathcal{S}_f \vdash s_1$, $\mathcal{S}_f \vdash s_2$, and $\mathcal{S}_f \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_f$, each of which are intermediate constructed sorts, therefore condition 3 of definition 6.4 is satisfied by $\mathcal{I}_f$. Thus, $ivalid(g_1, g_2, \mathcal{S}_f, \mathcal{I}_f)$ holds, which demonstrates this case, which was the last to be proven. □

The following proposition will be helpful to the termination proof, given below.

**Proposition 6.5** For all $(s_1, s_2, \mathcal{S}, \mathcal{I})$ such that $ivalid(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds, if $intersect(s_1', s_2', \mathcal{S}', \mathcal{I}')$ is a descendant recursive call of $intersect(s_1, s_2, \mathcal{S}, \mathcal{I})$, then $\mathcal{S} \vdash s_1'$ and $\mathcal{S} \vdash s_2'$.

**Proof** By well-founded induction on $n$, the number of calls between $intersect(s_1, s_2, \mathcal{S}, \mathcal{I})$ and $intersect(s_1', s_2', \mathcal{S}', \mathcal{I}')$.

Suppose $ivalid(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds, and suppose $intersect(s_1', s_2', \mathcal{S}', \mathcal{I}')$ is a descendant recursive call of $intersect(s_1, s_2, \mathcal{S}, \mathcal{I})$. Let $intersect(s_1', s_2', \mathcal{S}', \mathcal{I}')$ be an immediate recursive call of $intersect(r_1, r_2, \mathcal{S}_r, \mathcal{I}_r)$, so that by the induction hypothesis, $\mathcal{S} \vdash r_1$ and $\mathcal{S} \vdash r_2$. By the definition of *intersect*, $intersect(r_1, r_2, \mathcal{S}_r, \mathcal{I}_r)$ is either a product or constructor call. In the former case, $s_1'$ and $s_2'$ are subterms of $r_1$ and $r_2$ respectively by the definition of *intersect*, so that $\mathcal{S} \vdash s_1'$ and $\mathcal{S} \vdash s_2'$ by definition 3.26. In the latter case, $s_1'$ and $s_2'$ are sorts appearing in the declarations of $r_1$ and $r_2$, so that $\mathcal{S} \vdash s_1'$ and $\mathcal{S} \vdash s_2'$ by definition 3.27. □

This next lemma gives the desired termination result for *intersect*. The strategy here is similar to the strategy employed for the *elab* and *empty* termination proofs; the algorithm is shown to be a well-founded recursion.

**Lemma 6.6** For all $(s_1, s_2, \mathcal{S}, \mathcal{I})$ such that $ivalid(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds, $intersect(s_1, s_2, \mathcal{S}, \mathcal{I})$ terminates.

**Proof** Suppose $ivalid(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds for some $(s_1, s_2, \mathcal{S}, \mathcal{I})$. Let $\Psi$ be the set of pairs $(r_1, r_2)$ of intermediate constructed sorts in $\mathcal{S}$. Since there are a finite number of declarations in $\mathcal{S}$ by assumption, therefore $\Psi$ is finite, since each $r_1$ and $r_2$ must be on the LHS of some declaration in $\mathcal{S}$ by definitions 3.25 and 6.3. Define the ordering $\prec_I$ on (sort,$\Upsilon \subseteq \Psi$) pairs as follows: $(r_1, \Upsilon_1) \prec_I (r_2, \Upsilon_2) \leftrightarrow |\Upsilon_1| > |\Upsilon_2| \vee |\Upsilon_1| = |\Upsilon_2|$ and $r_1$ is a subterm of $r_2$. Clearly, $\prec_I$ is a well-founded lexicographic ordering.

Then, suppose that $intersect(s'_1, s'_2, \mathcal{S}', \mathcal{I}')$ is a recursive call of $intersect(s_1, s_2, \mathcal{S}, \mathcal{I})$, and let $\Upsilon_1$ be the set of pairs $(r_1, r_2) \in \Psi$ such that $(r_1, r_2)$ is not a key in $\mathcal{I}'$. The proof proceeds via the following claim:

***claim:*** For any immediate recursive call $intersect(k_1, k_2, \mathcal{S}_r, \mathcal{I}_r)$ of $intersect(s'_1, s'_2, \mathcal{S}', \mathcal{I}')$, where $\Upsilon_2$ is the set of pairs $(r_1, r_2) \in \Psi$ such that $(r_1, r_2)$ is not a key in $\mathcal{I}_r$, $(k_1, \Upsilon_2) \prec_I (s'_1, \Upsilon_1)$.

**proof** There are two cases in which a recursive call to $intersect(s'_1, s'_2, \mathcal{S}', \mathcal{I}')$ might occur:

*case (i)* $s'_1 = s^1_1 * \cdots * s^1_j$, $s'_2 = s^2_1 * \cdots * s^2_j$, and for all $0 < i \leq j$ the recursive call $intersect(s^1_i, s^2_i, \mathcal{S}', \mathcal{I}')$ occurs. Thus, $\Upsilon_1 = \Upsilon_2$ in this case, so clearly, for all $0 < i \leq j$, $(s^1_i, \Upsilon_1) \prec_I (s'_1, \Upsilon_1)$, so the claim holds in this case.

*case (ii)* $s'_1 = (r\alpha_1, \ldots, r\alpha_j)\eta_1$, $s'_2 = (r\alpha_1, \ldots, r\alpha_j)\eta_2$, and $(s'_1, s'_2)$ is not a key in $\mathcal{I}'$. In this case, where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_1 = \Omega_1$ and $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_2 = \Omega_2$, for each $F(g_1) \in \Omega_1$ and $F(g_2) \in \Omega_2$, the recursive call $intersect(g_1, g_2, \mathcal{S}_f, \mathcal{I}_f)$ occurs, where $\mathcal{I}_f = \mathcal{I}' \cup \{\langle (s'_1, s'_2), s' \rangle\}$. But by proposition 6.5, $\mathcal{S} \vdash s'_1$ and $\mathcal{S} \vdash s'_2$, and since $ivalid(g_1, g_2, \mathcal{S}_f, \mathcal{I}_f)$ holds by lemma 6.4, therefore $s'_1$, and $s'_2$ are intermediate constructed sorts. Thus, $(s'_1, s'_2) \in \Psi$, therefore $(s'_1, s'_2) \in \Upsilon_2$; and $(s'_1, s'_2) \notin \Upsilon_1$ by assumption in this case. Therefore, $\Upsilon_1 \subset \Upsilon_2$, hence $(g_1, \Upsilon_2) \prec_I (s'_1, \Upsilon_1)$, so the claim holds in this case.

Thus, since $\prec_I$ is a lexicographic ordering, *intersect* is a well-founded recursion on $\prec_I$ by the above claim, so $intersect(s_1, s_2, \mathcal{S}, \mathcal{I})$ terminates, which proves the lemma. □

## 6.2 Soundness of *intersect*

Turning now to the soundness of *intersect*, it is necessary to prove the following lemma, which shows that recursive calls to *intersect* leave the definitions of pre-existing sorts intact.

**Lemma 6.7** For all $(s_1, s_2, \mathcal{S}, \mathcal{I})$ such that $ivalid(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds, if $intersect(s_1, s_2, \mathcal{S}, \mathcal{I})$ returns $(s', \mathcal{S}')$, then for all sorts $r$, if $\mathcal{S} \vdash r :: \mathrm{t}$, then $\mathcal{S}' \vdash r :: \mathrm{t}$, and for all $\Theta$, $[\Theta(r)]_{\mathcal{S}} = [\Theta(r)]_{\mathcal{S}'}$.

**Proof** By well founded induction on $n$, the number of calls to *intersect*.

Suppose that $ivalid(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds for some $(s_1, s_2, \mathcal{S}, \mathcal{I})$, and that $intersect(s_1, s_2, \mathcal{S}, \mathcal{I})$ returns $(s', \mathcal{S}')$. By the definition of *intersect*, the proof proceeds by the following cases:

*case (i)* $s_1 = s_2$, so that $(s', \mathcal{S}') = (s_1, \mathcal{S})$, and so the lemma clearly holds in this case.

*case (ii)* $s_1 = s_1^1 * \cdots * s_j^1$, $s_2 = s_1^2 * \cdots * s_j^2$, and for all $0 < i \leq j$ the recursive call $intersect(s_i^1, s_i^2, \mathcal{S}, \mathcal{I})$ occurs, returning $(s_i', \mathcal{S}_i)$, so that $(s', \mathcal{S}') = (s_1' * \cdots * s_j', \cup_{i=1}^{j} \mathcal{S}_j)$. But by the induction hypothesis, for all $0 < i \leq j$ and sorts $r$, if $\mathcal{S} \vdash r :: \mathrm{t}$ then $\mathcal{S}_i \vdash r :: \mathrm{t}$, and for all $\Theta$, $[\Theta(r)]_{\mathcal{S}} = [\Theta(r)]_{\mathcal{S}_i}$. And since $\mathcal{S}' = \cup_{i=1}^{j} \mathcal{S}_j$, therefore for all sorts $r$, if $\mathcal{S} \vdash r :: \mathrm{t}$ then $\mathcal{S}' \vdash r :: \mathrm{t}$, and for all $\Theta$, $[\Theta(r)]_{\mathcal{S}} = [\Theta(r)]_{\mathcal{S}'}$ in this case.

*case (iii)* $s_1' = (r\alpha_1, \ldots, r\alpha_j)\eta_1$, $s_2' = (r\alpha_1, \ldots, r\alpha_j)\eta_2$. Suppose on the one hand that there exists $s_3$ such that $\langle (s_1, s_2), s_3 \rangle \in \mathcal{I}$; then by the definition of *elab*,$(s', \mathcal{S}') = (s_3, \mathcal{S})$, so the lemma clearly holds in this case (since $\mathcal{S}' = \mathcal{S}$). Suppose on the other hand that $(s_1, s_2)$ is not a key in $\mathcal{I}'$. Let:

$$\begin{aligned}
&\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_1 = \Omega_1 \\
&\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_2 = \Omega_2 \\
&\mathcal{S}_f = \mathcal{S} \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta_f = \circ\}, \text{ where } \eta_f \text{ is fresh} \\
&\mathcal{I}_f = \mathcal{I} \cup \{\langle (s_1, s_2), (r\alpha_1, \ldots, r\alpha_j)\eta_f \rangle\}
\end{aligned}$$

Then for each $F_1(g_1) \in \Omega_1$ and $F_2(g_2) \in \Omega_2$ such that $F_1 = F_2$, the recursive call $intersect(g_1, g_2, \mathcal{S}_f, \mathcal{I}_f)$ occurs, returning $(g', \mathcal{S}_{g'})$, so that by the induction hypothesis, for all sorts $r$, if $\mathcal{S}_f \vdash r :: \mathrm{t}$ then $\mathcal{S}_{g'} \vdash r :: \mathrm{t}$, and for all $\Theta$, $[\Theta(r)]_{\mathcal{S}_f} = [\Theta(r)]_{\mathcal{S}_{g'}}$. And $\mathcal{S}'$ is the union of $\mathcal{S}$ and each $\mathcal{S}_{g'}$, with the declaration of $(r\alpha_1, \ldots, r\alpha_j)\eta_f$ updated with its full definition as computed by *intersect*. But by the definition of *intersect*, $\eta_f$ is a fresh constructor symbol, and therefore not defined in $\mathcal{S}$, so that the alteration of $\eta_f$'s definition leaves any sort defined in $\mathcal{S}$ unhanged. And thus, for all sorts $r$, if $\mathcal{S} \vdash r :: \mathrm{t}$ then $\mathcal{S}' \vdash r :: \mathrm{t}$, and for all $\Theta$, $[\Theta(r)]_{\mathcal{S}} = [\Theta(r)]_{\mathcal{S}'}$. And this was the last case to be proven. $\square$

**Lemma 6.8** For all $(s_1, s_2, \mathcal{S}, \mathcal{I})$ such that $ivalid(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds, if $intersect(r_1^1, r_2^1, \mathcal{S}_1, \mathcal{I}_1)$ and $intersect(r_1^2, r_2^2, \mathcal{S}_2, \mathcal{I}_2)$ are sibling recursive calls of $intersect(s_1, s_2, \mathcal{S}, \mathcal{I})$, returning $(s_1', \mathcal{S}_1')$ and $(s_2', \mathcal{S}_2')$ respectively, then for all sorts $r$, if $\mathcal{S}_1' \vdash r :: \mathrm{t}$, then $\mathcal{S}_1' \cup \mathcal{S}_2' \vdash r :: \mathrm{t}$, and for all $\Theta$, $[\Theta(r)]_{\mathcal{S}_1'} = [\Theta(r)]_{\mathcal{S}_1' \cup \mathcal{S}_2'}$.

