The Semantics of K

Formal Systems Laboratory
University of Illinois

September 6, 2017

Please feel free to contribute to this report in all ways. You could add new contents, remove redundant ones, refactor and organize the texts, and correct typos, but please follow the FSL rules for editing, though; e.g., <80 characters per line, each sentence on a new line, etc.

1 Matching Logic

Let us recall the basic grammar of matching logic from [**?**]. Assume a matching logic *signature* (S, Σ), and let Var_s be a countable set of *variables* of sort s, where the sets of *sorts* S and of *symbols* Σ are enumerable sets. We partition Σ in sets of symbols $\Sigma_{s_1 \ldots s_n, s}$ of *arity* $s_1 \ldots s_n, s$, where $s_1, \ldots, s_n, s \in S$. Then *patterns* of sort $s \in S$ are generated by the following grammar:

Add references.

$$
\begin{aligned}
\varphi_s ::= {} & x{:}s \quad \text{where } x \in Var \\
& \mid \varphi_s \wedge \varphi_s \\
& \mid \neg \varphi_s \\
& \mid \exists x{:}s'.\varphi_s \quad \text{where } x \in N \text{ and } s' \in S \\
& \mid \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) \quad \text{where } \sigma \in \Sigma \text{ has } n \text{ arguments, and } \ldots
\end{aligned}
$$

Figure 1: The grammar of matching logic.

The grammar above only defines the syntax of (well-formed) patterns of sort s. It says nothing about their semantics. For example, patterns $x{:}s \wedge y{:}s$ and $y{:}s \wedge x{:}s$ are distinct elements in the language of the grammar, in spite of them being semantically/provably equal in matching logic.

For notational convenience, we take the liberty to use mix-fix syntax for operators in Σ, parentheses for grouping, and omit variable sorts when understood. For example, if $Nat \in S$ and $\_+\_, \_*\_ \in \Sigma_{Nat \times Nat, Nat}$ then we may write $(x + y) * z$ instead of $\_*\_(\_+\_(x{:}Nat, y{:}Nat), z{:}Nat)$. More notational convenience and conventions will be introduced along the way as use them.

A matching logic *theory* is a triple (S, Σ, A) where (S, Σ) is a signature and A is a set of patterns called *axioms*. Like in many logics, sets of patterns may be presented as *schemas* making use of meta-variables ranging over patterns, sometimes constrained to subsets of patterns using side conditions. For example:

$\varphi[\varphi_1/x] \wedge (\varphi_1 = \varphi_2) \to \varphi[\varphi_2/x]$	where φ is any pattern and φ_1, φ_2 are any patterns of same sort as x
$(\lambda x.\varphi)\varphi' = \varphi[\varphi'/x]$	where φ, φ' are *syntactic patterns*, that is, ones formed only with variables and symbols This is not true. Pattern φ contains quantifiers.
$\varphi_1 \texttt{+} \varphi_2 = \varphi_1 +_{Nat} \varphi_2$	where φ, φ' are *ground* syntactic patterns of sort *Nat*, that is, patterns built only with symbols `zero` and `succ`
$(\varphi_1 \to \varphi_2) \to (\varphi[\varphi_1/x] \to \varphi[\varphi_2/x])$	where φ is a *positive context in* x, that is, a pattern containing only one occurrence of x with no negation ($\neg$) on the path to x, and where φ_1, φ_2 are any patterns having the same sort

One of the major goals of this paper is to propose a formal language and an implementation, that allows us to write such pattern schemas.

2 A Calculus of Matching Logic

In this section, we define a matching logic theory $K = (S_K, \Sigma_K, A_K)$ as *the calculus of matching logic*, where S_K, Σ_K, and A_K are sets of sorts, symbols, and axioms, respectively.

