A Method to Translate Order-Sorted Algebras to Many-Sorted Algebras

Liyi Li and Elsa Gunter
Department of Computer Science,
University of Illinois at Urbana-Champaign
{liyili2,egunter}@illinois.edu

Order-sorted algebras and many sorted algebras exist in a long history with many different implementations and applications. A lot of language specifications have been defined in order-sorted algebra frameworks such as the language specifications in $\mathbb{K}$ (an order-sorted algebra framework). The biggest problem in a lot of the order-sorted algebra frameworks is that even if they might allow developers to write programs and language specifications easily, but they do not have a large set of tools to provide reasoning infrastructures to reason about the specifications built on the frameworks, which are very common in some many-sorted algebra framework such as Isabelle/HOL [24], Coq [6] and FDR [27]. This fact brings us the necessity to marry the worlds of order-sorted algebras and many sorted algebras. In this paper, we propose an algorithm to translate a *strictly sensible* order-sorted algebra to a many-sorted one in a restricted domain by requiring the order-sorted algebra to be *strictly sensible*. The key idea of the translation is to add an equivalence relation called *core equality* to the translated many-sorted algebras. By defining this relation, we reduce the complexity of translating a *strictly sensible* order-sorted algebra to a many-sorted one, make the translated many-sorted algebra equations only increasing by a very small amount of new equations, and keep the number of rewrite rules in the algebra in the same amount. We then prove the order-sorted algebra and its translated many-sorted algebra are bisimilar. To the best of our knowledge, our translation and bisimilar proof is the first attempt in translating and relating an order-sorted algebra with a many-sorted one in a way that keeps the size of the translated many-sorted algebra relatively small.

1 Motivation

Currently, order-sorted algebras are used widely in defining specifications and programs. Maude [4] and $\mathbb{K}$ [25] are successful programming languages for defining order-sorted algebras. The specifications of a lot of popular programming languages, such as Java [3], Javascript [23], PHP [10], C [9, 13] and LLVM [15] semantics, have been defined in $\mathbb{K}$. Experience shows that order-sorted algebras allow users to define specifications easily. In the paper [23], Park et al. show how they can define the full semantics of Javascript by using $\mathbb{K}$ in only three months.

On the other hand, many-sorted algebras also have wide usage. Many people define pieces of popular programming languages such as C, Java, LLVM and Javascript in forms of many-sorted algebras. For example, people define specifications based on many-sorted algebras in some interactive theorem provers, such as Isabelle/HOL [24] and Coq [6], where people commonly use their many-sorted type theories to prove properties about language specifications. The advantage of using many-sorted algebra based frameworks is that they usually associate with a large amount of tools and applications for users to prove

Acknowledgments. This material is based upon work supported in part by NSF Grant 0917218. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

H. Cirstea, D. Sabel (Eds.): Fourth International Workshop on Rewriting Techniques for Program Transformations and Evaluation (WPTE 2017)
EPTCS 265, 2018, pp. 20–34, doi:10.4204/EPTCS.265.3

properties about the programs or language specifications they define, such as the tools set of Isabelle/HOL [24], Coq [6] and FDR [27].

In order to connect these two worlds, especially to connect the existing programming language semantic specifications defined in the order-sorted algebra $\mathbb{K}$ with the traditional theorem provers such as Isabelle/HOL and Coq, the key is to discover a way to translate an order-sorted algebra into a many-sorted algebra. The reason we want to do this is to use the theorem proving engines and their existing toolsets to develop theories about specifications defined in the order-sorted world. Please note that the syntax of the specification that we are interested in translating from a order-sorted form to a many-sorted form is an abstract syntax, not a concrete syntax of a language. Even though users are allowed to define mixfix syntax in order-sorted programming languages such as $\mathbb{K}$ or Maude, they are still representing the abstract syntax and not the concrete syntax of a specification because the mixfix syntax forms are just syntactic sugars in $\mathbb{K}$ and Maude to write abstract syntax for a specification. For example, both $\mathbb{K}$ and Maude do not allow users to create overloaded constants.

To the best of our knowledge, the most recent and relevant work on defining a translation mechanism is that of Meseguer and Skeirik [21], who created an algorithm to translate an initial free order-sorted algebra (algebras without equations and rules) to a many-sorted one. They only propose a naive algorithm to translate a general and *sensible* order-sorted algebra (an order-sorted algebra are usually *sensible*) to a many-sorted one. However, this naive algorithm deals with the most general cases, so it adds a lot more sorts and rewrite rules than needed in more restricted cases. In some order-sorted algebras, if some rewrite rules have many sorts and the sorts have many subsorts, it can cause their algorithm to generate exponentially many rewrite rules. Even though the chance of this extreme situation is rare for a normal order-sorted algebra, their algorithm squares or cubes the number of equations and rewrite rules when they translate an order-sorted algebra to a many-sorted one, which is not desirable.

Our main goal is to marry the world of people defining language specifications using order-sorted algebras with that of people using theorem provers to develop theories about language specifications by using many-sorted algebras. In order to succeed, our translation of an order-sorted algebra must be understood by the people who are using the theorem provers. Making a many-sorted algebra with relatively the same number of rewrite rules would significantly reduce the users' efforts to understand the translated language specifications. That is the reason for us to present a way to translate an interesting subset of order-sorted algebras into many-sorted algebras that increase the number of the equations by less than a linear factor and keep the number of rewrite rules the same.

By requiring the target order-sorted algebra to be *strictly sensible*, the basic idea of our algorithm is to view the subsort relation $s \leq s'$ defined in an order-sorted algebra as the implicit coercion of a term in the subsort s to a term in the supersort s'. Then, we borrow the idea of constructors as a way of explicit coercion from other functional programming languages, such as Standard ML [22]. We add an explicit coercion with a constructor for each subsort relation and view these subsort relations as unary operators in the translated many-sorted algebra. After that, we add a new equivalence relation for operators, which we call *core equality*. *Core equality* allows users to equate two terms as long as their core parts (not counting the generated subsort unary operator parts) are the same. By this translation process, we are able to translate a valuable subset of order-sorted algebras into many-sorted ones. Specifically, we are able to translate all those valuable language specifications in $\mathbb{K}$ mentioned above into ones in Isabelle/HOL.

It is worth noting that the reason for us to require the target order-sorted algebra to be *strictly sensible* is that we want to outline a subset of order-sorted algebras that can be translated into some many-sorted algebras easily and concisely, as well as being able to prove the bi-simulation between the order-sorted algebras and the translated many-sorted ones in this case. Since there is a naive algorithm proposed by Meseguer and Skeirik to translate a general and *sensible* order-sorted algebra to a many-sorted one, by a

little engineering work, one can always divide an order-sorted algebra into a part that is *strictly sensible* and another part that is not *strictly sensible* but *sensible*, and translate the first part by using our algorithm and the second part by using the naive algorithm. We do not specify the engineering task in this paper, because we want to focus on the theories of discovering a subset of order-sorted algebras that can be translated into many-sorted ones easily and concisely.

2 The Scope of the Solution

In this section, we describe the preliminaries related to the problem. The basic idea is to find a translation function tr to translate an order-sorted algebra to a many-sorted one and preserve the meaning of the former one in the latter one. We first define term algebras for many-sorted algebras and order sorted algebras in Definitions 2.1 and 2.2, respectively. A term algebra is a trivial algebra that defines the terms allowed in an algebra without variables.

Definition 2.1. A *sorted signature* is a tuple of (S,Φ,Σ), where S is a set of sorts, Φ is a finite set of constructors, and Σ is the set of all operators in the system, where an operator is of the form $f : s_1 \times ... \times s_n \to s$, where f is a constructor defined in set Φ, $s_1,...,s_n$ is a list of argument sorts and s is the target sort. Sorts $s_1,...,s_n$ and s are elements of set S. Sometimes we use Σ to refer to the signature. The *sorted ground term algebra* of (S,Φ,Σ) is the set of terms T_Σ equal to $\cup_{s\in S}(T_{\Sigma,s})$, where the sets $(T_{\Sigma,s})$ are mutually defined by:

(1) For each operator $a : nil \to s \in \Sigma$, the constructor $a \in T_{\Sigma,s}$, where *nil* means that the argument sort list of the operator is an empty list.

(2) For each non-zero arity operator $f : w \to s \in \Sigma$, where $w = s_1 \times ... \times s_n$ and $n > 0$, and for each $(t_1,...t_n) \in T_{\Sigma,s_1} \times ... \times T_{\Sigma,s_n}$, the term $f(t_1,...,t_n) \in T_{\Sigma,s}$.