**Proof** Suppose $ivalid(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds for some $(s_1, s_2, \mathcal{S}, \mathcal{I})$, and suppose $intersect(r_1^1, r_2^1, \mathcal{S}_1, \mathcal{I}_1)$ and $intersect(r_1^2, r_2^2, \mathcal{S}_2, \mathcal{I}_2)$ are sibling recursive calls of $intersect(s_1, s_2, \mathcal{S}, \mathcal{I})$, returning $(s_1', \mathcal{S}_1')$ and $(s_2', \mathcal{S}_2')$ respectively. And, suppose on the one hand that a sort $r$ is defined in $\mathcal{S}_1$; then the result follows by the previous lemma. Suppose on the other hand that $r$ is not defined in $\mathcal{S}_1$. Then, $r$ must contain some sort constructor $\eta_f$ which is defined by a descendant call of $intersect(r_1^1, r_2^1, \mathcal{S}_1, \mathcal{I}_1)$. By the definition of *intersect*, any sort constructors which are defined in a recursive call are fresh, previously undefined symbols. And the store of contructor symbols is assumed to be global, hence $\eta_f$ will not be redefined or altered by any descendant recursive call of $intersect(r_1^2, r_2^2, \mathcal{S}_2, \mathcal{I}_2)$. Therefore, for all sorts $r$, if $\mathcal{S}_1' \vdash r :: \mathrm{t}$, then $\mathcal{S}_1' \cup \mathcal{S}_2' \vdash r :: \mathrm{t}$, and for all $\Theta$, $[\Theta(r)]_{\mathcal{S}_1'} = [\Theta(r)]_{\mathcal{S}_1' \cup \mathcal{S}_2'}$. $\square$

**Definition 6.9** For all environments $\mathcal{S}$ and co-matchings $\mathcal{I}$, the relation $imatch(\mathcal{S},\mathcal{I})$ holds iff for all $\langle(s_1,s_2),s\rangle \in \mathcal{I}$, for all $\Theta$, $[\Theta(s)]_{\mathcal{S}} \subseteq [\Theta(s_1)]_{\mathcal{S}} \cap [\Theta(s_2)]_{\mathcal{S}}$.

Now, soundness can be proved fairly straightforwardly as follows.

**Lemma 6.10** For all $(s_1,s_2,\mathcal{S},\mathcal{I})$ such that $ivalid(s_1,s_2,\mathcal{S},\mathcal{I})$ and $imatch(\mathcal{S},\mathcal{I})$ hold, if intersect$(s_1,s_2,\mathcal{S},\mathcal{I})$ returns $(s',\mathcal{S}')$, then for all $\Theta$, $[\Theta(s')]_{\mathcal{S}'} \subseteq [\Theta(s_1)]_{\mathcal{S}} \cap [\Theta(s_2)]_{\mathcal{S}}$.

**Proof** By well-founded induction on $n$, the number of recursive calls to *intersect*.

**Induction Hypothesis** The lemma holds for all $m < n$.

Suppose that for some $(s_1,s_2,\mathcal{S},\mathcal{I})$, $ivalid(s_1,s_2,\mathcal{S},\mathcal{I})$ and $imatch(\mathcal{S},\mathcal{I})$ hold, and $intersect(s_1,s_2,\mathcal{S},\mathcal{I})$ returns $(s',\mathcal{S}')$, and suppose that for some substitution $\Theta$, there is a value $\mathbf{c} \in [\Theta(s')]_{\mathcal{S}'}$. Then to demonstrate the theorem, it is sufficient to prove that $\mathbf{c} \in [\Theta(s_1)]_{\mathcal{S}} \cap [\Theta(s_2)]_{\mathcal{S}}$.

If it is the case that $s_1 = s_2$, then by the definition of *elab*, $(s',\mathcal{S}') = (s_1,\mathcal{S})$, so clearly the lemma holds in this instance. Note that if $s_1$ and $s_2$ are sort variables, then since $ivalid(s_1,s_2,\mathcal{S},\mathcal{I})$ holds by assumption, so that $s_1$ and $s_2$ refine the same type by definition 6.3, therefore it must be the case that $s_1 = s_2$ by definition 2.7. Suppose on the other hand that $s_1 \neq s_2$; under this assumption, the proof then proceeds by the following cases:

*case (i)* $s' = s'_1 * \cdots * s'_j$, so that $s_1 = s^1_1 * \cdots * s^1_j$ and $s_2 = s^2_1 * \cdots * s^2_j$, where $\forall 0 < i \leq j \,.\, (s'_i, \mathcal{S}_i) = intersect(s^1_i, s^2_i, \mathcal{S}, \mathcal{I})$ and $\mathcal{S}' = \cup^j_{i=1}\mathcal{S}_i$. Since $s' = s'_1 * \cdots * s'_j$, therefore $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_j)$, where $\forall 0 < i \leq j \,.\, \mathbf{c}_i \in [\Theta(s'_i)]_{\mathcal{S}}$ by definition 3.24 and proposition 6.8. But $ivalid(s^1_i, s^2_i, \mathcal{S}, \mathcal{I})$ holds by lemma 6.4, and $imatch(\mathcal{S},\mathcal{I})$ holds by assumption, and each $intersect(s^1_i, s^2_i, \mathcal{S}, \mathcal{I})$ terminates in less than $n$ calls. Therefore by the induction hypothesis and proposition 6.8, $\forall 0 < i \leq j \,.\, [\Theta(s'_i)]_{\mathcal{S}'} \subseteq [\Theta(s^1_i)]_{\mathcal{S}} \cap [\Theta(s^2_i)]_{\mathcal{S}}$, hence $\forall 0 < i \leq j \,.\, \mathbf{c}_i \in [\Theta(s^1_i)]_{\mathcal{S}} \cap [\Theta(s^2_i)]_{\mathcal{S}}$. And therefore, $\mathbf{c} \in [\Theta(s_1)]_{\mathcal{S}} \cap [\Theta(s_2)]_{\mathcal{S}}$ by theorem 3.24.

*case (ii)* $s' = (r\alpha_1, \ldots, r\alpha_j)\eta_f$, so that $s_1 = (r\alpha_1, \ldots, r\alpha_j)\eta_1$ and $s_2 = (r\alpha_1, \ldots, r\alpha_j)\eta_2$, where $\mathcal{S}' \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_f = \Omega'$, $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_1 = \Omega_1$ and $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_2 = \Omega_2$. Suppose on the one hand that $\langle(s_1,s_2),s'\rangle \in \mathcal{I}$, and therefore $\mathcal{S}' = \mathcal{S}$ by the definition of *intersect*. Then since $imatch(\mathcal{S},\mathcal{I})$ holds by assumption, $\mathbf{c} \in [\Theta(s_1)]_{\mathcal{S}} \cap [\Theta(s_2)]_{\mathcal{S}}$ by definition 6.9.

Suppose on the other hand that $\langle(s_1,s_2),s'\rangle \notin \mathcal{I}$; then the definition $(r\alpha_1, \ldots, r\alpha_j)\eta_f = \Omega'$ is computed by *intersect* in this call. And by definition 3.24, $\mathbf{c}$ is either a constant $C$, such that $C \in \Omega'$, or $\mathbf{c}$ is a value constructor term $F(\mathbf{d})$, such that $\exists F(g') \in \Omega'$ such that $\mathbf{d} \in [\Theta(g')]_{\mathcal{S}'}$, by proposition 3.30. In the former case, since $C \in \Omega_1$ and $C \in \Omega_2$ by the definition of *intersect*, $C \in [\Theta(s_1)]_{\mathcal{S}} \cap [\Theta(s_2)]_{\mathcal{S}}$ by definition 3.24.

If $\mathbf{c} = F(\mathbf{d})$, so that $\exists F(g') \in \Omega'$ such that $\mathbf{d} \in [\Theta(g')]_{\mathcal{S}'}$, then by the definition of *intersect*, $(g', \mathcal{S}_{g'}) = intersect(g_1, g_2, \mathcal{S}_f, \mathcal{I}_f)$, where $F(g_1) \in \Omega_1$, $F(g_2) \in \Omega_2$, $\mathcal{I}_f = \mathcal{I} \cup \{\langle(s_1,s_2),s'\rangle\}$ and $\mathcal{S}_f = \mathcal{S} \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta_f = \circ\}$. Now, by lemma 6.4, $ivalid(g_1, g_2, \mathcal{S}_f, \mathcal{I}_f)$ holds, and since $[\Theta(s')]_{\mathcal{S}_f} = \emptyset$, therefore $[\Theta(s')]_{\mathcal{S}_f} \subseteq [\Theta(s_1)]_{\mathcal{S}_f} \cap [\Theta(s_2)]_{\mathcal{S}_f}$, so $imatch(\mathcal{S}_f, \mathcal{I}_f)$ holds. And since $intersect(g_1, g_2, \mathcal{S}_f, \mathcal{I}_f)$ terminates in less than $n$ recursive calls, therefore by the induction hypothesis and proposition 4.29, $[\Theta(g')]_{\mathcal{S}'} \subseteq [\Theta(g_1)]_{\mathcal{S}} \cap [\Theta(g_2)]_{\mathcal{S}}$. Therefore, since $\mathbf{d} \in [\Theta(g')]_{\mathcal{S}'}$ by assumption, $\mathbf{d} \in [\Theta(g_1)]_{\mathcal{S}} \cap [\Theta(g_2)]_{\mathcal{S}}$. And since $F(g_1) \in \Omega_1$ and $F(g_2) \in \Omega_2$, therefore $F(\mathbf{d}) \in [\Theta(s_1)]_{\mathcal{S}}$ and $F(\mathbf{d}) \in [\Theta(s_2)]_{\mathcal{S}}$ by proposition 3.30. And this covers both possibilities for $\mathbf{c}$, so that $\mathbf{c} \in [\Theta(s_1)]_{\mathcal{S}} \cap [\Theta(s_2)]_{\mathcal{S}}$ in this case, which was the last to be proven. □

## 6.3 Completeness of *intersect*, correctness of *intersection*

Completeness for *intersect* is considerably more difficult to prove than soundness. The desired completeness result must be obtained by way of derivations; however, since *intersect* memoizes partially computed declarations, a restricted sort of derivation must be defined. These derivations, called co-derivations, are "synchronized" derivations which take pairs of similar terms to pair of similar terms, with "similar" as defined immediately below. Co-derivations are like end-term derivations, in that they do not reduce certain sort symbols. A full description is given in the following definitions.