2.1 Boolean Algebra

The matching logic theory of Boolean algebra is included in K, and the corresponding sort is named *KBool*. Constructors of the sort *KBool* are two functional symbols

$$\mathit{Ktrue}\colon \to \mathit{KBool} \qquad \mathit{Kfalse}\colon \to \mathit{KBool}.$$

Common Boolean operators are defined as functional symbols with their corresponding axioms:

$$\begin{aligned}
&\mathit{KnotBool}\colon \mathit{KBool} \to \mathit{KBool} && \mathit{KnotBool}(\mathit{Ktrue}) = \mathit{Kfalse}\\
&\mathit{KandBool}\colon \mathit{KBool} \times \mathit{KBool} \to \mathit{KBool} && \mathit{KnotBool}(\mathit{Kfalse}) = \mathit{Ktrue}\\
&\mathit{KorBool}\colon \mathit{KBool} \times \mathit{KBool} \to \mathit{KBool} && \mathit{KandBool}(\mathit{Ktrue}, b) = b\\
&\mathit{KimpliesBool}\colon \mathit{KBool} \times \mathit{KBool} \to \mathit{KBool} && \mathit{KandBool}(\mathit{Kfalse}, b) = \mathit{Kfalse}
\end{aligned}$$

The symbols *KorBool* and *KimpliesBool* are defined in terms of the symbols *KnotBool* and *KandBool* in the usual way:

$$KorBool(b_1, b_2) = KnotBool(KandBool(KnotBool(b_1), KnotBool(b_2)))$$
$$KimpliesBool(b_1, b_2) = KorBool(KnotBool(b_1), b_2).$$

Notation 1. If b is a term pattern of sort *KBool*, then we will write just b instead of $b = Ktrue$ so that we can use Boolean expressions in any sort context.

Notation 2. To write Boolean expressions compactly, we adopt the following abbreviations if there is no confusion

$$\neg b \equiv KnotBool(b) \qquad b_1 \wedge b_2 \equiv KandBool(b_1, b_2)$$
$$b_1 \vee b_2 \equiv KorBool(b_1, b_2) \qquad b_1 \to b_2 \equiv KimpliesBool(b_1, b_2).$$

2.2 Strings

The sort *KString* is the sort for strings. It has the following $26+26+10+2 = 64$ functional constructors:

$$\begin{array}{lll}
\text{“a”}\colon \to KString & \text{“b”}\colon \to KString & \text{“c”}\colon \to KString \\
\dots & \dots & \dots \\
\text{“x”}\colon \to KString & \text{“y”}\colon \to KString & \text{“z”}\colon \to KString \\
\text{“A”}\colon \to KString & \text{“B”}\colon \to KString & \text{“C”}\colon \to KString \\
\dots & \dots & \dots \\
\text{“X”}\colon \to KString & \text{“Y”}\colon \to KString & \text{“Z”}\colon \to KString \\
\text{“0”}\colon \to KString & \dots & \text{“9”}\colon \to KString \\
\epsilon\colon \to KString & \multicolumn{2}{l}{Kconcat\colon KString \times KString \to KString.}
\end{array}$$

The associativity and identity of *Kconcat* are defined by the following axioms:

$$Kconcat(s_1, (Kconcat(s_2, s_3))) = Kconcat(Kconcat(s_1, s_2), s_3)$$
$$Kconcat(s, \epsilon) = s \quad Kconcat(\epsilon, s) = s.$$

Notation 3. As a convention, strings are often wrapped with quotation marks and the constructor *Kconcat* is often omitted. Therefore, instead of writing

$$Kconcat(\text{“a”}, Kconcat(\text{“b”}, \text{“c”})),$$

we simply write “abc”, thanks to the associativity of *Kconcat*.

2.3 Matching Logic Sorts and Symbols

The sort *KSort* is the sort for matching logic sorts. The only constructor of the sort *KSort* is the functional symbol:

$$Ksort\colon KString \to KSort.$$

The sort *KSymbol* is the sort for matching logic symbols. The only constructor of the sort *KSymbol* is the functional symbol:

$$Ksymbol\colon KString \to KSymbol.$$

2.4 Finite Lists

Whenever we introduce a sort, say X, to S_K, we feel free to use *XList* as the sort of finite lists whose elements are of sort X. If we do that, it means three things. Firstly, the sort *XList* is in S_K. Secondly, the following functional symbols are in Σ_K:

$$\begin{aligned} &\mathit{nilXList}\colon \to \mathit{XList} && \mathit{inXList}\colon X \times \mathit{XList} \to \mathit{KBool} \\ &\mathit{appendXList}\colon \mathit{XList} \times \mathit{XList} \to \mathit{XList} && \mathit{XListAsX}\colon X \to \mathit{XList} \\ &\mathit{deleteXList}\colon X \times \mathit{XList} \to \mathit{XList} \end{aligned}$$

where *nilXList*, *XListAsX*, and *appendXList* are constructors of sort *XList*. Thirdly, the following axioms are in A_K:

$$\begin{aligned} &\mathit{appendXList}(l_1, \mathit{appendXList}(l_2, l_3)) = \mathit{appendXList}(\mathit{appendXList}(l_1, l_2), l_3) \\ &\mathit{appendXList}(l, \mathit{nilXList}) = l \quad \mathit{appendXList}(\mathit{nilXList}, l) = l \\ &\mathit{inXList}(x, \mathit{nilXList}) = \mathit{Kfalse} \quad \mathit{inXList}(x, \mathit{XListAsX}(x)) = \mathit{Ktrue} \\ &x \neq y \to \mathit{inXList}(x, \mathit{XListAsX}(y)) = \mathit{Kfalse} \\ &\mathit{inXList}(x, \mathit{appendXList}(l_1, l_2)) = \mathit{inXList}(x, l_1) \vee \mathit{inXList}(x, l_2) \\ &\mathit{deleteXList}(x, \mathit{nilXList}) = \mathit{nilXList} \quad \mathit{deleteXList}(x, \mathit{XListAsX}(x)) = \mathit{nilXList} \\ &x \neq y \to \mathit{deleteXList}(x, \mathit{XListAsX}(y)) = \mathit{XListAsX}(y) \\ &\mathit{deleteXList}(x, \mathit{appendXList}(l_1, l_2)) = \mathit{appendXList}(\mathit{deleteXList}(x, l_1), \mathit{deleteXList}(x, l_2)). \end{aligned}$$

Notation 4. To write lists expressions compactly, we adopt the following shorthands for *inXList* and *deleteXList* when there is no confusion:

$$\begin{aligned} &x \in l \equiv (\mathit{inXList}(x, l) = \mathit{Ktrue}) \qquad x \notin l \equiv (\mathit{inXList}(x, l) = \mathit{Kfalse}) \\ &\mathit{deleteXList}(x, l) \equiv \mathit{delete}(x, l). \end{aligned}$$

Note that one should not confuse the shorthand for *inXList* with the matching logic membership connective, which we all write as the belongs-to symbol "$\in$". If the two patterns before and after the belongs-to symbol are of the same sort, it is the membership connective. If the left of them is of sort X and the right is of sort *XList*, the belongs-to symbol is the shorthand of *inXList*.

Constructors *XListAsX* and *appendXList* are often omitted, too:

$$x_1, \ldots, x_n \equiv \mathit{appendXList}(x_1, \mathit{appendXList}(\ldots, \mathit{appendXList}(x_{n-1}, x_n) \ldots)).$$

2.5 Matching Logic Patterns

The sort *KPattern* is the sort for matching logic patterns. It has the following functional symbols:

$$\begin{aligned}
&\mathit{Kvariable}\colon \mathit{KString} \times \mathit{KSort} \to \mathit{KPattern}\\
&\mathit{Kand}, \mathit{Kor}, \mathit{Kimplies}, \mathit{Kiff}\colon \mathit{KPattern} \times \mathit{KPattern} \times \mathit{KSort} \to \mathit{KPattern}\\
&\mathit{Knot}\colon \mathit{KPattern} \times \mathit{KSort} \to \mathit{KPattern}\\
&\mathit{Kapplication}\colon \mathit{KSymbol} \times \mathit{KPatternList} \to \mathit{KPattern}\\
&\mathit{Kexists}, \mathit{Kforall}\colon \mathit{KString} \times \mathit{KSort} \times \mathit{KPattern} \times \mathit{KSort} \to \mathit{KPattern}\\
&\mathit{Kequals}, \mathit{Kin}, \mathit{Kcontains}\colon \mathit{KPattern} \times \mathit{KPattern} \times \mathit{KSort} \times \mathit{KSort} \to \mathit{KPattern}\\
&\mathit{Kfloor}, \mathit{Kceil}\colon \mathit{KPattern} \times \mathit{KSort} \times \mathit{KSort} \to \mathit{KPattern}\\
&\mathit{Ktop}, \mathit{Kbottom}\colon \mathit{KSort} \to \mathit{KPattern},
\end{aligned}$$

where *Kvariable*, *Kand*, *Kor*, *Kapplication*, and *Kexists* are constructors of sort *KPattern*.