Definition 2.2. An *order-sorted signature* is a tuple (S,O,Φ,Σ), where (S,Φ,Σ) is a *sorted signature* and the set O is a set of pairs of sorts, such that its reflexive and transitive closure $\leq$ forms a partial order. This means that O cannot have cycles if we view the pairs of O as defining a directed graph. The poset $(S,\leq)$ represents the subsort relations of the system. The *order-sorted ground term algebra* of (S,Φ,Σ) is the least set of terms T_Σ equal to $\cup_{s\in S}(T_{\Sigma,s})$, where the sets $(T_{\Sigma,s})$ are mutually defined by:

(1) For each operator $a : nil \to s \in \Sigma$, the constructor $a \in T_{\Sigma,s}$, where *nil* means that the argument sort list of the operator is an empty list.

(2) For each non-zero arity operator $f : w \to s \in \Sigma$, where $w = s_1 \times ... \times s_n$ and $n > 0$, and for each $(t_1,...t_n) \in T_{\Sigma,s_1} \times ... \times T_{\Sigma,s_n}$, the term $f(t_1,...,t_n) \in T_{\Sigma,s}$.

(3) If $s \leq s'$, then $T_{\Sigma,s} \subseteq T_{\Sigma,s'}$.

In Figure 1, we show an example of an order-sorted signature: IMP, and list the sets of S, O, Φ and Σ accordingly. Based on the signature, the order-sorted ground term algebra T_Σ can be generated by the rules in Definition 2.2. If we drop the O set, the signature becomes a sorted signature, and we can generate the sorted ground term algebra T_Σ by the rules in Definition 2.1. In our version of the IMP language, we assume that all identifiers in a given term have been initialized. To make the IMP language simple enough, we do not provide semantics for how to lookup the value for an identifier. Instead, we assume that there is a ***guess*** function that will guess a value for an identifier, which happens to be the same as the value previously defined for the identifier. Finally, we use the operator $-$ to mean both an integer negative sign and a negation of a boolean formula, as well as $+$ to mean both an arithmetic addition operator and a conjunctive boolean operator, in order to show how we deal with overloaded operators.

S: $\{\texttt{nat}, \texttt{int}, \texttt{AExp}, \texttt{Id}, \texttt{bool}, \texttt{BExp}, \texttt{Block}, \texttt{Stmt}, \texttt{Map}, \texttt{Pgm}\}$

O: $\{\texttt{nat} < \texttt{int}, \texttt{int} < \texttt{AExp}, \texttt{Id} < \texttt{AExp}, \texttt{bool} < \texttt{BExp}, \texttt{Block} < \texttt{Stmt}\}$

Φ: $\{\boldsymbol{v}, \boldsymbol{true}, \boldsymbol{false}, 0, \boldsymbol{s}, +, -, <=, \{\}, \{\_\}, \_ = \_;, \_\_, \boldsymbol{if\_else}, \_,\_, \boldsymbol{.Map}, \boldsymbol{guess}, < \_,\_ >, \_[\_/\_], \_ \mapsto \_\}$

Σ: $\{\boldsymbol{true} :\to \texttt{bool},\ \boldsymbol{false} :\to \texttt{bool},\ 0 :\to \texttt{nat},\ \boldsymbol{s} : \texttt{nat} \to \texttt{nat},\ - : \texttt{int} \to \texttt{int},\ - : \texttt{nat} \to \texttt{int},$
$+ : \texttt{AExp} * \texttt{AExp} \to \texttt{AExp},\ + : \texttt{nat} * \texttt{nat} \to \texttt{AExp},\ + : \texttt{int} * \texttt{int} \to \texttt{AExp},\ \{\} :\to \texttt{Block},$
$<= : \texttt{AExp} * \texttt{AExp} \to \texttt{BExp},\ - : \texttt{BExp} \to \texttt{BExp},\ - : \texttt{bool} \to \texttt{BExp},\ + : \texttt{bool} * \texttt{bool} \to \texttt{BExp},$
$\boldsymbol{v} : \texttt{nat} \to \texttt{Id},\ \{\_\} : \texttt{Stmt} \to \texttt{Block},\ \_ = \_; : \texttt{Id} * \texttt{AExp} \to \texttt{Stmt},\ \_\_ : \texttt{Stmt} * \texttt{Stmt} \to \texttt{Stmt},$
$\boldsymbol{if\_else} : \texttt{BExp} * \texttt{Block} * \texttt{Block} \to \texttt{Stmt},\ \boldsymbol{guess} : \texttt{Id} \to \texttt{int},$
$< \_,\_ > : \texttt{Map} * \texttt{Stmt} \to \texttt{Pgm},\ \_,\_ : \texttt{Map} * \texttt{Map} \to \texttt{Map},\ \boldsymbol{.Map} :\to \texttt{Map},\ + : \texttt{BExp} * \texttt{BExp} \to \texttt{BExp},$
$\_[\_/\_] : \texttt{Map} * \texttt{int} * \texttt{Id} \to \texttt{Map},\ \_ \mapsto \_ : \texttt{Id} * \texttt{int} \to \texttt{Map}\}$

Figure 1: IMP Signature

Based on the ground term algebra T_Σ, we define the terms with variables as $T_\Sigma(X)$. Given a term with variables $t(X) \in T_\Sigma(X)$, where all variables in $t(X)$ are contained in X, term $t \in T_\Sigma$ is an *instance* of $t(X)$ if there exists a substitution h mapping X to T_Σ such that t is the result of replacing each variable x in $t(X)$ by $h(x)$. Every variable in a term in $T_\Sigma(X)$ is represented by a name. Even though we refer to T_Σ and $T_\Sigma(X)$ as term algebras in both many-sorted algebras and order sorted algebras, They are sorted term algebras in the many-sorted world and order sorted term algebras in the order sorted world. It is worth noting that, while Σ contains sort information, T_Σ and $T_\Sigma(X)$ do not. A mapping function $x : s$ maps a variable x to a sort s representing the target sort of x. We now define a many-sorted algebra and an order sorted-algebra in Definitions 2.3 and 2.4, respectively.

Definition 2.3. A *many-sorted algebra* B is defined by a tuple (S, Φ, Σ, E, R), where S is a set of sorts, Φ is the finite set of constructors allowed in the system, Σ represents all operators in the system and (S, Φ, Σ) is the many-sorted signature, which we refer to as Σ. The equation set E is a set of pairs of terms in $T_\Sigma(X)$. and partitions the terms of B, T_Σ, into equivalence classes, denoted $T_{(\Sigma,E)}$. The terms allowed to construct each equation in E are in sorted term algebra $T_\Sigma(X)$, while the equations are applied on the terms in the sorted ground term algebra T_Σ. We introduce the quotient structure $T_{(\Sigma,E)}$, which we call terms T_Σ modulo equations E. For two terms t and t' in T_Σ, if we can prove they are equal through the equations E, we say these two terms are equivalent modulo E, which partitions T_Σ into different equivalence classes and forms $T_{(\Sigma,E)}$. A set of rewrite rules R defines the semantics of system B. The rule set R is a set of pairs of terms in $T_\Sigma(X)$, while the rules are applied on the terms in $T_{(\Sigma,E)}$. If a rule $r \in R$ is applied to a term t, it means that the rule r is applied on the class $c \in T_{(\Sigma,E)}$ where $t \in c$ and t is the representative of c. The transition $c \longrightarrow_r c'$ means that for $t(X)$ as the left hand side and $t'(X)$ as the right hand side of rule r, there is a substitution h mapping X to T_Σ such that t and t' are the result of replacing each variable x in $t(X)$ and $t'(X)$ by $h(x)$ and $t \in c$ and $t' \in c'$, respectively. The rule r generates an endomorphic relation, and applications of rules are closed under applications of constructors.