**Definition 6.11** Two sort terms $t_1$ and $t_2$ are *similar*, written $t_1 \approx t_2$, iff one of the following conditions holds:

1. $t_1$ and $t_2$ are sort symbols
2. $t_1$ and $t_2$ are both some constant $C$
3. $t_1 = (u_1, \ldots, u_j)$ and $t_2 = (v_1, \ldots, v_j)$, or $t_1 = u_1 * \cdots * u_j$ and $t_2 = v_1 * \cdots * v_j$, and for all $0 < i \leq j$, $u_i \approx v_i$.
4. $t_1 = F(u)$, $t_2 = F(v)$, and $u \approx v$.

Note that this is recursive definition. Also, note that if $t_1 = t_2$, then $t_1 \approx t_2$.

**Definition 6.12** For all sorts $s_1$ and $s_2$ and environments $\mathcal{S}$, a *simultaneous derivation* is a sequence of pairs of valid normal-form derivation steps $(t_0 \Rightarrow_{\mathcal{S}} t_1, t'_0 \Rightarrow_{\mathcal{S}} t'_1), (t_1 \Rightarrow_{\mathcal{S}} t_2, t'_1 \Rightarrow_{\mathcal{S}} t'_2), \ldots, (t_{n-1} \Rightarrow_{\mathcal{S}} t_n, t'_{n-1} \Rightarrow_{\mathcal{S}} t'_n)$ such that for all $0 < i \leq n$, $t_{i-1} \approx t'_{i-1}$ and $t_i \approx t'_i$. A simultaneous derivation of any number of elements, including the identity derivation, may be denoted by $(t_1 \Rightarrow^*_{\mathcal{S}} u_1, t_2 \Rightarrow^*_{\mathcal{S}} u_2)$, or $(t_1 \Rightarrow^n_{\mathcal{S}} u_1, t_2 \Rightarrow^n_{\mathcal{S}} u_2)$ to indicate a simultaneous derivation of length $n$.

**Definition 6.13** For all terms $t_1$, $t_2$ such that $t_1 \approx t_2$, the *corresponding subterms* of $t_1$ and $t_2$ are defined as follows:

1. $(t_1, t_2)$ are corresponding subterms of $(t_1, t_2)$.
2. If $t_1 = (u_1, \ldots, u_j)$ and $t_2 = (v_1, \ldots, v_j)$, or $t_1 = u_1 * \cdots * u_j$ and $t_2 = v_1 * \cdots * v_j$, for all $0 < i \leq j$, every corresponding subterm of $(u_i, v_i)$ is a corresponding subterm of $(t_1, t_2)$.
3. If $t_1 = F(u)$ and $t_2 = F(u)$, then every corresponding subterm of $(u, v)$ is a corresponding subterm of $(t_1, t_2)$.

**Definition 6.14** A corresponding occurence of sorts symbols $(r_1, r_2)$ in terms $(t_1, t_2)$ such that $t_1 \approx t_2$ is defined as in definition 4.40 on page 37.

**Proposition 6.15** For all derivation steps $(t_1 \Rightarrow_{\mathcal{S}} u_1, t_2 \Rightarrow_{\mathcal{S}} u_2)$ in any simultaneous derivation, corresponding subterms of $(t_1, t_2)$ are reduced to corresponding subterms of $(u_1, u_2)$ in the derivation steps $t_1 \Rightarrow_{\mathcal{S}} u_1$ and $t_2 \Rightarrow_{\mathcal{S}} u_2$.

**Proof** This follows immediately from definitions 6.11 and 6.12, since the derivation steps $t_1 \Rightarrow_{\mathcal{S}} u_1$ and $t_2 \Rightarrow_{\mathcal{S}} u_2$ are in normal form, and $t_1 \approx t_2$ and $u_1 \approx u_2$. □

**Definition 6.16** A *pairing* $\mathcal{P}$ is a set of instantiated intermediate constructed sort pairs $(s_1, s_2)$.

In any simultaneous derivation, each corresponding occurence of sort symbols $(s_1, s_2)$ will have a first occurence, either due to a derivation step or because they appear in the initial terms in the derivation. This is described formally in the following definition. The number of first, or *initial*, occurences of sort symbols $(s_1, s_2)$, will be a useful value for the purposes of induction in in lemma 6.29, which is the completeness proof for *intersect*.

**Definition 6.17** For any simultaneous derivation $(s_1 \Rightarrow^*_{\mathcal{S}} t_1, s_2 \Rightarrow^*_{\mathcal{S}} t_2)$ and sort symbols $(r_1, r_2)$, any corresponding occurence of $(r_1, r_2)$ in $(s_1, s_2)$ is an *initial occurence* of $(r_1, r_2)$ in $(s_1 \Rightarrow^*_{\mathcal{S}} t_1, s_2 \Rightarrow^*_{\mathcal{S}} t_2)$; and if a derivation step $(u_1 \Rightarrow_{\mathcal{S}} v_1, u_2 \Rightarrow_{\mathcal{S}} v_2)$ in $(s_1 \Rightarrow^*_{\mathcal{S}} t_1, s_2 \Rightarrow^*_{\mathcal{S}} t_2)$ reduces the corresponding subterms $(u'_1, u'_2)$ of $(u_1, u_2)$ to the corresponding subterms $(v'_1, v'_2)$ of $(v_1, v_2)$, then any corresponding occurence of $(r_1, r_2)$ in $(v'_1, v'_2)$ is an *initial occurrence* of $(r_1, r_2)$ in the co-derivation $(s_1 \Rightarrow^*_{\mathcal{S}} t_1, s_2 \Rightarrow^*_{\mathcal{S}} t_2)$.

**Definition 6.18** For all sorts $s_1$ and $s_2$, environments $\mathcal{S}$, pairings $\mathcal{P}$ and terms $t_1$ and $t_2$, any simultaneous derivation $(s_1 \Rightarrow^*_{\mathcal{S}} t_1, s_2 \Rightarrow^*_{\mathcal{S}} t_2)$ is a *co-derivation* in $\mathcal{P}$ iff one of the two following conditions holds:

1. $s_1 = s_2$ and $t_1$ and $t_2$ are the same value $\mathbf{c}$.
2. $s_2 \neq s_2$, and for all $(r_1, r_2) \in \mathcal{P}$, if there is an initial occurence of $(r_1, r_2)$ in $(s_1 \Rightarrow^*_{\mathcal{S}} t_1, s_2 \Rightarrow^*_{\mathcal{S}} t_2)$, then those particular occurences of $r_1$ and $r_2$ are not reduced in any subsequent derivation steps in $(s_1 \Rightarrow^*_{\mathcal{S}} t_1, s_2 \Rightarrow^*_{\mathcal{S}} t_2)$, and $t_1$ and $t_2$ are irreducible by this criteria; i.e., $t_1$ and $t_2$ are values, or if there is a sort symbol $r_1$ in $t_1$, there is a sort symbol $r_2$ in the corresponding subterm of $t_2$ and $(r_1, r_2) \in \mathcal{P}$.

As will be demonstrated in proposition 6.23, if some value $\mathbf{c}$ is in $[s_1]_{\mathcal{S}} \cap [s_2]_{\mathcal{S}}$, then there exists a co-derivation $(s_1 \Rightarrow^*_{\mathcal{S}} \mathbf{c},\ s_2 \Rightarrow^*_{\mathcal{S}} \mathbf{c})$ in $\emptyset$. Thus, co-derivations are introduced as a way to obtain a generalized completeness result for *intersect*, since co-derivations have explicit properties supporting such a proof (in much the same way that end-term and simple derivations supported the completeness results for *elab* and *empty*).

**Example 6.19** Let $\mathcal{S}$ be an environment containing the declarations $(r\alpha)\eta_1 = C \mid G \mid F(r\alpha * (r\alpha)\eta_1)$, $(r\alpha)\eta_2 = C \mid F(r\alpha * (r\alpha)\eta_2)$, $\eta_3 = A \mid B$ and $\eta_4 = A$. Then the following is a co-derivation in $\{(\eta_3, \eta_4)\}$:

$$((\eta_3)\eta_1 \Rightarrow_{\mathcal{S}} F(\eta_3 * (\eta_3)\eta_1),\ (\eta_4)\eta_2 \Rightarrow_{\mathcal{S}} F(\eta_4 * (\eta_4)\eta_2)),$$
$$(F(\eta_3 * (\eta_3)\eta_1) \Rightarrow_{\mathcal{S}} F(\eta_3, (\eta_3)\eta_1),\ F(\eta_4 * (\eta_4)\eta_2) \Rightarrow_{\mathcal{S}} F(\eta_4, (\eta_4)\eta_2)),$$
$$(F(\eta_3, (\eta_3)\eta_1) \Rightarrow_{\mathcal{S}} F(\eta_3, C),\ F(\eta_4, (\eta_4)\eta_2) \Rightarrow_{\mathcal{S}} F(\eta_4, C))$$

and this co-derivation can be written $((\eta_3)\eta_1 \Rightarrow^*_{\mathcal{S}} F(\eta_3, C),\ (\eta_4)\eta_2 \Rightarrow^*_{\mathcal{S}} F(\eta_4, C))$.

**Proposition 6.20** For all sorts $s_1$ and $s_2$, environments $\mathcal{S}$ terms $t_1$ and $t_2$ and pairings $\mathcal{P}$, if $(s_1 \Rightarrow_{\mathcal{S}}^* t_1, s_2 \Rightarrow_{\mathcal{S}}^* t_2)$ is a co-derivation in $\mathcal{P}$, then one of the following cases hold:

1. $s_1 = s_2$, and $t_1$ and $t_2$ are both the same value $\mathbf{c}$.

2. $s_1$ is a product sort $r_1 * \cdots * r_j$, $s_2$ is a product sort $k_1 * \cdots * k_j$, and $t_1$ and $t_2$ are $j$-tuples $(u_1, \ldots, u_j)$ and $(v_1, \ldots, v_j)$.

3. $s_1$ and $s_2$ are sort symbols, $(s_1, s_2) \notin \mathcal{P}$ and $t_1$ and $t_2$ are both the same constant $C$, or $t_1$ and $t_2$ are value constructor terms $F(u)$ and $F(v)$.

4. $s_1$ and $s_2$ are sort symbols, $(s_1, s_2) \in \mathcal{P}$, and $t_1 = s_1$ and $t_2 = s_2$.

**Proof** Suppose that $(s_1 \Rightarrow_{\mathcal{S}}^* t_1, s_2 \Rightarrow_{\mathcal{S}}^* t_2)$ is a co-derivation in $\mathcal{P}$. If $s_1 = s_2$, then $t_1 = t_2 = \mathbf{c}$ by definition 6.18. Otherwise, by definition 6.12, $s_1 \approx s_2$; thus, $s_1$ and $s_2$ are either product sorts of the same arity, or $s_1$ and $s_2$ are sort symbols. If $s_1$ and $s_2$ are product sorts, then clearly $t_1$ and $t_2$ are $j$-tuples $(u_1, \ldots, u_j)$ and $(v_1, \ldots, v_j)$ by definition 6.12.