Notation 5. As a convention, we use b for *KBool* variables, x, y, z for *KString* variables, s for *KSort* variables, σ for *KSymbol* variables, and φ, ψ for *KPattern* variables.

Notation 6. Patterns of sort *KPattern* are easily getting huge quickly. The following abbreviations are adopted to write compact *KPattern* patterns.

$x{:}s \equiv \mathit{Kvariable}(x, s)$. Sometimes it abbreviates to just x.

$$\begin{array}{ll}
\varphi \wedge_s \psi \equiv \mathit{Kand}(\varphi, \psi, s) & \varphi \vee_s \psi \equiv \mathit{Kor}(\varphi, \psi, s)\\
\varphi \to_s \psi \equiv \mathit{Kimplies}(\varphi, \psi, s) & \varphi \leftrightarrow_s \psi \equiv \mathit{Kiff}(\varphi, \psi, s)\\
\neg_s \varphi \equiv \mathit{Knot}(\varphi, x, s) & \exists_{s_1}^{s_2} x.\varphi \equiv \mathit{Kexists}(x, s, \varphi, s')\\
\forall_{s_1}^{s_2} x.\varphi \equiv \mathit{Kforall}(x, s, \varphi, s') & \varphi =_{s_1}^{s_2} \psi \equiv \mathit{Kequals}(\varphi, \psi, s_1, s_2)\\
\varphi \supseteq_{s_1}^{s_2} \psi \equiv \mathit{Kcontains}(\varphi, \psi, s_1, s_2) & \\
\sigma(\varphi_1, \ldots, \varphi_n) \equiv \mathit{Kapplication}(\sigma, (\varphi_1, \ldots, \varphi_n)) & \\
\lfloor \varphi \rfloor_{s_1}^{s_2} \equiv \mathit{Kfloor}(\varphi, s_1, s_2) & \lceil \varphi \rceil_{s_1}^{s_2} \equiv \mathit{Kceil}(\varphi, s_1, s_2)\\
\top_s \equiv \mathit{Ktop}(s) & \bot_s \equiv \mathit{Kbottom}(s)
\end{array}$$

Apart from the five constructors of sort *KPattern*, all the other symbols are derived connectives, which are defined by the following axioms:

$$\begin{array}{lr}
\varphi \vee_s \psi = \neg_s(\neg_s \varphi \wedge_s \neg_s \psi) & \varphi \to_s \psi = \neg_s \varphi \vee_s \psi\\
\varphi \leftrightarrow_s \psi = (\varphi \to_s \psi) \wedge_s (\psi \to_s \varphi) & \forall_{s_1}^{s_2} x.\varphi = \neg_{s_2} \exists_{s_1}^{s_2} x.(\neg_{s_2} \varphi)\\
\varphi =_{s_1}^{s_2} \psi = \lfloor \varphi \leftrightarrow_{s_1} \psi \rfloor_{s_1}^{s_2} & \varphi \supseteq_{s_1}^{s_2} \psi = \lfloor \psi \to_{s_1} \varphi \rfloor_{s_1}^{s_2}\\
x \in_{s_1}^{s_2} \varphi = x \subseteq_{s_1}^{s_2} \varphi & \\
\top_s \equiv \exists_s^s x.x & \bot_s \equiv \neg_s \top_s\\
\lfloor \varphi \rfloor_{s_1}^{s_2} \equiv \neg_{s_2} \lceil \neg_{s_1} \varphi \rceil_{s_1}^{s_2} & \lceil \varphi \rceil_{s_1}^{s_2} \equiv \mathit{Ksymbol}(\text{“ceil”})(\varphi)
\end{array}$$

not sure about the last one ... the definedness symbol...