Definition 2.4. An *order-sorted algebra* A is a tuple $(S, O, \Phi, \Sigma, E, R)$, where (S, Φ, Σ) is a many-sorted signature, and O is a set of pairs of sorts, such that the reflexive and transitive closure $\leq$ forms a partial order. The poset $(S, \leq)$ represents the subsort relations of the system. We call (S, O, Φ, Σ) the signature of the system, which we refer to as Σ. The terms allowed to construct each equation in E are in the order-sorted term algebra $T_\Sigma(X)$, while the equations are applied on the terms in the order-sorted ground term algebra T_Σ. A set of rewrite rules R defines the semantics of system B. The rule set R is a set of pairs

of terms in $T_\Sigma(X)$, while the rules are applied on the terms in $T_{(\Sigma,E)}$. The two elements of a pair in E are required to have the same sort, while for any pair (t,t') in R, the sort of t' is a subsort of the sort of t. This property is called *sort decreasing*. We introduce the quotient structure $T_{(\Sigma,E)}$, which we call terms T_Σ modulo equations E. For two terms t and t' in T_Σ, if we can prove they are equal through the equations E, we say these two terms are equivalent modulo E, which partitions T_Σ into different equivalence classes and forms $T_{(\Sigma,E)}$. A set of rewrite rules R defines the semantics of system B. The rule set R is a set of pairs of terms in the order-sorted term algebra $T_\Sigma(X)$, while the rules are applied on the terms in $T_{(\Sigma,E)}$. If a rule $r \in R$ is applied to a term t, it means that the rule r is applied on the class $c \in T_{(\Sigma,E)}$ where $t \in c$ and t is the representative of c. The transition $c \longrightarrow_r c'$ means that for $t(X)$ as the left hand side and $t'(X)$ as the right hand side of rule r, there is a substitution h mapping X to T_Σ such that t and t' are the result of replacing each variable x in $t(X)$ and $t'(X)$ by $h(x)$ and $t \in c$ and $t' \in c'$, respectively. The rule r generates an endomorphic relation, and applications of rules are closed under applications of constructors.

The $\leq$ relation can be viewed as a directed graph where each relation is an edge. The graph may have different connected components. For any two sorts in a connected component in an order-sorted algebra, we require there is a unique top *supersort* of them.

In Figure 2, we show the equations and rules for the order-sorted algebra IMP. With the information in Figure 1, this information constructs a well-defined order-sorted algebra. A many-sorted algebra is similar to this one with more restrictions. For example, the left hand side and right hand side of a rule need to be sort equivalent in a many-sorted algebra. In order to write an equation for the operator $+$, we need to write three versions of equations: one for $+$ with argument sorts $\texttt{AExp} * \texttt{AExp}$, one for it with argument sorts $\texttt{int} * \texttt{int}$ and one for argument sorts $\texttt{nat} * \texttt{nat}$.

In a many-sorted algebra and order-sorted algebra, even though the terms that are used to construct an equation or a rule is in the form of $T_\Sigma(X)$, they are representatives of equivalence classes in $T_{(\Sigma,E)}$. One thing to keep in mind is that we are defining algebras in this paper, not transition systems. The rewrite rules in an algebra can be applied to any subterm of a given term, not only to its top-most operator. This idea is similar to the rewrite rules in Rewriting Logic [16]. Based on the order-sorted algebra definition, the only input restriction of our translation function tr is that the order-sorted algebra A should be, not just *sensible*, but *strictly sensible*. The former term is defined in Definition 2.6, while the latter is defined in Definition 2.7. One thing about *overloaded operators* (two operators having the same constructor) in an algebra is that if the two overloaded operators f and f' have argument sorts that have no common supersorts, we treat them as different operators since they can be easily distinguished by combining the constructor and the list of argument sorts.

Definition 2.5. We define two overloaded operators f and f' to be *argument compatible*, if they have the same arities, and f has argument sorts $s_1,...,s_n$, and f' has argument sorts $s'_1,...,s'_n$, and $s_i \equiv_\leq s'_i$ for $i = 1,...,n$, where $\equiv_\leq$ means that the two given sorts have a common supersort.

Definition 2.6. (Goguen and Meseguer [12]) An order-sorted algebra is *sensible*, if for any pair of argument compatible constructors f and f' with target sorts s and s', respectively, we have $s \equiv_\leq s'$.

Definition 2.7. An order-sorted algebra is *strictly sensible* if:

(1) Whenever there are two argument compatible operators f and f' with target sorts s and s', respectively, then we have $s = s'$. We then call the order-sorted algebra being *strong sensible*. It is worth noting that a strong sensible algebra cannot have overloaded constant operators.

(2) For each operator f, there exists an operator $f' : s_1 \times ... \times s_n \to s$, such that for every operator f'' being argument compatible with f, f' is argument compatible with f'', and if f'' has argument sorts $s'_1 \times ... \times s'_n$, then $s'_i \leq s_i$ for all $i = 1,...,n$. We then call the order-sorted algebra being *maximal argument-bounding*.

$$
\begin{array}{ll}
E: & \{0 + A : \texttt{AExp} = A : \texttt{AExp},\ \boldsymbol{s}(A : \texttt{nat}) + B : \texttt{nat} = A : \texttt{nat} + \boldsymbol{s}(B : \texttt{nat}),\ -\,-A : \texttt{int} = A : \texttt{int},\\
& A : \texttt{AExp} + B : \texttt{AExp} = B : \texttt{AExp} + A : \texttt{AExp},\\
& \boldsymbol{s}(A : \texttt{nat}) + -\boldsymbol{s}(B : \texttt{nat}) = A : \texttt{nat} + B : \texttt{nat},\ \boldsymbol{true} + A : \texttt{BExp} = A; \texttt{BExp},\\
& A : \texttt{BExp} + B : \texttt{BExp} = B : \texttt{BExp} + A : \texttt{BExp},\ \_,\_(A : \texttt{Map}, B : \texttt{Map}) = \_,\_(B : \texttt{Map}, A : \texttt{Map}),\\
& \boldsymbol{s}(A : \texttt{nat}) <= B : \texttt{AExp} = 0 <= B : \texttt{AExp} + -\boldsymbol{s}(A : \texttt{nat}),\ \_,\_(A : \texttt{Map}, \boldsymbol{.Map}) = A : \texttt{Map},\\
& -\boldsymbol{s}(A : \texttt{nat}) <= B : \texttt{AExp} = 0 <= B : \texttt{AExp} + \boldsymbol{s}(A : \texttt{nat}),\\
& \_,\_(A : \texttt{Map}, \_,\_(B : \texttt{Map}, C : \texttt{Map})) = \_,\_(\_,\_(A : \texttt{Map}, B : \texttt{Map}), C : \texttt{Map}),\\
& \_[\_/\_](\boldsymbol{.Map}, A : \texttt{int}, B : \texttt{Id}) = B : \texttt{Id} \mapsto A : \texttt{int},\\
& \_[\_/\_](\_,\_(A : \texttt{Id} \mapsto B : \texttt{int}, C : \texttt{Map}), D : \texttt{int}, A : \texttt{Id}) = \_,\_(A : \texttt{Id} \mapsto D : \texttt{int}, C : \texttt{Map}),\\
& \_[\_/\_](\_,\_(A : \texttt{Id} \mapsto B : \texttt{int}, C : \texttt{Map}), D : \texttt{int}, E : \texttt{Id})\\
& \qquad = \_,\_(A : \texttt{Id} \mapsto B : \texttt{int}, \_[\_/\_](C : \texttt{Map}, D : \texttt{int}, E : \texttt{Id})),\ \_\_(\{\}, A : \texttt{Stmt}) = A : \texttt{Stmt},\\
& \_\_(\{\_\}(A : \texttt{Stmt}), B : \texttt{Stmt}) = \_\_(A : \texttt{Stmt}, B : \texttt{Stmt})\ \}\\
R: & \{-0 \Rightarrow 0,\ A : \texttt{AExp} + \boldsymbol{v}(B : \texttt{Id}) \Rightarrow A : \texttt{AExp} + \boldsymbol{guess}(B : \texttt{Id}),\ -\boldsymbol{true} \Rightarrow \boldsymbol{false},\ -\boldsymbol{false} \Rightarrow \boldsymbol{true},\\
& A : \texttt{AExp} <= \boldsymbol{v}(B : \texttt{Id}) \Rightarrow A : \texttt{AExp} <= \boldsymbol{guess}(B : \texttt{Id}),\ 0 <= A : \texttt{nat} \Rightarrow \boldsymbol{true},\\
& 0 <= -\boldsymbol{s}(A : \texttt{nat}) \Rightarrow \boldsymbol{false},\ \boldsymbol{v}(A : \texttt{Id}) <= B : \texttt{AExp} \Rightarrow \boldsymbol{guess}(A : \texttt{Id}) <= B : \texttt{AExp},\\
& <\_,\_> (A : \texttt{Map}, \_\_(\boldsymbol{if\_else}(\boldsymbol{false}, B : \texttt{Block}, C : \texttt{Block}), D : \texttt{Stmt}))\\
& \qquad \Rightarrow <\_,\_> (A : \texttt{Map}, \_\_(C : \texttt{Block}, D : \texttt{Stmt})),\\
& <\_,\_> (A : \texttt{Map}, \_\_(\boldsymbol{if\_else}(\boldsymbol{true}, B : \texttt{Block}, C : \texttt{Block}), D : \texttt{Stmt}))\\
& \qquad \Rightarrow <\_,\_> (A : \texttt{Map}, \_\_(B : \texttt{Block}, D : \texttt{Stmt})),\\
& <\_,\_> (A : \texttt{Map}, \_\_(\_ = \_;(B : \texttt{Id}, C : \texttt{int}), D : \texttt{Stmt}))\\
& \qquad \Rightarrow <\_,\_> (\_[\_/\_](A : \texttt{Map}, C : \texttt{int}, B : \texttt{Id}), D : \texttt{Stmt}),\\
& -A : \texttt{int} + -B : \texttt{int} \Rightarrow -(A : \texttt{int} + B : \texttt{int}),\ \boldsymbol{false} + A : \texttt{BExp} \Rightarrow \boldsymbol{false}\ \}
\end{array}
$$