If $s_1$ and $s_2$ are sort symbols, then it may or may not be the case that $(s_1, s_2) \in \mathcal{P}$. If so, then $t_1 = s_1$ and $t_2 = s_2$ by definition 6.18. Otherwise, $s_1 \Rightarrow_{\mathcal{S}}^* t_1$ and $s_2 \Rightarrow_{\mathcal{S}}^* t_2$ are valid derivations which are not the identity derivation, by definition 6.18, and $t_1 \approx t_2$, by definition 6.12. Therefore, by definitions 3.14 and 6.11, $t_1$ and $t_2$ are both the same constant $C$, or $t_1$ and $t_2$ are value constructor terms $F(u)$ and $F(v)$. $\square$

**Proposition 6.21** For all intermediate sorts $s_1$ and $s_2$, environments $\mathcal{S}$, pairings $\mathcal{P}$, terms $t_1$ and $t_2$ and substitutions $\Theta$, the following statements hold:

1. If $s_1 = s_1^1 * \cdots * s_j^1$, $s_2 = s_1^2 * \cdots * s_j^2$, $t_1 = (u_1, \ldots, u_j)$ and $t_2 = (v_1, \ldots, v_j)$, then there exists a co-derivation $(\Theta(s_1) \Rightarrow_{\mathcal{S}}^* t_1, \Theta(s_2) \Rightarrow_{\mathcal{S}}^* t_2)$ in $\mathcal{P}$ iff for all $0 < i \leq j$, there exists a co-derivation $(\Theta(s_i^2) \Rightarrow_{\mathcal{S}}^* u_i, \Theta(s_i^2) \Rightarrow_{\mathcal{S}}^* v_i)$ in $\mathcal{P}$.

2. If $t_1 = F(u)$ and $t_2 = F(v)$, and $s_1 = (r\alpha_1, \ldots, r\alpha_j)\eta_1$ and $s_2 = (r\alpha_1, \ldots, r\alpha_j)\eta_2$, where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_1 = \Omega_1$ and $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_2 = \Omega_2$, then there exists a co-derivation $(\Theta(s_1) \Rightarrow_{\mathcal{S}}^* t_1, \Theta(s_2) \Rightarrow_{\mathcal{S}}^* t_2)$ in $\mathcal{P}$ iff $(\Theta(s_1), \Theta(s_2)) \notin \mathcal{P}$ and there is some $F(g_1) \in \Omega_1$ and $F(g_2) \in \Omega_2$ such that there exists a co-derivation $(\Theta(g_1) \Rightarrow_{\mathcal{S}}^* u, \Theta(g_2) \Rightarrow_{\mathcal{S}}^* v)$ in $\mathcal{P}$.

**Proof** 1. Let there be $s_1 = s_1^1 * \cdots * s_j^1$, $s_2 = s_1^2 * \cdots * s_j^2$, $t_1 = (u_1, \ldots, u_j)$ and $t_2 = (v_1, \ldots, v_j)$, and for all $0 < i \leq j$, let $r_i = \Theta(s_i^1)$ and $k_i = \Theta(s_i^2)$.

*a)* Suppose on the one hand that $(\Theta(s_1) \Rightarrow_{\mathcal{S}}^* t_1, \Theta(s_2) \Rightarrow_{\mathcal{S}}^* t_2)$ is a co-derivation in $\mathcal{P}$. Clearly, $(r_1 * \cdots * r_j \Rightarrow_{\mathcal{S}} (r_1, \ldots, r_j), k_1 * \cdots * k_j \Rightarrow_{\mathcal{S}} (k_1, \ldots, k_j))$ is the first step in $(\Theta(s_1) \Rightarrow_{\mathcal{S}}^* t_1, \Theta(s_2) \Rightarrow_{\mathcal{S}}^* t_2)$, by definitions 3.3 and 6.12. And since the derivations are in normal form, for all $0 < i \leq j$, there is a simultaneous derivation $((u_1, \ldots, r_i, \ldots, r_j) \Rightarrow_{\mathcal{S}}^* (u_1, \ldots, u_i, \ldots, r_j), (v_1, \ldots, k_i, \ldots, k_j) \Rightarrow_{\mathcal{S}}^* (v_1, \ldots, v_i, \ldots, k_j))$ which is a sub-sequence of the co-derivation $(\Theta(s_1) \Rightarrow_{\mathcal{S}}^* t_1, \Theta(s_2) \Rightarrow_{\mathcal{S}}^* t_2)$. Let $(r_i \Rightarrow_{\mathcal{S}}^* u_i, k_i \Rightarrow_{\mathcal{S}}^* v_i)$ be the sequence obtained by replacing each step $((u_1, \ldots, y_1, \ldots, r_j) \Rightarrow_{\mathcal{S}} (u_1, \ldots, y_2, \ldots, r_j), (v_1, \ldots, y_1', \ldots, k_j) \Rightarrow_{\mathcal{S}} (v_1, \ldots, y_2', \ldots, k_j))$ in each of these subsequences by $(y_1 \Rightarrow_{\mathcal{S}} y_2, y_1' \Rightarrow_{\mathcal{S}} y_2')$. Then each step $(y_1 \Rightarrow_{\mathcal{S}} y_2, y_1' \Rightarrow_{\mathcal{S}} y_2')$ is a valid derivation step by definition 6.18, and since $(y_1, y_1')$ are corresponding subterms of $((u_1, \ldots, y_1, \ldots, r_j), (v_1, \ldots, y_1', \ldots, k_j))$, and $(u_1, \ldots, y_1, \ldots, r_j) \approx (v_1, \ldots, y_1', \ldots, k_j)$ by definition 6.12, therefore $y_1 \approx y_1'$ by definition 6.11, and $y_2 \approx y_2'$ by similar reasoning. Hence, for all $0 < i \leq j$, $(r_i \Rightarrow_S^* u_i, k_i \Rightarrow_S^* v_i)$ is a simultaneous derivation.

Also, since for each simultaneous derivation step $(y_1 \Rightarrow_S y_2, y_1' \Rightarrow_S y_2')$ in $(r_i \Rightarrow_S^* u_i, k_i \Rightarrow_S^* v_i)$ there exists a derivation step $((u_1, \ldots, y_1, \ldots, r_j) \Rightarrow_S (u_1, \ldots, y_2, \ldots, r_j), (v_1, \ldots, y_1', \ldots, k_j) \Rightarrow_S (v_1, \ldots, y_2', \ldots, k_j))$ in $(\Theta(s_1) \Rightarrow_S^* t_1, \Theta(s_2) \Rightarrow_S^* t_2)$ by construction, therefore there is an initial occurrence of some $(d_1, d_2) \in \mathcal{P}$ in the derivation step $(y_1 \Rightarrow_S y_2, y_1' \Rightarrow_S y_2')$ only if there is a initial occurence of $(d_1, d_2)$ in the derivation step $((u_1, \ldots, y_1, \ldots, r_j) \Rightarrow_S (u_1, \ldots, y_2, \ldots, r_j), (v_1, \ldots, y_1', \ldots, k_j) \Rightarrow_S (v_1, \ldots, y_2', \ldots, k_j))$. And if there is an initial occurence of $(d_1, d_2)$ in this derivation step, the simultaneous derivation $((u_1, \ldots, y_2, \ldots, r_j) \Rightarrow_S^* (u_1, \ldots, u_i, \ldots, r_j), (v_1, \ldots, y_2', \ldots, k_j) \Rightarrow_S^* (v_1, \ldots, v_i, \ldots, k_j))$, which is a sub-sequence of $(\Theta(s_1) \Rightarrow_S^* t_1, \Theta(s_2) \Rightarrow_S^* t_2)$, does not reduce those occurences of $d_1$ and $d_2$, by definition 6.18. But then the sequence $(y_2 \Rightarrow_S^* u_i, y_2' \Rightarrow_S^* v_i)$ does not reduce those corresponding occurences of $(d_1, d_2)$, since each transition in the sequence is taken from $((u_1, \ldots, y_2, \ldots, r_j) \Rightarrow_S^* (u_1, \ldots, u_i, \ldots, r_j), (v_1, \ldots, y_2', \ldots, k_j) \Rightarrow_S^* (v_1, \ldots, v_i, \ldots, k_j))$.

Thus, for all $0 < i \leq j$ if there is an initial occurence of any $(d_1, d_2) \in \mathcal{P}$ in the simultaneous derivation $(r_i \Rightarrow_S^* u_i, k_i \Rightarrow_S^* v_i)$, then those occurences of $d_1$ and $d_2$ are not reduced by any subsequent step. And clearly, for all $0 < i \leq j$, $u_i$ and $v_i$ are irreducible, since they are corresponding subterms of $t_1$ and $t_2$ respectively, which by assumption are irreducible. And therefore, for all $0 < i \leq j$, $(\Theta(s_i^1) \Rightarrow_S^* u_i, \Theta(s_i^2) \Rightarrow_S^* v_i)$ is a co-derivation in $\mathcal{P}$.

*b)* Suppose, on the other hand, that for all $0 < i \leq j$, $(r_i \Rightarrow_S^* u_i, k_i \Rightarrow_S^* v_i)$ is a co-derivation in $\mathcal{P}$. For all $0 < i \leq j$, let $((u_1, \ldots, r_i, \ldots, r_j) \Rightarrow_S^* (u_1, \ldots, u_i, \ldots, r_j), (v_1, \ldots, k_i, \ldots, k_j) \Rightarrow_S^* (v_1, \ldots, v_i, \ldots, k_j))$ be the sequence obtained by replacing each step $(y_1 \Rightarrow_S y_2, y_1' \Rightarrow_S y_2')$ in $(r_i \Rightarrow_S^* u_i, k_i \Rightarrow_S^* v_i)$ by $((u_1, \ldots, y_1, \ldots, r_j) \Rightarrow_S (u_1, \ldots, y_2, \ldots, r_j), (v_1, \ldots, y_1', \ldots, k_j) \Rightarrow_S (v_1, \ldots, y_2', \ldots, k_j))$. For each such step obtained, since $y_1 \approx y_1'$, $\forall 0 < x \leq i-1 \,.\, u_x \approx v_x$ and $\forall i < x \leq j \,.\, r_x \approx k_x$ by definition 6.12, therefore $((u_1, \ldots, y_1, \ldots, r_j) \approx (v_1, \ldots, y_1', \ldots, k_j)$ by definition 6.11, and likewise $(u_1, \ldots, y_2, \ldots, r_j) \approx (v_1, \ldots, y_2', \ldots, k_j)$. Therefore, each sequence obtained by this replacement is a simultaneous derivation.

Also, since for each simultaneous derivation step $((u_1, \ldots, y_1, \ldots, r_j) \Rightarrow_S (u_1, \ldots, y_2, \ldots, r_j), (v_1, \ldots, y_1', \ldots, k_j) \Rightarrow_S (v_1, \ldots, y_2', \ldots, k_j))$ there exists a step $(y_1 \Rightarrow_S y_2, y_1' \Rightarrow_S y_2')$ in $(r_i \Rightarrow_S^* u_i, k_i \Rightarrow_S^* v_i)$ by construction, therefore there is an initial occurence of some $(d_1, d_2) \in \mathcal{P}$ in $((u_1, \ldots, y_1, \ldots, r_j) \Rightarrow_S (u_1, \ldots, y_2, \ldots, r_j), (v_1, \ldots, y_1', \ldots, k_j) \Rightarrow_S (v_1, \ldots, y_2', \ldots, k_j))$ only if there is an initial occurrence of $(d_1, d_2)$ in the step $(y_1 \Rightarrow_S y_2, y_1' \Rightarrow_S y_2')$. And since the sequence $(y_2 \Rightarrow_S^* u_i, y_2' \Rightarrow_S^* v_i)$ does not reduce these occurences of $d_1$ and $d_2$ by definition 6.18, therefore the sequence $((u_1, \ldots, y_2, \ldots, r_j) \Rightarrow_S (u_1, \ldots, u_i, \ldots, r_j), (v_1, \ldots, y_2', \ldots, k_j) \Rightarrow_S (v_1, \ldots, v_i, \ldots, k_j))$ does not reduce these corresponding occurences of $d_1$ and $d_2$, since each transition in the sequence is taken from $(y_2 \Rightarrow_S^* u_i, y_2' \Rightarrow_S^* v_i)$.