The functional symbol

$$\mathit{KgetFvs}\colon \mathit{KPattern} \to \mathit{KPatternList}$$

collects all free variables in a pattern. It has the following axioms:

$$\begin{aligned}
&\mathit{KgetFvs}(x{:}s) = x{:}s \quad \mathit{KgetFvs}(\neg_s \varphi) = \mathit{KgetFvs}(\varphi)\\
&\mathit{KgetFvs}(\varphi \wedge_s \psi) = \mathit{KgetFvs}(\varphi), \mathit{KgetFvs}(\psi)\\
&\mathit{KgetFvs}(\sigma(\varphi_1, \ldots, \varphi_n)) = \mathit{KgetFvs}(\varphi_1), \ldots, \mathit{KgetFvs}(\varphi_n)\\
&\mathit{KgetFvs}(\exists_{s_1}^{s_2} x.\varphi) = \mathit{delete}(\mathit{Kvariable}(x, s_1), \mathit{KgetFvs}(\varphi)).
\end{aligned}$$

The functional symbol

$$\mathit{KfreshName}\colon \mathit{KPatternList} \to \mathit{KString}$$

generates a variable name that does not occur free in the argument patterns. It has the following axiom:

$$\mathit{Kvariable}(\mathit{KfreshName}(\varphi_1, \ldots, \varphi_n), s) \notin \mathit{KgetFvs}(\varphi_1), \ldots, \mathit{KgetFvs}(\varphi_n).$$

The functional symbol

$$\mathit{Ksubstitute}\colon \mathit{KPattern} \times \mathit{KPattern} \times \mathit{KPattern} \to \mathit{KPattern}$$

takes a target pattern φ, a “find”-pattern x, and a “replace”-pattern ψ, and returns $\varphi[\psi/x]$.

Notation 7. We abbreviate $\mathit{Ksubstitute}(\varphi, \psi, x)$ as $\varphi[\psi/x]$.

The function *Ksubstitute* has the following axioms:

$$\begin{array}{ll}
x[\psi/x] = \psi & x \neq y \to y[\psi/x] = y\\
(\varphi_1 \wedge_s \varphi_2)[\psi/x] = \varphi_1[\psi/x] \wedge_s \varphi_2[\psi/x] & (\neg_s \varphi)[\psi/x] = \neg_s \varphi[\psi/x]\\
\sigma(\varphi_1, \ldots, \varphi_n)[\psi/x] = \sigma(\varphi_1[\psi/x], \ldots, \varphi_n[\psi/x]) & (\exists_{s_1}^{s_2} x.\varphi)[\psi/x] = \exists_{s_1}^{s_2} x.\varphi\\
\multicolumn{2}{l}{(\exists_{s_1}^{s_2} x.\varphi)[\psi/y] = \exists z.z = \mathit{KfreshName}(\varphi, \psi, x) \wedge \exists_{s_1}^{s_2} z.(\varphi[z/x][\psi/y]).}
\end{array}$$

2.6 Matching Logic Signatures

The sort *KSignature* is the sort for matching logic signatures, and it has just one constructor symbol:

$$\mathit{Ksignature}\colon \mathit{KSortList} \times \mathit{KSymbolList} \to \mathit{KSignature}.$$

Notation 8. As a convention, we use Σ, Ψ for *KSignature* variables. We use S for *KSortList* variables and Σ for *KSymbolList* variables if they appear in *Ksignature*.

Notation 9. We abbreviate $\mathit{Ksignature}(S, \Sigma)$ as (S, Σ).