Figure 2: IMP Order-Sorted Equations and Rules

The order-sorted algebra in Figures 1 and 2 is strictly sensible, but if we change the operator $+ : \texttt{nat} * \texttt{nat} \to \texttt{AExp}$ to be $+ : \texttt{nat} * \texttt{nat} \to \texttt{nat}$, the algebra becomes sensible but not strictly sensible. The second condition of the strictly sensible definition is not necessary for the translation algorithm in this paper, but it ensures that the translated many-sorted algebra from a *strictly sensible* order-sorted algebra are bisimilar. Without the condition, the translated many-sorted algebra simulates the order-sorted algebra, but there might be some behaviors in the translated many-sorted algebra that cannot be observed in the original order-sorted algebra. The first condition is the key distinction between a sensible order-sorted algebra and a strictly sensible order-sorted algebra. In the translation algorithm described in this paper, we rule out the possibility for users to define overloaded operator pairs like $+ : \texttt{AExp} * \texttt{AExp} \to \texttt{AExp}$ and $+ : \texttt{nat} * \texttt{nat} \to \texttt{nat}$.

The reason that we are willing to accept the *strictly sensible* restriction is that users only need the limited world in defining language specifications from scratch usually. This is a real restriction that will affect some situations, because without the restriction of strictly sensible, we can define two overloaded $+$ operators $+ : \texttt{int} * \texttt{int} \to \texttt{int}$ and $+ : \texttt{nat} * \texttt{nat} \to \texttt{nat}$, where int and nat have a subsort relation. However, there are no such operators in the order-sorted specifications of C [9], PHP [10], JavaScriptv[23], and Java [3] in $\mathbb{K}$. In addition, the operators, such as $+ : \texttt{int} * \texttt{nat} \to \texttt{int}$ and $+ : \texttt{nat} * \texttt{int} \to \texttt{nat}$, are usually defined as different operators with different names by users. Even though we are able to solve them easily by adding more rules and creating more sorts, such as what the algorithm of Meseguer and Skeirik [21] does, we do not want to take that approach because the whole point of the

translation is to have a many-sorted algebra that is concise enough for users to use and read the translated language specifications. Squaring or cubing the size of the rewrite rules is quite undesirable.

On the other hand, it does not mean that we cannot translate order-sorted algebras that are *sensible* but not *strictly sensible* to many-sorted algebras. The paper by Meseguer and Skeirik [21] gives us an naive solution to cover all interesting translation cases. By giving an order-sorted algebra, people can divide it by collecting all operators that do not match with the *strictly sensible* requirements and translate these operators by using the naive algorithm introduced by Meseguer and Skeirik, and then translate the rest of the algebra by using our algorithm, and it will reduce the size of the translated many-sorted algebra disregarding the set of operators that do not match with the *strictly sensible* requirements. We do not research in deep in this path because this is just an engineering task which requires a little careful design. The paper mainly focuses on coming up with a subset of the order-sorted algebras that can be translated to many-sorted ones easily and proving their bi-simulation to the translated many-sorted ones.

Now, we can formally state the properties of the translation function tr to be: given a strictly sensible order-sorted algebra A and a translation function tr applied on A, we have a many-sorted algebra B such that $B = tr(A)$, and for any rule r_A in A, if term t_A in A can transition to t'_A through rule r_A, such that $t_A \longrightarrow_{r_A} t'_A$, then we have $tr(r_A)$ is a rule in B and $tr(t_A) \longrightarrow_{tr(r_A)} tr(t'_A)$. The output of our translation is a many-sorted algebra: B, where the rewrite rules of A and B have the above relation.

3 Translation and Proofs

In this section, a description of the translation function tr and some theories about it are given. For a given order-sorted algebra A with $(S, O, \Phi, \Sigma, E, R)$, we do not need to translate the sort set S because our translation does not change sorts at all. We eliminate the relation O, and we have the functions tr_Σ, tr_E and tr_R for translating operator definitions, equations and rewrite rules.

Translating Operators. Operators are translated in two steps, such that $tr_\Sigma = tr^{\#}_\Sigma \circ tr'_\Sigma$. The first step tr'_Σ is to find a maximal argument-bounding operator f for every operator f'. Since the *strictly sensible* assumptions require any pair of *argument compatible* operators f' and f'' to have the same target sort, we restrict the nature of the argument sorts in these overloaded operators by picking its maximal argument-bounding operator f as a representative for any *argument compatible* operator f'. We then eliminate the operator f' if f is different from f'. Hence, if $\Sigma' = tr'_\Sigma$, then Σ' has fewer operators than Σ and for every overloaded operator set, whose elements are *argument compatible*, Σ' picks exactly one representative operator for it. If the overloaded operators are not *argument compatible*, then we distinguish them by picking different constructors in the translated many-sorted algebra.

For example, in the order-sorted algebra in Figures 1 and 2, there are five different overloaded operators with $+$ constructor, where $+ : \texttt{AExp} * \texttt{AExp} \to \texttt{AExp}$, $+ : \texttt{nat} * \texttt{nat} \to \texttt{AExp}$ and $+ : \texttt{int} * \texttt{int} \to \texttt{AExp}$ are *argument compatible*, while $+ : \texttt{bool} * \texttt{bool} \to \texttt{BExp}$ and $+ : \texttt{BExp} * \texttt{BExp} \to \texttt{BExp}$ are also *argument compatible*. These two groups of $+$ operators are not *argument compatible* cross groups. When translating these operators, we first pick $+ : \texttt{AExp} * \texttt{AExp} \to \texttt{AExp}$ and $+ : \texttt{BExp} * \texttt{BExp} \to \texttt{BExp}$ as the representatives for the first and second group, then we change the name of the first one to $+\boldsymbol{AExp}$ and the second one to $+\boldsymbol{BExp}$ to avoid conflicts in constructor names. Finally, in the translated many-sorted signature, we have two translated operators for the original overloaded operators with $+$ constructor, they are: $+\boldsymbol{AExp} : \texttt{AExp} * \texttt{AExp} \to \texttt{AExp}$ and $+\boldsymbol{BExp} : \texttt{BExp} * \texttt{BExp} \to \texttt{BExp}$.

The translation $tr^{\#}_\Sigma$ translates a given Σ' (a signature translated by tr'_Σ on an order-sorted signature Σ) by adding operators. For each pair defined in set O as (s, s'), which is a subsort relation $s \leq s'$, we create

one more unary operator **Cast_s_to_s'** : $s \to s'$ that does not appear in Σ'. This operator has argument sort s and target sort s'. The result signature $\Sigma^{\#} = tr^{\#}_{\Sigma}(\Sigma')$ contains a set of newly generated unary operators being bijective with the pairs in O.

For example, when translating the order-sorted algebra in Figures 1 and 2, we add the following five unary operators: ***Cast_nat_to_int*** : `nat` $\to$ `int`, ***Cast_int_to_AExp*** : `int` $\to$ `AExp`, ***Cast_Id_to_AExp*** : `Id` $\to$ `AExp`, ***Cast_bool_to_BExp*** : `bool` $\to$ `BExp` and ***Cast_Block_to_Stmt*** : `Block` $\to$ `Stmt`, since there are exactly five tuples in the set O: `nat` $<$ `int`, `int` $<$ `AExp`, `Id` $<$ `AExp`, `bool` $<$ `BExp` and `Block` $<$ `Stmt`.

Similar to the theorems in the signature translation of the paper of Meseguer and Skeirik, we also have the following theorem about the final result $\Sigma^{\#}$. The proof of the theorem about is similar to the one in the paper of Meseguer and Skeirik, and is a direct result of the *strictly sensible* requirement of an order-sorted algebra and our translation of the operators in the algebra.