Thus, by concatenating each of these sequences together in order, a simultaneous derivation $((r_1,\ldots,r_j) \Rightarrow_S^* (u_1,\ldots,u_j),(k_1,\ldots,k_j) \Rightarrow_S^* (v_1,\ldots,v_j)$ is obtained, and if any element $(d_1,d_2) \in \mathcal{P}$ has an initial occurence in the sequence, then those occurrences of $d_1$ and $d_2$ are not reduced by any subsequent transition. Since for all $0 < i \leq j$, $u_i$ and $v_i$ are not reducible by assumption, therefore $t_1$ and $t_2$ are not reducible. And since $(\Theta(s_1) \Rightarrow_S (r_1,\ldots,r_j),\Theta(s_2) \Rightarrow_S (k_1,\ldots,k_j))$ is a valid derivation step, therefore by definition 6.18, $(\Theta(s_1) \Rightarrow_S^* t_1, \Theta(s_2) \Rightarrow_S^* t_2)$ is a co-derivation in $\mathcal{P}$.

2. Let there be $t_1 = F(u)$, $t_2 = F(v)$, $s_1 = (r\alpha_1,\ldots,r\alpha_j)\eta_1$ and $s_2 = (r\alpha_1,\ldots,r\alpha_j)\eta_2$, and let $\mathcal{S} \vdash (r\alpha_1,\ldots,r\alpha_j)\eta_1 = \Omega_1$ and $\mathcal{S} \vdash (r\alpha_1,\ldots,r\alpha_j)\eta_2 = \Omega_2$.

*a)* Suppose on the one hand that $(\Theta(s_1) \Rightarrow_S^* F(u), \Theta(s_2) \Rightarrow_S^* F(v))$ is a co-derivation in $\mathcal{P}$. Then by proposition 6.20, $(\Theta(s_1),\Theta(s_2)) \notin \mathcal{P}$. And by definition 6.18, for sorts $g_1'$ and $g_2'$, $(\Theta(s_1) \Rightarrow_S F(g_1'), \Theta(s_2) \Rightarrow_S F(g_2'))$ is the first step in this derivation; and since each of these transitions must be valid, therefore it must be the case that there exist $F(g_1) \in \Omega_1$ and $F(g_2) \in \Omega_2$ such that $\Theta(g_1) = g_1'$ and $\Theta(g_2) = g_2'$, by proposition 3.30.

Now, let $(g_1' \Rightarrow_S^* u, g_2' \Rightarrow_S^* v)$ be the sequence obtained by replacing every step $(F(y_1) \Rightarrow_S F(y_2), F(y_1') \Rightarrow_S F(y_2'))$ in the sequence $(F(g_1') \Rightarrow_S^* F(u), F(g_2') \Rightarrow_S^* F(v))$ by $(y_1 \Rightarrow_S y_2, y_1' \Rightarrow_S y_2')$. Since for each of these steps, $F(y_1) \approx F(y_1')$ and $F(y_2) \approx F(y_2')$, therefore $y_1 \approx y_1'$ and $y_2 \approx y_2'$ by definition 6.11, and also clearly each step $(y_1 \Rightarrow_S y_2, y_1' \Rightarrow_S y_2')$ is a valid derivation step by definition 6.11. Therefore $(g_1' \Rightarrow_S^* u, g_2' \Rightarrow_S^* v)$ is a simultaneous derivation.

Also, since for each step $(y_1 \Rightarrow_S y_2, y_1' \Rightarrow_S y_2')$ there exists a step $(F(y_1) \Rightarrow_S F(y_2), F(y_1') \Rightarrow_S F(y_2'))$ by construction, therefore some $(d_1,d_2) \in \mathcal{P}$ has an initial occurence in the step $(y_1 \Rightarrow_S y_2, y_1' \Rightarrow_S y_2')$ only if it has an initial occurence in the step $(F(y_1) \Rightarrow_S F(y_2), F(y_1') \Rightarrow_S F(y_2'))$. And since the simultaneous derivation $(F(y_2) \Rightarrow_S^* F(u), F(y_2') \Rightarrow_S^* F(v))$ does not reduce these corresponding occurences of $d_1$ and $d_2$ by assumption and definition 6.18, therefore the simultaneous derivation $(y_2 \Rightarrow_S^* u, y_2' \Rightarrow_S^* v)$ does not reduce these occurrences of $d_1$ and $d_2$, since the transitions in $(y_2 \Rightarrow_S^* u, y_2' \Rightarrow_S^* v)$ are taken from the transitions in $(F(y_2) \Rightarrow_S^* F(u), F(y_2') \Rightarrow_S^* F(v))$. Therefore, the simultaneous derivation $(g_1' \Rightarrow_S^* u, g_2' \Rightarrow_S^* v)$ does not subsequently reduce any initial occurence of any $(d_1,d_2) \in \mathcal{P}$, and since $F(u)$ and $F(v)$ are irreducible, therefore $u$ and $v$ are irreducible. Thus, $(g_1' \Rightarrow_S^* u, g_2' \Rightarrow_S^* v)$, that is, $(\Theta(g_1) \Rightarrow_S^* u, \Theta(g_2) \Rightarrow_S^* v)$ is a co-derivation in $\mathcal{P}$.

*b)* Suppose on the other hand that $(\Theta(s_1),\Theta(s_2)) \notin \mathcal{P}$, and that there exists $F(g_1) \in \Omega_1$ and $F(g_2) \in \Omega_2$ such that $(\Theta(g_1) \Rightarrow_S^* u, \Theta(g_2) \Rightarrow_S^* v)$ is a co-derivation in $\mathcal{P}$.

Let $g_1' = \Theta(g_1)$ and $g_2' = \Theta(g_2)$, and let $(F(g_1') \Rightarrow_S^* F(u), F(g_2') \Rightarrow_S^* F(v))$ be the sequence obtained by replacing every step $(y_1 \Rightarrow_S y_2, y_1' \Rightarrow_S y_2')$ in the sequence $(g_1' \Rightarrow_S^* u, g_2' \Rightarrow_S^* v)$ by $(F(y_1) \Rightarrow_S F(y_2), F(y_1') \Rightarrow_S F(y_2'))$. Since for each of these steps $y_1 \approx y_1'$ and $y_2 \approx y_2'$, by definition 6.11, therefore $F(y_1) \approx F(y_1')$ and $F(y_2) \approx F(y_2')$, and also clearly each step $(F(y_1) \Rightarrow_S F(y_2), F(y_1') \Rightarrow_S F(y_2'))$ is a valid derivation step by definitions 6.11 and 3.14. Therefore $(F(g'_1) \Rightarrow^*_S F(u), F(g'_2) \Rightarrow^*_S F(v))$ is a simultaneous derivation.

Also, since for each step $(F(y_1) \Rightarrow_S F(y_2), F(y'_1) \Rightarrow_S F(y'_2))$ there exists a step $(y_1 \Rightarrow_S y_2, y'_1 \Rightarrow_S y'_2)$ by construction, therefore some $(d_1, d_2) \in \mathcal{P}$ has an initial occurence in the step $(F(y_1) \Rightarrow_S F(y_2), F(y'_1) \Rightarrow_S F(y'_2))$ only if it has an initial occurence in the step $(y_1 \Rightarrow_S y_2, y'_1 \Rightarrow_S y'_2)$. And since the simultaneous derivation $(y_2 \Rightarrow^*_S u, y'_2 \Rightarrow^*_S v)$ does not reduce these corresponding occurences of $d_1$ and $d_2$ by assumption and definition 6.18, therefore the simultaneous derivation $(F(y_2) \Rightarrow^*_S F(u), F(y'_2) \Rightarrow^*_S F(v))$ does not reduce these occurrences of $d_1$ and $d_2$, since the transitions in $(F(y_2) \Rightarrow^*_S F(u), F(y'_2) \Rightarrow^*_S F(v))$ are taken from the transitions in $(y_2 \Rightarrow^*_S u, y'_2 \Rightarrow^*_S v)$. Therefore, the simultaneous derivation $(g'_1 \Rightarrow^*_S u, g'_2 \Rightarrow^*_S v)$ does not subsequently reduce any initial correspoding occurence of any $(d_1, d_2) \in \mathcal{P}$, and since $u$ and $v$ are irreducible, therefore $F(u)$ and $F(v)$ are irreducible. And since $(\Theta(s_1), \Theta(s_2)) \notin \mathcal{P}$, and $F(g_1) \in \Omega_1$ and $F(g_2) \in \Omega_2$, therefore $(\Theta(s_1) \Rightarrow_S F(g'_1), \Theta(s_2) \Rightarrow_S F(g'_2))$ is a valid simultaneous derivation step by definition 6.11 and proposition 3.30, therefore $(\Theta(s_1) \Rightarrow^*_S F(u), \Theta(s_2) \Rightarrow^*_S F(v))$ is a co-derivation in $\mathcal{P}$. $\square$

**Proposition 6.22** If $(s_1 \Rightarrow^*_S t_1, s_2 \Rightarrow^*_S t_2)$ is a co-derivation in $\mathcal{P}$, and $(r_1, r_2)$ has no initial occurence in this co-derivation, then $(s_1 \Rightarrow^*_S t_1, s_2 \Rightarrow^*_S t_2)$ is a co-derivation in $\mathcal{P} \cup \{(r_1, r_2)\}$.

**Proof** This is true by definition, since if $(r_1, r_2)$ has no initial occurence in $(s_1 \Rightarrow^*_S t_1, s_2 \Rightarrow^*_S t_2)$, then there are no such occurences of $r_1$ and $r_2$ which can be reduced by a subsequent derivation step in $(s_1 \Rightarrow^*_S t_1, s_2 \Rightarrow^*_S t_2)$. $\square$

**Proposition 6.23** For all intermediate sorts $s_1$ and $s_2$, environments $\mathcal{S}$ such that $\mathcal{S} \vdash s_1 :: \mathrm{t}$ and $\mathcal{S} \vdash s_2 :: \mathrm{t}$, substitutions $\Theta$ and values $\mathbf{c}$, if $\mathbf{c} \in [\Theta(s_1)] \cap [\Theta(s_2)]$, then there exists a co-derivation $(\Theta(s_1) \Rightarrow^*_S \mathbf{c}, \Theta(s_2) \Rightarrow^*_S \mathbf{c})$ in $\emptyset$.

Suppose that for some $s_1$, $s_2$ and $\mathcal{S}$ satisfying the necessary preconditions of the proposition, there exists a value $\mathbf{c} \in [\Theta(s_1)] \cap [\Theta(s_2)]$. Therefore, there exist derivations $\Theta(s_1) \Rightarrow^*_S \mathbf{c}$ and $\Theta(s_2) \Rightarrow^*_S \mathbf{c}$, by definition 3.22. To demonstrate the proposition, it is sufficient to prove that there exists a co-derivation $(\Theta(s_1) \Rightarrow^*_S \mathbf{c}, \Theta(s_2) \Rightarrow^*_S \mathbf{c})$ in $\emptyset$.