The functional symbol

$$\mathit{KgetSort}\colon \mathit{KPattern} \times \mathit{KSignature} \rightharpoonup \mathit{KSort}$$

returns the sort of the argument pattern in the argument signature if the argument pattern is well-formed. Otherwise, it returns $\bot_{KSort}$. The following axioms define *KgetSort*:

$$\begin{aligned}
&\mathit{KgetSort}(x{:}s, (S, \Sigma)) = (s \in S) \wedge s \\
&\mathit{KgetSort}(\varphi \wedge_s \psi, \Sigma) = (\mathit{KgetSort}(\varphi, \Sigma) = s) \wedge (\mathit{KgetSort}(\psi, \Sigma) = s) \wedge s \\
&\mathit{KgetSort}(\neg_s \varphi, \Sigma) = (\mathit{KgetSort}(\varphi, \Sigma) = s) \wedge s \\
&\mathit{KgetSort}(\sigma(\varphi_1, \ldots, \varphi_n), (S, \Sigma)) = (\ldots) \wedge s \\
&\mathit{KgetSort}(\exists_{s_1}^{s_2} x.\varphi, \Sigma) = (s_1 \in S) \wedge (s_2 \in S) \wedge (\mathit{KgetSort}(\varphi, \Sigma) = s_2) \wedge s_2
\end{aligned}$$

The functional symbol

$$\mathit{KwellFormed}\colon \mathit{KPattern} \times \mathit{KSignature} \to \mathit{KBool}$$

returns *Ktrue* if the argument pattern is well-formed in the argument signature. It has the following axioms:

$$\begin{aligned}
&\mathit{KwellFormed}(x{:}s, (S, \Sigma)) = s \in S \\
&\mathit{KwellFormed}(\varphi \wedge_s \psi, \Sigma) = \mathit{KgetSort}(\varphi, \Sigma) = s \wedge \mathit{KgetSort}(\psi, \Sigma) = s \\
&\mathit{KwellFormed}(\neg_s \varphi, \Sigma) = \mathit{KgetSort}(\varphi, \Sigma) = s \\
&\mathit{KwellFormed}(\sigma(\varphi_1, \ldots, \varphi_n), \Sigma) = \ldots \\
&\mathit{KwellFormed}(\exists_{s_1}^{s_2} x.\varphi, (S, \Sigma)) = s_1 \in S \wedge \mathit{KgetSort}(\varphi, (S, \Sigma)) = s_2
\end{aligned}$$

Notation 10. The well-formedness is an important premise in axioms about the sort *KPattern*. To keep our notation simple, we often omit the well-formedness premises when defining axioms in Σ_K.

2.7 Matching Logic Theories

The sort *KTheory* is the sort of matching logic theories. The only constructor symbol is

$$\mathit{Ktheory}\colon \mathit{KSignature} \times \mathit{KPatternList} \to \mathit{KTheory}.$$

Notation 11. As a convention, we use A, F for *KPatternList* variables if they appear in *Ktheory*. We use T for *Ktheory* variables.

Notation 12. We abbreviate $\mathit{Ktheory}(\Sigma, A)$ as (Σ, A), and abbreviate $\mathit{Ktheory}(\mathit{Ksignature}(S, \Sigma), A)$ as (S, Σ, A).

2.8 Matching Logic Proof System

A sound and complete proof system has been introduced in [**?**].

The functional symbol

$$\mathit{Kdeduce}\colon \mathit{KTheory} \times \mathit{KPattern} \to \mathit{KBool}$$

returns *Ktrue* if the argument pattern is deducible in the argument theory. The functional symbol *Kdeduce* has axioms in correspondence to the inference rules in the proof system. In the following, we are going to list all the inference rules in the matching logic proof system followed by their correspondent axioms of *Kdeduce*, in which the well-formedness premises are omitted (Notation 10).

Rule (Axiom). $F \vdash \varphi$ if $\varphi \in F$.

$$\varphi \in F \to \mathit{Kdeduce}((\Sigma, F), \varphi).$$

Rule (K1). $\vdash \varphi \to (\psi \to \varphi)$.

$$\mathit{Kdeduce}(T, \varphi \to_s (\psi \to_s \varphi)).$$

Rule (K2). $\vdash (\varphi_1 \to (\varphi_2 \to \varphi_3)) \to ((\varphi_1 \to \varphi_2) \to (\varphi_1 \to \varphi_3))$.

$$\mathit{Kdeduce}(T, (\varphi_1 \to_s (\varphi_2 \to_s \varphi_3)) \to_s ((\varphi_1 \to_s \varphi_2) \to_s (\varphi_1 \to_s \varphi_3))).$$