Theorem 3.1. Let Σ be an order-sorted signature, $\Sigma^{\#}$ is the translated many-sorted algebra of it. All overloaded operators (viewing ***+AExp*** and ***+BExp*** to be overloaded operators) in $\Sigma^{\#}$ have at least one argument position having distinct argument sorts that have no common supersort in the original order-sorted algebra.

Proof. The proof of the theorem about our translation is similar to the one in the paper of Meseguer and Skeirik [21]. Here, we only sketch why it is true. The theorem is a direct result of the *strictly sensible* requirement of an order-sorted algebra and our translation of the operators in the algebra. If an order-sorted algebra is *strictly sensible*, then two overloaded operators are *argument compatible*. After the translation, only one representing operator is selected, so there cannot be two operators being *argument compatible* in the original order-sorted algebra.

□

Translating Terms and Equations. After translating order-sorted signatures to many-sorted ones, we generate terms for the translated many-sorted algebras as the same way to generate sorted ground term algebras in Definition 2.1. Once we have all valid terms for the translated many-sorted algebras, we can define a function tr_E to translate every equation in E to $E^{\#}$ and also add a set of *core equality* equations to $E^{\#}$. After the translation, the terms in the translated many-sorted algebra with the new equation set $E^{\#}$ form a term algebra $T_{(\Sigma^{\#},E^{\#})}$, and $T_{(\Sigma^{\#},E^{\#})}$ also represents the union of term sets for each sort $s \in S$ as $T_{(\Sigma^{\#},E^{\#},s)}$. In the quotient structure $T_{(\Sigma^{\#},E^{\#})}$, the equivalence classes are partitioned by the combination effects of equations $E^{\#}$ and sorts S.

First, the translation tr_E adds equations to the equation set E to generate $E^{\#}$. The new equations are related to the idea of the core of a term. In order to define the core of a term, we first define non-core constructors as the new constructors generated during the operator translation $tr^{\#}_{\Sigma}$. The core constructors are the constructors of the normal operators of Σ. The core part of a term is the top most t of a term $C_1(...(C_n(t))...)$, where $C_1,...,C_n$ are unary non-core constructors. We now show the definition of *core equality*.

Definition 3.1. If there are two lists of unary non-core constructors $C_1,...,C_n$ and $K_1,...,K_m$, such that $t = C_1(...(C_n(x))...)$ and $t' = K_1(...(K_m(x))...)$ are well-formed, i.e., the input sort of C_i is equal to the output sort of C_{i+1} for all $i = 1,...,n-1,...$ and the input sort of K_j is equal to the output sort of K_{j+1} for all $j = 1,...,m-1,...$, as well as C_1 and K_1 has target sort s, C_n and K_m has input sort s', then for each pair of directed paths from s' to s in the graph of $\leq$ in the original order-sorted algebra, i.e., $s' \leq s$, if there are two different paths from s' to s in the graph, we have an equation $C_1(...(C_n(x:s'))...) = K_1(...(K_m(x:s'))...)$. The congruence closure of all these equations is *core equality*.

Theorem 3.2. *Core equality* is an equivalence relation.

Proof. The proof of reflexivity and symmetricity of *core equality* is simple. Here, we only show the transitivity proof. By giving $t =_{core} t'$ and $t' =_{core} t''$, we have $t = C_1(...(C_n(t_c))...)$, $t' = K_1(...(K_m(t_k))...)$ and $t'' = G_1(...(G_p(t_g))...)$ being well formed, $C_1,...,C_n$, $K_1,...,K_m$ and $G_1,...,G_p$ are lists of non-core unary constructors, and $C_1(...(C_n(t_c))...) =_{core} K_1(...(K_m(t_k))...)$ and $K_1(...(K_m(t_k))...) =_{core} G_1(...(G_p(t_g))...)$, then $t_c = t_k = t_g$ under the assumption of $E^{\#}$, and the output sorts of C_1, K_1 and G_1 are the same as well as the input sort of C_n, K_m and G_p are the same; hence, $C_1(...(C_n(t_c))...) =_{core} G_1(...(G_p(t_g))...)$ and *core equality* is transitive. Thus, *core equality* is an equivalence relation.

□

The reason of having *core equality* is that we have new generated terms due to inserting non-core operators to generate terms in a sort by terms in the subsorts of the sort. Semantically, these new terms are translated from the same term in the original order-sorted algebra. If we cannot equate them, it means that after the translation, we have some terms with different meanings that originally belong to the same term. The way to equate these terms is to put them into the same equivalence classes by using equations defined by *core equality*.

For example, when translating the order-sorted algebra in Figures 1 and 2, the generated unary operators, ***Cast_nat_to_int*** : `nat` $\rightarrow$ `int`, ***Cast_int_to_AExp*** : `int` $\rightarrow$ `AExp`, ***Cast_Id_to_AExp*** : `Id` $\rightarrow$ `AExp`, ***Cast_bool_to_BExp*** : `bool` $\rightarrow$ `BExp` and ***Cast_Block_to_Stmt*** : `Block` $\rightarrow$ `Stmt`, are non-core constructors and operators, while the original operators are core ones. To generate the set of *core equality* equations for the order-sorted algebra, we have a practical way to do it; that is to examine the $\leq$ relation. For every two nodes in $\leq$, if there is more than one path from the first node to the second one, we add equations to connect them. In the order-sorted algebra in Figures 1 and 2, if we have one more sort `real` and two more subsort relations, `nat` $<$ `real` and `real` $<$ `AExp`, then two paths can go from `nat` to `AExp` in $\leq$. We add an equation ***Cast_int_to_AExp***(***Cast_nat_to_int***$(A$: `nat`$)) =$ ***Cast_real_to_AExp***(***Cast_nat_to_real***$(A$: `nat`$))$ to $E^{\#}$. By doing a rough counting, we can see that the number of new equations adding into the set $E^{\#}$ is less than $|S|^2$, since the number of elements in set O is bound to $|S|^2$, and we add an core equality equation if and only if there is a diamond relation in the set O: there exist different sorts D, E, $A_1,...,A_n$, $C_1,...,C_m$ such that, $D < A_1 < ... < A_n < E$, $D < C_1 < ... < C_m < E$ and $A_i \neq C_i$ for all $i = 1...min(n,m)$.

After we have *core equality*, we can translate terms of two sides of an equation in E. The two sides belong to $T_\Sigma(X)$. We define a translation function tr_{term} to translate a term in $T_\Sigma(X)$ to a term in $T_{\Sigma^{\#}}(X)$ for every side of an equation in E. For every sub-term, having the form $f(t_1,...,t_i,...,t_m)$, of a term t in $T_\Sigma(X)$, if we compare the constructor f with the set of operators in the translated many-sorted signature $\Sigma^{\#}$, there is a unique operator $f : (s_1,...s_i,...,s_m)$ (the translated many-sorted signature only keeps one operator if there is a set of *argument compatible* operators, and if there are overloaded operators that are not *argument compatible* in the original order-sorted algebra, we can also find the unique f' that is translated from the original f by comparing the sort of s_i with the sort of t_i, because overloaded operators that are not *argument compatible* are translated into two different operators by distinguishing them with more information in the constructors, and they must have at least one argument position with different sorts). If the sort of a position i in the sub-term $f(t_1,...,t_i,...,t_m)$ is defined with sort s according to the operator signature above, but t_i actually has target sort s' and $s' \leq s$, then we find a list of non-core unary constructors $C_1,...,C_n$ to cast the sub-term to sort s as $f(t_1,...,C_1(...(C_n(t_i))...),...,t_m)$ in a well-formed way. If $s' = s$, then we do not need to find the constructors. We know that such a sequence of unary constructors must exist because the set of non-core

unary constructors is bijective with the pairs in O, and $\leq$ is the reflexive and transitive closure of O. If $s' \leq s$, there is a list of pairs in O as $(s', s_1), ..., (s_{n-1}, s)$. Through the list, s' reaches s. For each pair in the list, we have generated a unary constructor. Hence, the sequence of constructors $C_1, ..., C_n$ is exactly the constructors generated for pairs $(s', s_1), ..., (s_{n-1}, s)$. For example, when translating the order-sorted algebra in Figures 1 and 2, the equation $\boldsymbol{s}(A : \texttt{nat}) + B : \texttt{nat} = A : \texttt{nat} + \boldsymbol{s}(B : \texttt{nat})$ is translated to $\boldsymbol{Cast\_int\_to\_AExp}(\boldsymbol{Cast\_nat\_to\_int}(\boldsymbol{s}(A : \texttt{nat}))) + \boldsymbol{Cast\_int\_to\_AExp}\ (\boldsymbol{Cast\_nat\_to\_int}(B : \texttt{nat}))$ $= \boldsymbol{Cast\_int\_to\_AExp}(\boldsymbol{Cast\_nat\_to\_int}(A : \texttt{nat})) + \boldsymbol{Cast\_int\_to\_AExp}(\boldsymbol{Cast\_nat\_to\_int}(\boldsymbol{s}(B : \texttt{nat})))$.