**Proof** By structural induction on $\mathbf{c}$.

**Induction Hypothesis** For all intermediate sorts $r_1$ and $r_2$ such that $\mathcal{S} \vdash r_1 :: \mathrm{t}'$ and $\mathcal{S} \vdash r_2 :: \mathrm{t}'$ for some type $\mathrm{t}'$, and for all subterms $\mathbf{d}$ of $\mathbf{c}$, if there exist derivations $\Theta(r_1) \Rightarrow^*_S \mathbf{d}$ and $\Theta(r_2) \Rightarrow^*_S \mathbf{d}$, then there exists a co-derivation $(\Theta(r_1) \Rightarrow^*_S \mathbf{d}, \Theta(r_2) \Rightarrow^*_S \mathbf{d})$ in $\emptyset$.

*case(i)* $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_j)$, so that $s_1 = s^1_1 * \cdots * s^1_j$ and $s_2 = s^2_1 * \cdots * s^2_j$ by definitions 3.24 and 3.22, and for all $0 < i \leq j$, $\mathcal{S} \vdash s^1_i :: \mathrm{t}'$ and $\mathcal{S} \vdash s^1_i :: \mathrm{t}'$ by definition 3.27, and $\Theta(s^1_i) \Rightarrow^*_S \mathbf{c}_i$ and $\Theta(s^2_i) \Rightarrow^*_S \mathbf{c}_i$ by definitions 3.24 and 3.22. Therefore by the induction hypothesis, for all $0 < i \leq j$, there exists a co-derivation $(\Theta(s^1_i) \Rightarrow^*_S \mathbf{c}_i, \Theta(s^2_i) \Rightarrow^*_S \mathbf{c}_i)$ in $\emptyset$, and therefore by proposition 6.21, there exists a co-derivation $(\Theta(s_1) \Rightarrow^*_S \mathbf{c}, \Theta(s_2) \Rightarrow^*_S \mathbf{c})$ in $\emptyset$ in this case.

*case(ii)* $\mathbf{c} = C$, so that $s_1 = (r\alpha_1, \ldots, r\alpha_j)\eta_1$ and $s_2 = (r\alpha_1, \ldots, r\alpha_j)\eta_2$, and where $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_1 = \Omega_1$ and $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_2 = \Omega_2$, $C \in \Omega_1, \Omega_2$ by definition 3.14. And clearly $(s_1, s_2) \notin \emptyset$, therefore by definition 6.18, $(\Theta(s_1) \Rightarrow_S C, \Theta(s_2) \Rightarrow_S C)$ is a co-derivation in $\emptyset$, so that there exists a co-derivation $(\Theta(s_1) \Rightarrow_S^* \mathbf{c}, \Theta(s_2) \Rightarrow_S^* \mathbf{c})$ in $\emptyset$ in this case.

*case (iii)* $\mathbf{c} = F(\mathbf{d})$, so that with $s_1$, $s_2$, $\Omega_1$ and $\Omega_2$ as defined in the previous case, there exists $F(g_1) \in \Omega_1$ and $F(g_2) \in \Omega_2$ such that $\Theta(g_1) \Rightarrow_S^* \mathbf{d}$ and $\Theta(g_2) \Rightarrow_S^* \mathbf{d}$, by proposition 3.30. But since $\mathcal{S} \vdash g_1 :: \mathrm{t}'$ and $\mathcal{S} \vdash g_2 :: \mathrm{t}'$ by definition 3.27, therefore by the induction hypothesis, there exists a co-derivation $(\Theta(g_1) \Rightarrow_S^* \mathbf{d}, \Theta(g_2) \Rightarrow_S^* \mathbf{d})$ in $\mathcal{P}$. And since $(s_1, s_2) \notin \emptyset$, therefore by proposition 6.21, there exists a co-derivation $(\Theta((r\alpha_1, \ldots, r\alpha_j)\eta_1) \Rightarrow_S^* F(\mathbf{d}), \Theta((r\alpha_1, \ldots, r\alpha_j)\eta_2)) \Rightarrow_S^* F(\mathbf{d})$ in $\emptyset$, so that there exists a co- derivation $(\Theta(s_1) \Rightarrow_S^* \mathbf{c}, s_2 \Rightarrow_S^* \mathbf{c})$ in $\emptyset$ in this case.

And this covers all possible cases, so the proposition holds. □

**Definition 6.24** A term $t$ is said to be equivalent to a pair of terms $(t_1, t_2)$ in a co-matching $\mathcal{I}$, written $t \equiv_{\mathcal{I}} (t_1, t_2)$, iff one of the following conditions holds:

1. $t_1 = t_2 = t$
2. $t_1 = (u_1, \ldots, u_j)$, $t_2 = (v_1, \ldots, v_j)$, $t = (t'_1, \ldots, t'_j)$, and for all $0 < i \leq j$, $t'_i \equiv_{\mathcal{I}} (u_i, v_i)$.
3. $t_1 = F(u_1)$, $t_2 = F(u_2)$, $t = F(u)$, and $u \equiv_{\mathcal{I}} (u_1, u_2)$.
4. $t_1$, $t_2$ and $t$ are sort symbols, and $\langle (t_1, t_2), t \rangle \in \mathcal{I}$.

Note that this is a recursive definition.

**Example 6.25** The following terms are equivalent in a co-matching $\mathcal{I} = \{\langle (\eta_3, \eta_4), \eta' \rangle\}$:

$$F(\eta', C) \equiv_{\mathcal{I}} (F(\eta_3, C), F(\eta_4, C))$$

$$F(\eta', F(\eta', C)) \equiv_{\mathcal{I}} (F(\eta_3, F(\eta_3, C)), F(\eta_4, F(\eta_4, C)))$$

$$(G, H(G, \eta')) \equiv_{\mathcal{I}} ((G, H(G, \eta_3)), (G, H(G, \eta_4)))$$

**Definition 6.26** For all sort symbols $s$, $s_1$ and $s_2$, terms $t$, $t_1$, $t_2$ and co-matchings $\mathcal{I}$ such that $t \equiv_{\mathcal{I}} (t_1, t_2)$, the corresponding occurences of $s$ and $(s_1, s_2)$ in $t$ and $(t_1, t_2)$ are defined as follows (note that this makes "corresponding occurences" an overloaded term):

1. If $t = (t'_1, \ldots, t'_j)$, $t_1 = (u_1, \ldots, u_j)$ and $t_2 = (v_1, \ldots, v_j)$, then for all $0 < i \leq j$, the corresponding occurences of $s$ and $(s_1, s_2)$ in $t'_i$ and $(u_i, v_i)$ are the corresponding occurences of $s$ and $(s_1, s_2)$ in $t$ and $(t_1, t_2)$.
2. If $t = F(u)$, $t_1 = F(u_1)$ and $t_2 = F(u_2)$, then the corresponding occurences of $s$ and $(s_1, s_2)$ in $u$ and $(u_1, v_2)$ are the corresponding occurences of $s$ and $(s_1, s_2)$ in $t$ and $(t_1, t_2)$.
3. If $t = s$, $t_1 = s_1$ and $t_2 = s_2$, then $t$, $(t_1, t_2)$ are the corresponding occurences of $s$ and $(s_1, s_2)$ in $t$ and $(t_1, t_2)$.
4. Otherwise, there are no corresponding occurences of $s$ and $(s_1, s_2)$ in $t$ and $(t_1, t_2)$.

**Proposition 6.27** If $t \equiv_{\mathcal{I}} (t_1, t_2)$, and there exists $\langle (s_1, s_2), s \rangle \in \mathcal{I}$ such that there are no corresponding occurences of $s$ and $(s_1, s_2)$ in $t$ and $(t_1, t_2)$, then $t \equiv_{\mathcal{I}'} (t_1, t_2)$, where $\mathcal{I}' = \mathcal{I} - \{\langle (s_1, s_2), s_3 \rangle\}$.

**Proof** If $s$ and $(s_1, s_2)$ have no corresponding occurences in $t$ and $(t_1, t_2)$ by assumption, therefore for all sort symbols $r$, $r_1$ and $r_2$ such that $r$ and $(r_1, r_2)$ do have corresponding occurences in $t$ and $(t_1, t_2)$, there exists an element $\langle (r_1, r_2), r\rangle \in \mathcal{I}'$ which ensures the equivalence of $t$ and $(t_1, t_2)$ in the co-matching $\mathcal{I}'$. $\square$

**Proposition 6.28** Let $t \equiv_{\mathcal{I}} (t_1, t_2)$, and let $\mathcal{I}' = \mathcal{I} - \{\langle (s_1, s_2), s\rangle\}$. Let $t'$, $t_1'$ and $t_2'$ be the terms obtained by replacing each corresponding occurence of $s$ and $(s_1, s_2)$ in $t$ and $(t_1, t_2)$ by terms $p$, $p_1$ and $p_2$ such that $p \equiv_{\mathcal{I}'} (p_1, p_2)$. Then, $t' \equiv_{\mathcal{I}'} (t_1', t_2')$.

**Proof** By well-founded structural induction on $t$ and $(t_1, t_2)$.

*case(i)* If $t = (w_1, \ldots, w_j)$, $t_1 = (u_1, \ldots, u_j)$ and $t_2 = (v_1, \ldots, v_j)$, then $\forall 0 < i \leq j \,.\, w_i \equiv_{\mathcal{I}} (u_i, v_i)$ by definition 6.24. For all $0 < i \leq j$, let $w_i'$, $u_i'$ and $v_i'$ be the terms obtained by replacing each corresponding occurence of $s$ and $(s_1, s_2)$ in $w_i$ and $(u_i, v_i)$ by terms $p$, $p_1$ and $p_2$ such that $p \equiv_{\mathcal{I}'} (p_1, p_2)$. Then, for all $0 < i \leq j$, $w_i' \equiv_{\mathcal{I}'} (u_i', v_i')$ by the induction hypothesis, so $(w_1', \ldots, w_j') \equiv_{\mathcal{I}'} ((u_1', \ldots, u_n'), (v_1', \ldots, v_n'))$ by definition 6.24.

*case(ii)* If $t = F(u)$, $t_1 = F(u_1)$ and $t_2 = F(u_2)$, then $u \equiv_{\mathcal{I}} (u_1, u_2)$ by definition 6.24. Let $u'$, $u_1'$ and $u_2'$ be the terms obtained by replacing each corresponding occurence of $s$ and $(s_1, s_2)$ in $u$ and $(u_1, u_2)$ by $p$, $p_1$ and $p_2$ such that $p \equiv_{\mathcal{I}'} (p_1, p_2)$. Therefore by the induction hypothesis, $u' \equiv_{\mathcal{I}'} (u_1', u_2')$, so that by definition 6.24, $F(u') \equiv_{\mathcal{I}'} (F(u_1'), F(u_2'))$.

*case(iii)* If $t = s$, $t_1 = s_1$ and $t_2 = s_2$, then the proposition holds by definition.

*case(iv)* Otherwise, there are no corresponding occurences of $s$ and $(s_1, s_2)$ in $t$ and $(t_1, t_2)$, so in this case the proposition holds by the preceding proposition. $\square$

**Lemma 6.29** For all $(s_1, s_2, \mathcal{S}, \mathcal{I})$ such that $\mathit{ivalid}(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds, if $\mathit{intersect}(s_1, s_2, \mathcal{S}, \mathcal{I})$ returns $(s', \mathcal{S}')$, then for all $\Theta$ and terms $t_1$ and $t_2$, if there exists a co-derivation $(\Theta(s_1) \Rightarrow^*_{\mathcal{S}} t_1, \Theta(s_2) \Rightarrow^*_{\mathcal{S}} t_2)$ in the application set of $\Theta$ and the keys in $\mathcal{I}$, there exists a derivation $\Theta(s') \Rightarrow^*_{\mathcal{S}'} t'$, such that $t' \equiv_{\Theta(\mathcal{I})} (t_1, t_2)$.

**Proof** By well-founded induction on $n$, the length of the co-derivation.

**Induction Hypothesis** The lemma holds for all $m < n$.

Suppose that for some $(s_1, s_2, \mathcal{S}, \mathcal{I})$, $\mathit{ivalid}(s_1, s_2, \mathcal{S}, \mathcal{I})$ holds, and $\mathit{intersect}(s_1, s_2, \mathcal{S}, \mathcal{I})$ returns $(s', \mathcal{S}')$. Let $\mathcal{P}$ be the application set of $\Theta$ and the keys in $\mathcal{I}$, and let $\mathcal{I}' = \Theta(\mathcal{I})$, and suppose that for terms $t_1$ and $t_2$, $(\Theta(s_1) \Rightarrow^n_{\mathcal{S}} t_1, \Theta(s_2) \Rightarrow^n_{\mathcal{S}} t_2)$ is a co-derivation in $\mathcal{P}$. Then it is sufficient to demonstrate that there exists a derivation $\Theta(s') \Rightarrow^*_{\mathcal{S}'} t'$ such that $t' \equiv_{\mathcal{I}'} (t_1, t_2)$. By proposition 6.20, the proof proceeds by the following cases:

*case(i)* $s_1 = s_2$, so that $\Theta(s_1) = \Theta(s_2)$, and therefore by definition 6.18, $t_1$ and $t_2$ are the same value $\mathbf{c}$. But in this case *intersect* returns $(s_1, \mathcal{S})$, and since $\Theta(s_1) \Rightarrow^*_{\mathcal{S}} \mathbf{c}$ by assumption, therefore $\Theta(s') \Rightarrow^*_{\mathcal{S}'} \mathbf{c}$, and $\mathbf{c} \equiv_{\mathcal{I}'} (\mathbf{c}, \mathbf{c})$ by definition 6.24, so the lemma holds in this case.

*case (ii)* $s_1 = s^1_1 * \ldots * s^1_j$ and $s_2 = s^2_1 * \cdots * s^2_j$, so that $t_1 = (u_1, \ldots, u_j)$ and $t_2 = (v_1, \ldots, v_j)$, and for all $0 < i \leq j$, $(\Theta(s^1_i) \Rightarrow^{m_i}_{\mathcal{S}} u_i, \Theta(s^2_i) \Rightarrow^{m_i}_{\mathcal{S}} v_i)$ is a co-derivation in $\mathcal{P}$. Thus, by the definition of *intersect*, $s' = s'_1 * \cdots * s'_j$, where $\forall 0<i\leq j \;.\; (s'_i, \mathcal{S}_i) = \mathit{intersect}(s^1_i, s^2_i, \mathcal{S}, \mathcal{I})$, and $\mathcal{S}' = \cup^j_{i=1}\mathcal{S}_i$. But for all $0 < i \leq j$, $\mathit{ivalid}(s^1_i, s^2_i, \mathcal{S}, \mathcal{I})$ holds by lemma 6.4, and clearly $m_i < n$. Therefore by the induction hypothesis and lemma 6.8, for all $0 < i \leq j$, there exists a derivation $\Theta(s'_i) \Rightarrow^*_{\mathcal{S}'} t'_i$ such that $t'_i \equiv_{\mathcal{I}'} (u_i, v_i)$. But then by definitions 3.3 and 3.14, $\Theta(s') \Rightarrow^*_{\mathcal{S}'} (t'_1, \ldots, t'_j)$, and where $t' = (t'_1, \ldots, t'_j)$, $t' \equiv_{\mathcal{I}'} (t_1, t_2)$ by definition 6.24, so the lemma holds in this case.

*case (iii)* $s_1$ and $s_2$ are intermediate constructed sorts, $(\Theta(s_1), \Theta(s_2)) \in \mathcal{P}$, and $t_1 = \Theta(s_1)$ and $t_2 = \Theta(s_2)$. But if $(\Theta(s_1), \Theta(s_2)) \in \mathcal{P}$, then $\exists s_3 \;.\; \langle (s_1, s_2), s_3 \rangle \in \mathcal{I}$, and therefore $(s', \mathcal{S}') = (s_3, \mathcal{S})$ by the definition of *intersect*. And clearly, there exists the identity derivation $\Theta(s_3) \Rightarrow^0_{\mathcal{S}'} \Theta(s_3)$, and $\Theta(s_3) \equiv_{\mathcal{I}'} (\Theta(s_1), \Theta(s_2))$ by definition 6.24, so the lemma holds in this case.

*case (iv)* $s_1$ and $s_2$ are intermediate constructed sorts $(r\alpha_1, \ldots, r\alpha_j)\eta_1$ and $(r\alpha_1, \ldots, r\alpha_j)\eta_2$ respectively, $(\Theta(s_1), \Theta(s_2)) \notin \mathcal{P}$ and $t_1$ and $t_2$ are either the same constant $C$, or $t_1$ and $t_2$ are value constructor terms $F(u_1)$ and $F(u_2)$. Let $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_1 = \Omega_1$ and $\mathcal{S} \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_2 = \Omega_2$. In this case, $s'$ is a newly defined intermediate sort $(r\alpha_1, \ldots, r\alpha_j)\eta_f$, and $\mathcal{S}' \vdash (r\alpha_1, \ldots, r\alpha_j)\eta_f = \Omega_f$, where $\Omega_f$ is a set of clauses computed by *intersect*.

Suppose on the one hand that $t_1$ and $t_2$ are the same constant $C$. Then since $(\Theta(s_1) \Rightarrow_{\mathcal{S}} C, \Theta(s_2) \Rightarrow_{\mathcal{S}} C)$ are valid derivation steps by definition 6.12, $C \in \Omega_1, \Omega_2$. But therefore, by the definition of *intersect*, $C \in \Omega_f$, and therefore by definition 3.14, $\Theta((r\alpha_1, \ldots, r\alpha_j)\eta_f) \Rightarrow_{\mathcal{S}'} C$, and clearly $C \equiv_{\mathcal{I}'} (C, C)$, by definition 6.24.

Suppose on the other hand that $t_1$ and $t_2$ are value constructor terms $F(u_1)$ and $F(u_2)$. Then by proposition 6.21, there exists $F(g_1) \in \Omega_1$, $F(g_2) \in \Omega_2$ such that $(\Theta(g_1) \Rightarrow^*_{\mathcal{S}} u_1, \Theta(g_2) \Rightarrow^*_{\mathcal{S}} u_2)$ is a co-derivation in $\mathcal{P}$. Let $\mathcal{S}_f = \mathcal{S} \cup \{(r\alpha_1, \ldots, r\alpha_j)\eta_f = \circ\}$, and $\mathcal{I}_f = \mathcal{I} \cup \{\langle (s_1, s_2), (r\alpha_1, \ldots, r\alpha_j)\eta_f \rangle\}$. Then by the definition of *intersect*, there exists a value constructor term $F(g') \in \Omega_f$, such that $(g', \mathcal{S}_{g'}) = \mathit{intersect}(g_1, g_2, \mathcal{S}_f, \mathcal{I}_f)$. Let $\mathcal{P}_f$ be the application set of $\Theta$ and $\mathcal{I}_f$, and let $\mathcal{I}'_f = \Theta(\mathcal{I}_f)$. Since $\mathit{ivalid}(g_1, g_2, \mathcal{S}_f, \mathcal{I}_f)$ holds by lemma 4.8, therefore by the induction hypothesis and lemma 6.7, for all co-derivations $(\Theta(g_1) \Rightarrow^m_{\mathcal{S}} v_1, \Theta(g_2) \Rightarrow^m_{\mathcal{S}} v_2)$ in $\mathcal{P}_f$ with $m < n$, there exists a derivation $\Theta(g') \Rightarrow_{\mathcal{S}'} v'$ such that $v' \equiv_{\mathcal{I}'_f} (v_1, v_2)$. The proof then proceeds via the following claim:

***claim:*** There exists a derivation $\Theta(s') \Rightarrow^*_{\mathcal{S}} F(u)$ such that $F(u) \equiv_{\mathcal{I}'} (F(u_1), F(u_2))$.

**proof** By induction on $x$, the number of initial occurences of $(\Theta(s_1), \Theta(s_2))$ in the co-derivation $(\Theta(g_1) \Rightarrow^*_{\mathcal{S}} u_1, \Theta(g_2) \Rightarrow^*_{\mathcal{S}} u_2)$ in $\mathcal{P}$.

**basis** If there are no initial occurences of $(\Theta(s_1), \Theta(s_2))$ in the co-derivation $(\Theta(g_1) \Rightarrow^*_{\mathcal{S}} u_1, \Theta(g_2) \Rightarrow^*_{\mathcal{S}} u_2)$ in $\mathcal{P}$, then by proposition 6.22, $(\Theta(g_1) \Rightarrow^*_{\mathcal{S}} u_1, \Theta(g_2) \Rightarrow^*_{\mathcal{S}} u_2)$ is a co-derivation in $\mathcal{P} \cup \{(\Theta(s_1), \Theta(s_2))\}$; that is, a co-derivation in $\mathcal{P}_f$. And clearly, the co-derivation $(\Theta(g_1) \Rightarrow^*_{\mathcal{S}} u_1, \Theta(g_2) \Rightarrow^*_{\mathcal{S}} u_2)$ is of length less than $n$, so that by the induction hypothesis of the lemma as demonstrated above, $\Theta(g') \Rightarrow^*_{\mathcal{S}} u$ such that $u \equiv_{\mathcal{I}'_f} (u_1, u_2)$. And since there are no initial occurrences of $(\Theta(s_1), \Theta(s_2))$ in the co-derivation $(\Theta(g_1) \Rightarrow^*_{\mathcal{S}} u_1, \Theta(g_2) \Rightarrow^*_{\mathcal{S}} u_2)$, therefore there are no corresponding occurrences of $(\Theta(s_1), \Theta(s_2))$ in $(u_1, u_2)$. And since $\mathcal{I}' = \mathcal{I}'_f - \{\langle(\Theta(s_1), \Theta(s_2)), \Theta(s')\rangle\}$, therefore by proposition 6.27, $u \equiv_{\mathcal{I}'} (u_1, u_2)$, so that $F(u) \equiv_{\mathcal{I}'} (F(u_1), F(u_2))$; and since $F(g') \in \Omega_f$, therefore $\Theta(s') \Rightarrow^*_{\mathcal{S}} F(u)$ in this case by proposition 3.30.

**induction hypothesis** For all co-derivations $(\Theta(s_1) \Rightarrow^m_{\mathcal{S}} v_1, \Theta(s_2) \Rightarrow^m_{\mathcal{S}} v_2)$ in $\mathcal{P}$, with $m < n$, containing fewer than $x$ initial occurences of $(\Theta(s_1), \Theta(s_2))$, there exists a derivation $\Theta(s') \Rightarrow_{\mathcal{S}'} v$ such that $v \equiv_{\mathcal{I}'} (v_1, v_2)$.