In defining tr_{term}, for each pair of relation (s', s) in $\leq$, we pick a well-formed constructor sequence $C_1, ..., C_n$, such that the target sort of C_1 is s and the input sort of C_n is s'. The sequence defines the way of translating a sub-term t having sort s' to a sort s by constructing $C_1(...(C_n(t))...)$ in tr_{term}. Because of the *core equality* relation, the choice does not affect the construction of the equivalence classes in $T_{(\Sigma^\#, E^\#)}$, and not affect the represented equivalence classes by a term in $T_{\Sigma^\#}(X)$. We have three theorems about the translation function tr_{term} and term in $T_{(\Sigma^\#, E^\#)}$ and $T_{\Sigma^\#}(X)$.

Theorem 3.3. Let Σ be an order-sorted signature, T_Σ be the term algebra of it, E be the equation set of the order-sorted algebra, $T_{\Sigma,E}$ be the terms T_Σ modulo equations E, $T_\Sigma(X)$ be the terms with variables in the order-sorted algebra, $\Sigma^\#$ be the translated many-sorted algebra of signature Σ, $T_{\Sigma^\#}$, $E^\#$, $T_{(\Sigma^\#, E^\#)}$ and $T_{\Sigma^\#}(X)$ are the corresponding translations of items in the order-sorted algebra, and tr_{term} be the translation function of terms.

(1) If a term t has least sort s in Σ, then its translation t' has the target sort s.

(2) For a term t in $T_{\Sigma,E}$, for any two term translation functions tr_{term} and tr'_{term} having difference in picking different sequences of constructors for pairs in $\leq$, if $c \in T_{(\Sigma^\#, E^\#)}$ and $tr_{term}(t) \in c$, then $tr'_{term}(t) \in c$.

(3) For a term $t(X)$ in $T_\Sigma(X)$, for two translation functions tr_{term} and tr'_{term} having difference in picking different sequences of constructors for pairs in $\leq$, we have two terms $tr_{term}(t(X))$ and $tr'_{term}(t(X))$, for any substitution h mapping X to $T_\Sigma^\#$ such that t and t' are the result of replacing each variable x in $tr_{term}(t(X))$ and $tr'_{term}(t(X))$ by $h(x)$, if $c \in T_{(\Sigma^\#, E^\#)}$ and $t \in c$, then $t' \in c$.

Proof. Part (1) is trivial because after we require our operators to be *strictly sensible*, so any term must have a unique least target sort in the original order-sorted algebra and the target sort is also the target sort of the translated term in $T_{\Sigma^\#}$ without converting it to other supersort s' by adding non-core constructors on top of it.

To show (2), if for a term t having sort s', and the translation functions tr_{term} and tr'_{term} cast it into a term in sort s without the need of translating the subterms of t, then the two resulting terms $tr_{term}(t)$ and $tr'_{term}(t')$ are trivially in the same equivalence class based on the definition of *core equality*. The sorts s' and s must be the same because the order-sorted definition in this paper requires sort equivalence in two sides of an equation.

If there is a term t having a subterm $f(t_1, ..., t_n)$, if the position i of the list $t_1, ..., t_n$ has target sort s according to the signature, the term t_i has sort s' and $s' \leq s$, then a given translation function generates well-formed non-core constructor sequences having the form $C_1, ..., C_n$ to translate the term t_i. We refer to the number of the non-core constructors in this sequence as n, which is the same as one of the distances between s' and s in O. We induct on maximal numbers of the non-core constructors in each argument position in a term t. If the maximal number of argument non-core constructors is zero, it means that tr_{term} and tr'_{term} do not translate the direct subterms of $f(t_1, ..., t_n)$, so any translations on $f(t_1, ..., t_n)$ to a target sort s'' generate terms in the same equivalence class. Assuming that when the maximal numbers of non-core constructors are less than k, $tr_{term}(f(tr_{term}(t_1), ..., tr_{term}(t_n)))$ and $tr'_{term}(f(tr'_{term}(t_1), ..., tr'_{term}(t_n)))$ generate terms in the same equivalence class; if the position i in $f(t_1, ..., t_n)$ has sort s, the term t_i has sort

s', $s' \leq s$ and the maximal distance between s' and s is $k+1$, if there is only one path from s to reach s', then $tr_{term}(t_i)$ must be the same as $tr'_{term}(t_i)$ since we generate only one non-core constructor for each pair in O. If there are at least two paths, without losing generality, assuming that tr_{term} has the longest path, tr_{term} picks the well-formed sequence $C_1,...,C_{k+1}$ to translate t_i to a term having sort s and tr'_{term} picks the well-formed sequence $K_1,...,K_m$ to translate t_i to a term having sort s, where $m \leq k+1$. Based on the definition of *core equality*, $C_1(...(C_{k+1}(t_i))...) =_{core} K_1(...(K_m(t_i))...)$, hence, any argument t_i of $f(t_1,...,t_n)$ is translated by tr_{term} and tr'_{term} into terms in the same equivalence class and $f(t_1,...,t_n)$ are also translated by tr_{term} and tr'_{term} into terms in the same equivalence class.

To show (3), the proof basically modifies the proof of part (2) to allow variables in the term and by any substitution on the same variables in two terms t and t' that are generated by tr_{term} and tr'_{term} are in the same equivalence class.

□

Translating Semantic Rules. Translating semantic rules R to $R^{\#}$ is very straight forward and similar to the one in translating equations in E to $E^{\#}$. For each pair $(t(X),t'(X))$ in R, the first step is to apply the term translation on $t(X)$ and $t'(X)$, to be terms in $T_{\Sigma^{\#}}(X)$. Then, we add one rule $(tr_{term}(t(X)),tr_{term}(t'(X)))$ to $R^{\#}$. Since we assume that all order-sorted algebra are sort decreasing, the right hand side of a rule might have a top-most target sort being a subsort of the left hand side of the rule. After they are translated into many-sorted algebras, the two sides of a rule must have the same top-most target sort. We solve this problem by casting the right hand side of a rule to have the top-most sort equal to the left hand side. For example, in translating the rule $-0 \Rightarrow 0$ in the order-sorted algebra in Figures 1 and 2, we make a new rule $-0 \Rightarrow$ ***Cast_nat_to_int***(0) in the translated many-sorted algebra.

For a given order-sorted algebra A as (S,O,Φ,Σ,E,R), our translation produces the many-sorted algebra $(S,tr_\Sigma(\Sigma),tr_E(E),tr_R(R))$. We show that our many-sorted algebra maintains a bi-simulation relation as the original order-sorted algebra. The bi-simulation proof is based on structural inductions on the signature (S,O,Φ,Σ) and $(S,tr_\Sigma(\Sigma))$.

Theorem 3.4 (Bi-simulation between A and $tr(A)$)**.** Let $(S,tr_\Sigma(\Sigma),tr_E(E),tr_R(R))$ be the translated many-sorted algebra of a given order-sorted algebra (S,O,Φ,Σ,E,R), For any r in R and term t in $T_{(\Sigma,E)}$, if $t \longrightarrow_r t'$, then we have $tr_\Sigma(t) \longrightarrow_{tr_R(r)} tr_\Sigma(t')$. For any p in $T_{(\Sigma^{\#},E^{\#})}$, if $p \longrightarrow_{tr_R(r)} p'$, then there are terms t and t' such that $p = tr_\Sigma(t)$, $p' = tr_\Sigma(t')$ and $t \longrightarrow_r t'$.

Proof. Since all *argument compatible* operators are required to be *strong sensible* and maximal argument-bounding, all *argument compatible* operators with the same constructor should be translated into one specific operator in the many-sorted algebra. Hence, the proof of the bi-simulation relation can be divided into three parts. The first part is to show that for every term t in $T_{(\Sigma)}$, after we translate it to $T_{(\Sigma^{\#})}$, it might have many instances $t_1,...,t_n$, but they are equivalent under *core equality*, and vice versa.

We only show one direction of the proof of the first part. We can structurally induct on term t. t can be expressed as $f(t_1,...,t_n)$, after it is translated into a term in $T_{(\Sigma^{\#})}$, we inductively assume that all subterms of $t_1,...,t_n$ are in the same equivalence class under *core equality*. For a term t_i, if there is a list of constructors $C_1,...,C_q$ and $K_1,...,K_m$ such that term t_i can be translated into $C_1(...(C_q(t_i))...)$ and $K_1(...(K_m(t_i))...)$, then $C_1(...(C_q(t_i))...)$ and $K_1(...(K_m(t_i))...)$ are equivalent under *core equality*, because the translation function translates a term t to terms in $T_{(\Sigma^{\#})}$ by adding non-core constructions which are corresponding to the subsort relations in set O, and if it is translated into two different terms, then there are two paths from the target sort of C_1 to the argument sort of C_q (the target sort of C_1 must be the same as the target sort of K_1 and the argument sort of C_q must be the same as the argument sort of K_m), and the *core equality* equations should equate these two terms according to its definition.

The second part is to show that every class c in $T_{(\Sigma,E)}$, when the terms of c are translated to terms in $T_{(\Sigma^\#,E^\#)}$, these terms are still in the same equivalence classes, and for every class c' $T_{(\Sigma^\#,E^\#)}$, and for all terms t' in c', all the terms t that satisfy the relation $tr_{term}(t) = t'$ where $t' \in c'$, are in a unique class c in $T_{(\Sigma,E)}$.

In the proof of the second part, we only show one direction. For a class c in $T_{(\Sigma,E)}$, all its terms are $t_1,...,t_i,...$, for all these terms, when they are translated into terms in $T_{(\Sigma^\#)}$, we know that one term t_i might be translated into different terms $s_1,...,s_n$, but all these terms are equivalent under *core equality*. In addition, when we translate equations, we only translate the two sides of equations, which are terms in $T_{(\Sigma^\#)}(X)$, and the translated two sides have have different representations but they are all equivalent under *core equality*. So for any two terms t_i and t_k in c, when they are translated into $u_1,...,u_n$ and $v_1,...,v_m$ in $T_{(\Sigma^\#)}$; first, $u_1,...,u_n$ and $v_1,...,v_m$ are equivalent under *core equality*, respectively. Second, for any two terms u_a and u_b where a $\in [1,n]$ and $b \in [1,m]$, these two terms can be proved to be equivalent through set $E^\#$, so they are in the same equivalence class in $T_{(\Sigma^\#,E^\#)}$.

The third part is to show that for any r in R and term t in $T_{(\Sigma,E)}$, if $t \longrightarrow_r t'$, then we have $tr_\Sigma(t) \longrightarrow_{tr_R(r)} tr_\Sigma(t')$. For any p in $T_{(\Sigma^\#,E^\#)}$, if $p \longrightarrow_{tr_R(r)} p'$, then there are terms t and t' such that $p = tr_\Sigma(t)$, $p' = tr_\Sigma(t')$ and $t \longrightarrow_r t'$.

We also only show one direction here. For any r in R, it can be expressed as (t_1,t_2), where t_1 and t_2 are in $T_{(\Sigma)}(X)$, there are two situations. First, if the target sorts of t_1 and t_2 are the same, then the argument will be the same as the equation proof in the second part above. If the target sort s' of t_2 is a subsort of s of t_1, we need to show that for any context $C[]$, and for any term $t \longrightarrow_r t'$, and $tr_\Sigma(t) \longrightarrow_{tr_R(r)} tr_\Sigma(t')$, we have $C[t']$ and $tr_\Sigma(C[tr_\Sigma(t')])$ to be both valid (well-formed) terms. The notation $tr_\Sigma(C[])$ means that we have a way to translate the context $C[]$ such that if we put a redex a of $T_{(\Sigma^\#)}$ in the context, the whole expression is valid in $T_{(\Sigma^\#)}$. Recall that we have the condition that $C[t]$ must be a valid term. Hence, the hole in the context $C[]$ must at least be able to hold a term with target sort s, and the target sort of $tr_\Sigma(t')$ is also s, then $tr_\Sigma(C[tr_\Sigma(t')])$ is also a valid term as long as $C[t]$ is a valid term.

□

4 Related Work

The idea of order-sorted algebras was first systematically introduced into the programming language field by Goguen et al. [11]. The main contribution of the work is to introduce subtyping relations for the syntactic constructs so that operators do not only belong to one sort, but also act as constructs in supersort of the defined sort. In addition, it defines a general operational semantic model for order-sorted algebras. Many people tried to define rewriting strategies, unifications and equational rules on top of order-sorted algebras and further extended the operational semantics of order-sorted algebras [1, 5, 12, 14, 20]. Stell [26] tried to introduce a general framework to contain all existing order-sorted algebra semantics in his work. In the paper of Goguen et al. [11], they introduce a way of translating initial free (algebras that have no equations and rules) order-sorted algebras to many-sorted ones. Their way of translation is similar to our work by adding non-core constructors. However, their work is solely on dealing with initial free algebras without mentioning how to translate a general order-sorted algebra to a many-sorted one because the purpose of their translation is to translate their order-sorted logic into a first order logic in a many-sorted world, so that they can show their order-sorted logic is decidable. Obviously, they also do not need to investigate a bi-simulation relation between their order-sorted algebras and the translated many-sorted ones.

Based on the order-sorted algebras, Meseguer et al. [17, 18] developed rewriting logic. The biggest

contribution of rewriting logic is to contain the operational semantics of order-sorted algebras and distinguish equations and rewriting rules so that equations partition the terms into equivalence classes while rewriting rules act like traditional transition rules in structural operational semantics. Based on rewriting logic, Maude [4] implements the syntax and semantics of rewriting logic and provides several useful tools and applications [19, 7, 8]. Other implementation of order-sorted algebras include PROTOS(L) [2] which has an operational semantics based on polymorphic order-sorted resolution. $\mathbb{K}$ [25] is a framework based on order-sorted algebras, which provides language developers a convenient way to write language specifications. A lot of specifications have specified in $\mathbb{K}$, including the semantics of Java [3], Javascript [23], PHP [10], C [9, 13] and LLVM [15].

On the other hand, the study and exploration of many sorted algebra has a long history. Its logic system has been explored by Wang [28]. Many well-known programming languages such as C, Java, LLVM and Python are based on many-sorted algebras. One of the most prominent and mathematical of programming language specifications, Standard ML by Milner, Tofte, Harper, and Macqueen [22] is based on many sorted algebras. The simple type systems of the two famous theorem provers: Isabelle/HOL [24] and Coq [6] are also based on them, which motivates us to provide a translation from order-sorted algebras into many-sorted ones.

As far as we know, the most recent attempt of translating order-sorted algebras into many-sorted ones is given by Meseguer and Skeirik [21]. The purpose of the paper is to prove the decidability of the order-sorted logic defined in their paper by translating it to a many-sorted world, so their translation still focuses initial free order-sorted algebras (algebras that have no equations and rules), and only provide a naive translation to translate order-sorted equations and rules to many-sorted ones. In their translation, by adding possible more sorts, they calculate the least sorts of constructs and put them under corresponding sorts to create the signature of a many-sorted algebra. For any given rule, they add more rules if variables of the rule have subsorts in the original order-sorted algebra. They need to add one more rule for each subsort of a variable in a rule.

For example, in dealing with the order-sorted algebra in Figures 1 and 2, to translate the equation $A:\texttt{AExp}+B:\texttt{AExp}=B:\texttt{AExp}+A:\texttt{AExp}$, they generate three different equations: $A:\texttt{nat}+B:\texttt{nat}=B:\texttt{nat}+A:\texttt{nat}$, $A:\texttt{int}+B:\texttt{int}=B:\texttt{int}+A:\texttt{int}$ and $A:\texttt{AExp}+B:\texttt{AExp}=B:\texttt{AExp}+A:\texttt{AExp}$. The original rule involves only one sort `AExp`. if there is a rule involving `AExp`, `BExp` and `Stmt`, which all have subsorts, then the algorithm generates sixteen different equations in the translated many-sorted algebra. In fact, if there is a rule or an equation involving n variables having different sorts and each of them have m different subsorts, the algorithm generates m^n different rules or equations in the translated many-sorted algebra. On the other hand, our translation does not change their sorts. We view subsort relations as implicit coercions, while our translation makes them into explicit ones by inserting a constructor for each relation and making the relation into a unary operator in the given order-sorted algebra. We insert a new equational rule named *core equality* to introduce new partitions on the equivalence classes of the terms allowed in the algebra. Because of these features, our translation is of similar size of equations and rewrite rules in the translated many-sorted algebra (adding no more than $|S|^2$ new equations) and gives users a simpler final description of the language specifications.