**induction step** Suppose that for $x > 0$, there are $x$ initial occurences of $(\Theta(s_1), \Theta(s_2))$ in the co-derivation $(\Theta(g_1) \Rightarrow^*_{\mathcal{S}} u_1, \Theta(g_2) \Rightarrow^*_{\mathcal{S}} u_2)$ in $\mathcal{P}$, which is clearly of length less than $n$. Let $(\Theta(g_1) \Rightarrow^m_{\mathcal{S}} w_1, \Theta(g_2) \Rightarrow^m_{\mathcal{S}} w_2)$ be the simultaneous derivation which is identical to $(\Theta(g_1) \Rightarrow^*_{\mathcal{S}} u_1, \Theta(g_2) \Rightarrow^*_{\mathcal{S}} u_2)$, except that each initial occurence of $(\Theta(s_1), \Theta(s_2))$ is not reduced in any subsequent derivation step. Therefore clearly $(\Theta(g_1) \Rightarrow^m_{\mathcal{S}} w_1, \Theta(g_2) \Rightarrow^m_{\mathcal{S}} w_2)$ is an end-term derivation in $\mathcal{P} \cup \{(\Theta(s_1), \Theta(s_2))\}$; that is, an end-term derivation in $\mathcal{P}_f$, with $m < n$. Therefore by the induction hypothesis of the lemma as demonstrated above, there exists a derivation $\Theta(g') \Rightarrow^*_{\mathcal{S}'} w'$ such that $w' \equiv_{\mathcal{I}'_f} (w_1, w_2)$.

Also, in the co-derivation $(\Theta(g_1) \Rightarrow^*_{\mathcal{S}} u_1, \Theta(g_2) \Rightarrow^*_{\mathcal{S}} u_2)$ in $\mathcal{P}$, each corresponding occurrence of $(\Theta(s_1), \Theta(s_2))$ must be reduced to terms $p_1$ and $p_2$ which are subterms of $u_1$ and $u_2$,and each such reduction in itself comprises a co-derivation $(\Theta(s_1) \Rightarrow^*_{\mathcal{S}} p_1, \Theta(s_2) \Rightarrow^*_{\mathcal{S}} p_2)$ in $\mathcal{P}$ by proposition 6.21. But the length of these co-derivations are clearly less than $n$, and there are fewer than $x$ initial occurences of $(\Theta(s_1), \Theta(s_2))$ in each co-derivation $(\Theta(s_1) \Rightarrow^*_{\mathcal{S}} p_1, \Theta(s_2) \Rightarrow^*_{\mathcal{S}} p_2)$; therefore by the induction hypothesis of the claim, for each such co-derivation there exists a derivation $\Theta(s') \Rightarrow_{\mathcal{S}'} p'$ such that $p' \equiv_{\mathcal{I}'} (p_1, p_2)$.

Let $w'_1$, $w'_2$ and $u$ be the terms obtained by replacing each corresponding occurence of $\Theta(s')$ and $(\Theta(s_1), \Theta(s_2))$ in $w'$ and $(w_1, w_2)$ by $p'$ and $(p_1, p_2)$, such that $w'_1 = u_1$ and $w'_2 = u_2$, and such that $p' \equiv_{\mathcal{I}'} (p_1, p_2)$ and $\Theta(s') \Rightarrow_{\mathcal{S}'} p'$. Thus, by proposition 6.28, $u' \equiv_{\mathcal{I}'} (u_1, u_2)$. And since $\Theta(g') \Rightarrow^*_{\mathcal{S}'} w'$, and $\Theta(s') \Rightarrow_{\mathcal{S}'} p'$ for each $p'$ and for each occurence of $\Theta(s')$ in $\Theta(g') \Rightarrow^*_{\mathcal{S}'} w'$ by the preceding arguments, therefore clearly $\Theta(g') \Rightarrow^*_{\mathcal{S}'} u$. And therefore, by proposition 3.30, $\Theta(s') \Rightarrow^*_{\mathcal{S}'} F(u)$, and $F(u) \equiv_{\mathcal{I}'} (F(u_1), F(u_2))$ by definition 6.24, so the claim holds.

And since the claim holds, that covers all the possibilities for this case, which was the last to be proven, so the lemma holds. □

Now, we are in a position to give the full correctness proof of *intersection*.

**Theorem 6.30** *For all sorts $s_1$ and $s_2$ and environments $\mathcal{S}$ such that $\mathcal{S} \vdash s_1$ and $\mathcal{S} \vdash s_2$, intersection$(s_1, s_2, \mathcal{S})$ terminates, returning $(s', \mathcal{S}')$ such that for all $\Theta$, $[\Theta(s')]_{\mathcal{S}'} = [\Theta(s_1)]_{\mathcal{S}} \cap [\Theta(s_2)]_{\mathcal{S}}$.*

**Proof** By the definition of the algorithm, *intersection*$(s_1, s_2, \mathcal{S})$ returns the result of *intersect*$(s'_1, s'_2, \mathcal{S}_2, \emptyset)$, where $(s'_1, \mathcal{S}_1) = elab(s_1, \mathcal{S}, \langle\rangle, \emptyset)$ and $(s'_2, \mathcal{S}_2) = elab(s_2, \mathcal{S}_1, \langle\rangle, \emptyset)$. Since *evalid*$(s_1, \mathcal{S}, \langle\rangle,$

$\emptyset$) and *evalid*$(s_2, \mathcal{S}_1, \langle\rangle, \emptyset)$ hold by definition 4.5, therefore each of these calls to *elab* terminates, $s_1'$ and $s_2'$ are intermediate sorts, and for all $\Theta$, $[\Theta(s_1')]_{\mathcal{S}_2} = [\Theta(s_1)]_{\mathcal{S}}$ and $[\Theta(s_2')]_{\mathcal{S}_2} = [\Theta(s_2)]_{\mathcal{S}}$ by lemma 4.46 and proposition 4.28. And therefore, since *ivalid*$(s_1', s_2', \mathcal{S}_2, \emptyset)$ holds by definition 6.3, *intersect*$(s_1', s_2', \mathcal{S}_2, \emptyset)$ terminates, by lemma 6.6. Also, since *imatch*$(\mathcal{S}_2, \emptyset)$ holds by definition 6.9, therefore for all $\Theta$, $[\Theta(s')]_{\mathcal{S}'} \subseteq [\Theta(s_1')]_{\mathcal{S}_2} \cap [\Theta(s_2')]_{\mathcal{S}_2}$ by lemma 6.10.

Now, suppose there is some value $\mathbf{c}$ such that $\mathbf{c} \in [\Theta(s_1')]_{\mathcal{S}_2} \cap [\Theta(s_2')]_{\mathcal{S}_2}$. Then by proposition 6.23, there exists an co-derivation $(\Theta(s_1') \Rightarrow^*_{\mathcal{S}_2} \mathbf{c}, \Theta(s_2') \Rightarrow^*_{\mathcal{S}_2} \mathbf{c})$ in $\emptyset$, so that by lemma 6.29, there exists a derivation $\Theta(s') \Rightarrow^*_{\mathcal{S}'} \mathbf{c}'$ such that $\mathbf{c}' \equiv_{\emptyset} (\mathbf{c}, \mathbf{c})$, so that by definition 6.24, $\mathbf{c}' = \mathbf{c}$. Thus, if $\mathbf{c} \in [\Theta(s_1')]_{\mathcal{S}_2} \cap [\Theta(s_2')]_{\mathcal{S}_2}$, then there exists a derivation $\Theta(s') \Rightarrow^*_{\mathcal{S}'} \mathbf{c}$, so that by definition 3.22, $\mathbf{c} \in [\Theta(s')]_{\mathcal{S}'}$, hence $[\Theta(s_1')]_{\mathcal{S}_2} \cap [\Theta(s_2')]_{\mathcal{S}_2} \subseteq [\Theta(s')]_{\mathcal{S}'}$.

And therefore, since *intersection*$(s_1, s_2, \mathcal{S})$ returns $(s', \mathcal{S}')$, and since for all $\Theta$, $[\Theta(s_1')]_{\mathcal{S}_2} = [\Theta(s_1)]_{\mathcal{S}}$ and $[\Theta(s_2')]_{\mathcal{S}_2} = [\Theta(s_2)]_{\mathcal{S}}$, therefore for all $\Theta$, $[\Theta(s')]_{\mathcal{S}'} = [\Theta(s_1)]_{\mathcal{S}} \cap [\Theta(s_2)]_{\mathcal{S}}$, so the theorem holds. $\square$

# 7 Conclusion, future work

This thesis can be summarized as follows: first, a refinement type language was defined, which is an analogue of the ML refinement type language described in [Fre94] and [DP97]. Sort substitutions were introduced, and for any ground sort generated by the application of a substitution to a sort, a method for obtaining a grammar associated with that ground sort was described. The meaning of a ground sort was then defined as the set of terms which can be derived from its associated grammar.

Given this framework, three algorithms were defined and proven correct. First, the *elab* algorithm was given, which carries out the "elaboration" of a sort into its associated grammar, to the extent that the possibly polymorphic nature of the sort allows. Along with the termination, soundness and completeness results given for the algorithm, *elab* provides a constructive proof of the "regularity" of sort grammars. Also, since intermediate sorts are normalized data structures, the intermediate sort returned by *elab* has properties which simplify the definition and proof of correctness for the subsequent *emptysort* and *intersection* algorithms.

The *emptysort* algorithm decides whether a given sort is empty. The *intersection* algorithm takes two sorts and returns an intermediate sort which corresponds to the intersection of those two sorts. Given the approriate termination and correctness results, these algorithms demonstrate that the emptiness and intersection problems for sorts are decidable.

Since the refinement type checking algorithm requires the definition of subset and intersection algorithms for the purposes of inferring a complete sort lattice [DP97], it is hoped that the intersection algorithm given in this thesis will contribute to the implementation of the refinement type-checking algorithm, or at least to a justification of its correctness. Also, with the emptiness judgement given in this thesis, it is possible to decide the subset relation by composition with a difference algorithm, since $s_1 \leq s_2$ iff the difference of $s_1$ and $s_2$ is empty. Indeed, it was originally intended that this thesis would provide a difference algorithm, which would be combined with the emptiness judgement to decide the subset relation; however, the correctness proof for this algorithm proved to be beyond the scope of this thesis.

Thus, appropriate future work would surely be a proof of correctness for the difference algorithm. Also, it is probably the case that a more sophisticated mechanism than linear derivations can be provided for describing a model of sorts. Results like theorem 3.20 and proposition 6.21, and especially the very idea of "co-derivations", are clumsy workarounds, testament to the subtlety that is lost by describing tree-structured objects in a linear fashion. Gecseg and Steinby in [GS84] take an approach that is much more in keeping with the non-linear structure of regular terms, and future theoretical considerations of sorts, and decision procedures for sorts, would most likely benefit from a similar approach. Thus, a new semantic interpretation of sorts, reflecting the non-linear structure of sort terms, would be appropriate.

# References

[DP97] Rowan Davies and Frank Pfenning. *Practical Refinement-Type Checking.* Unpublished manuscript, 1997.

[FP91] Tim Freeman and Frank Pfenning. Refinement Types for ML. *Proceedings of the* SIGPLAN '91 *Symposium on Language Design*, pages 268-277. ACM press, Toronto, Ontario, 1991.

[Fre94] Tim Freeman. Refinement Types for ML. Technical Report CMU-CS-94-110, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1994.

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[DZ92] Philip W. Dart and Justin Zobel. A Regular Type Language for Logic Programs. In Frank Pfenning, editor, *Types in Logic Programming*, pages 157-188. MIT Press, Cambridge, Massachusetts, 1992.

[MTH90] Robin Milner, Mads Tofte and Robert Harper. *The Definition of Standard ML.* MIT Press, Cambridge, Massachusetts, 1990.

[Pau96] Lawrence C. Paulson. *ML for the Working Programmer.* Cambridge University Press, Cambridge, 1996.

[CW85] Luca Cardelli and Peter Wegner. On Understanding types, data abstraction and polymorphism. *ACM Computing Surveys*, 17:471-522, 1985.

[Llo84] John W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1984.