5 Conclusion.

In this paper, we propose an algorithm to translate an order-sorted algebra into a many-sorted one in a restricted domain by requiring the order-sorted algebra to be *strictly sensible*. The key idea of the translation is to add an equivalence relation called *core equality* to the translated many-sorted algebras.

By defining this relation, we reduce the complexity in translating a *strictly sensible* order-sorted algebra to a many-sorted one, and increase the translated many-sorted algebra equations by a number less than $|S|^2$, which is the square of the size of the sort set and is a very small number compared to the number of equations and rules in an algebra. We also keep the number of rewrite rules in the algebra in the same amount. We then prove the order-sorted algebra and the translated many-sorted algebra to be bisimilar (Section 3). We also showed that *core equality* is indeed an equivalence relation and other properties of our translated many-sorted algebras. Along with showing our algorithm and theories, an IMP language is introduced as an example of the algorithm. We believe that our translation facilitates transformations of order-sorted specifications in $\mathbb{K}$ or Maude into many-sorted systems in Isabelle/HOL or Coq, which will empower users to prove theorems about large and popular language specifications.

This work is a important part of building a compilation relation between the $\mathbb{K}$ framework and a functional programming language. We intend to build a transformation to translate specifications defined in $\mathbb{K}$ to specifications defined in Isabelle automatically and correctly. The translated specifications should be human readable and user friendly because the ultimate goal of the project is to use the translated specifications to prove properties about programming languages.

References

[1] María Alpuente, Santiago Escobar, Javier Espert & José Meseguer (2014): *A Modular Order-sorted Equational Generalization Algorithm*. *Inf. Comput.* 235, pp. 98–136, doi:10.1016/j.ic.2014.01.006.

[2] Christoph Beierle & Gregor Meyer (1994): *Run-time type computations in the Warren Abstract machine*. *The Journal of Logic Programming* 18(2), pp. 123 – 148, doi:10.1016/0743-1066(94)90049-3. Available at http://www.sciencedirect.com/science/article/pii/0743106694900493.

[3] Denis Bogdănaş & Grigore Roşu (2015): *K-Java: A Complete Semantics of Java*. In: *Proceedings of the 42nd Symposium on Principles of Programming Languages (POPL'15)*, ACM, pp. 445–456, doi:10.1145/2676726.2676982.

[4] Manuel Clavel, Steven Eker, Patrick Lincoln & José Meseguer (1996): *Principles of Maude*. In J. Meseguer, editor: *Electronic Notes in Theoretical Computer Science*, 4, Elsevier Science Publishers, doi:10.1016/S1571-0661(04)00034-9.

[5] Hubert Comon (1990): *Equational formulas in order-sorted algebras*, pp. 674–688. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/BFb0032066.

[6] Pierre Corbineau (2008): *A Declarative Language for the Coq Proof Assistant*, pp. 69–84. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-540-68103-8_5.

[7] Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln & Carolyn Talcott (2002): *Pathway Logic: Executable Models of Biological Networks*. In: *Fourth International Workshop on Rewriting Logic and Its Applications (WRLA 2002), Pisa, Italy, September 19 — 21, 2002*, *Electronic Notes in Theoretical Computer Science* 71, Elsevier, doi:10.1016/S1571-0661(05)82533-2.

[8] Steven Eker, José Meseguer & Ambarish Sridharanarayanan (2003): *The Maude LTL Model Checker and Its Implementation*. In: *Proceedings of the 10th International Conference on Model Checking Software*, SPIN'03, Springer-Verlag, Berlin, Heidelberg, pp. 230–234, doi:10.1007/3-540-44829-2_16.

[9] Chucky Ellison & Grigore Rosu (2012): *An Executable Formal Semantics of C with Applications*. In: *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'12)*, ACM, pp. 533–544, doi:10.1145/2103656.2103719.

[10] Daniele Filaretti & Sergio Maffeis (2014): *An Executable Formal Semantics of PHP*, pp. 567–592. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-662-44202-9_23.

[11] Joseph A. Goguen, Jean-Pierre Jouannaud & José Meseguer (1985): *Operational Semantics for Order-Sorted Algebra*. In: *Proceedings of the 12th Colloquium on Automata, Languages and Programming*, Springer-Verlag, London, UK, UK, pp. 221–231, doi:10.1007/BFb0015747.

[12] Joseph A. Goguen & José Meseguer (1992): *Order-sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations*. *Theor. Comput. Sci.* 105(2), pp. 217–273, doi:10.1016/0304-3975(92)90302-V.

[13] Chris Hathhorn, Chucky Ellison & Grigore Roşu (2015): *Defining the Undefinedness of C*. In: *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15)*, ACM, pp. 336–345, doi:10.1145/2813885.2737979.

[14] Claude Kirchner, Hélène Kirchner & José Meseguer (1988): *Operational semantics of OBJ-3*, pp. 287–301. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/3-540-19488-6_123.

[15] Liyi Li & Elsa Gunter (2016): *LLVM Semantics*. Available at `https://github.com/kframework/llvm-semantics`.

[16] Narciso Martí-Oliet & José Meseguer (2002): *Rewriting Logic as a Logical and Semantic Framework*, pp. 1–87. Springer Netherlands, Dordrecht, doi:10.1007/978-94-017-0464-9_1.

[17] Narciso Martí-Oliet & José Meseguer (2002): *Rewriting logic: roadmap and bibliography*. *Theoretical Computer Science* 285(2), pp. 121 – 154, doi:10.1016/S0304-3975(01)00357-7. Available at `http://www.sciencedirect.com/science/article/pii/S0304397501003577`. Rewriting Logic and its Applications.

[18] José Meseguer (1999): *Research Directions in Rewriting Logic*. In Ulrich Berger & Helmut Schwichtenberg, editors: *Computational Logic*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 347–398, doi:10.1007/978-3-642-58622-4_10.

[19] José Meseguer (2003): *Software specification and verification in rewriting logic*. *NATO SCIENCE SERIES SUB SERIES III COMPUTER AND SYSTEMS SCIENCES* 191, pp. 133–194.

[20] José Meseguer, Joseph A. Goguen & Gert Smolka (1989): *Order-sorted Unification*. *J. Symb. Comput.* 8(4), pp. 383–413, doi:10.1016/S0747-7171(89)80036-7.

[21] José Meseguer & Stephen Skeirik (2017): *Equational formulas and pattern operations in initial order-sorted algebras*. *Formal Aspects of Computing* 29(3), pp. 423–452, doi:10.1007/s00165-017-0415-5.

[22] Robin Milner, Mads Tofte & David Macqueen (1997): *The Definition of Standard ML*. MIT Press, Cambridge, MA, USA.

[23] Daejun Park, Andrei Ştefănescu & Grigore Roşu (2015): *KJS: A Complete Formal Semantics of JavaScript*. In: *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15)*, ACM, pp. 346–356, doi:10.1145/2737924.2737991.

[24] Lawrence C. Paulson (1990): *Isabelle: The Next 700 Theorem Provers*. In P. Odifreddi, editor: *Logic and Computer Science*, Academic Press, pp. 361–386.

[25] Grigore Roşu & Traian Florin Şerbănuţă (2010): *An Overview of the K Semantic Framework*. *Journal of Logic and Algebraic Programming* 79(6), pp. 397–434, doi:10.1016/j.jlap.2010.03.012.

[26] John G. Stell (2002): *A Framework for Order-Sorted Algebra*. In: *Proceedings of the 9th International Conference on Algebraic Methodology and Software Technology*, AMAST '02, Springer-Verlag, London, UK, UK, pp. 396–410, doi:10.1007/3-540-45719-4_27.

[27] Alexandre Boulgakov A.W. Roscoe Thomas Gibson-Robinson, Philip Armstrong (2014): *FDR3 — A Modern Refinement Checker for CSP*. In Erika Ábrahám & Klaus Havelund, editors: *Tools and Algorithms for the Construction and Analysis of Systems*, *Lecture Notes in Computer Science* 8413, pp. 187–201, doi:10.1007/978-3-642-54862-8_13.

[28] Hao Wang (1952): *Logic of many-sorted theories*. *Journal of Symbolic Logic* 17(2), pp. 105–116, doi:10.2307/2266241.