Rule (K3). $\vdash (\neg\psi \to \neg\varphi) \to (\varphi \to \psi)$.

$$\mathit{Kdeduce}(T, (\neg_s\psi \to_s \neg_s\varphi) \to_s (\varphi \to_s \psi)).$$

Rule (K4). $\vdash \forall x.\varphi \to \varphi[y/x]$.

$$\mathit{Kdeduce}(T, \forall_{s_1}^{s_2} x.\varphi \to_{s_2} \varphi[y/x]).$$

Rule (K5). $\vdash \forall x.(\varphi \to \psi) \to (\varphi \to \forall x.\psi)$ if x does not occur free in φ.

$$x \notin \mathit{KgetFvs}(\varphi) \to \mathit{Kdeduce}(T, \forall_{s_1}^{s_2} x.(\varphi \to_{s_2} \psi) \to_{s_2} (\varphi \to_{s_2} \forall_{s_1}^{s_2} x.\psi)).$$

Rule (K6). $\vdash \varphi_1 = \varphi_2 \to (\psi[\varphi_1/x] \to \psi[\varphi_2/x])$.

$$\mathit{Kdeduce}(T, \varphi_1 =_{s_1}^{s_2} \varphi_2 \to_{s_2} (\psi[\varphi_1/x] \to_{s_2} \psi[\varphi_2/x])).$$

Rule (Df). $\vdash \lceil x \rceil$. not sure...

$$\mathit{Kdeduce}(T, \mathit{Ksymbol}(\text{“}\mathit{ceil}\text{”})(x)).$$

Rule (M1). $\vdash x \in y = (x = y)$.

$$\mathit{Kdeduce}(T, x \in_{s_1}^{s_2} y =_{s_2}^{s_3} (x =_{s_1}^{s_2} y)).$$

Rule (M2). $\vdash x \in (\varphi \wedge \psi) = (x \in \varphi) \wedge (x \in \psi)$.

$$\mathit{Kdeduce}(T, x \in_{s_1}^{s_2} (\varphi \wedge_{s_1} \psi) =_{s_2}^{s_3} (x \in_{s_1}^{s_2} \varphi) \wedge_{s_2} (x \in_{s_1}^{s_2} \psi)).$$

Rule (M3). $\vdash x \in \neg\varphi = \neg(x \in \varphi)$.

$$\mathit{Kdeduce}(T, x \in_{s_1}^{s_2} \neg_{s_1}\varphi =_{s_2}^{s_3} \neg_{s_2}(x \in_{s_1}^{s_2} \varphi)).$$

Rule (M4). $\vdash x \in \forall y.\varphi = \forall y.x \in \varphi$ if x is distinct from y.

$$x \neq y \to \mathit{Kdeduce}(T, x \in_{s_2}^{s_3} \forall_{s_1}^{s_2} y.\varphi =_{s_3}^{s_4} \forall_{s_1}^{s_3} y.x \in_{s_2}^{s_3} \varphi).$$

Rule (M5). $\vdash x \in \sigma(\ldots \varphi_i \ldots) = \exists y. y \in \varphi_i \wedge x \in \sigma(\ldots y \ldots)$ where y is distinct from x and it does not occur free in $\sigma(\ldots \varphi_i \ldots)$.

$$
\begin{aligned}
&x \neq y \wedge y \notin \mathit{KgetFvs}(\mathit{Kapplication}(\sigma, (l{:}\mathit{KPatternList}, \varphi_i, r{:}\mathit{KPatternList})))\\
\to{}&\mathit{Kdeduce}(T, x \in_{s_1}^{s_2} \mathit{Kapplication}(\sigma, (l{:}\mathit{KPatternList}, \varphi_i, r{:}\mathit{KPatternList}))\\
&\quad =_{s_2}^{s_4} \exists_{s_3}^{s_4} y. y \in_{s_3}^{s_2} \varphi_i \wedge x \in_{s_1}^{s_2} \mathit{Kapplication}(\sigma, (l{:}\mathit{KPatternList}, y, r{:}\mathit{KPatternList}))).
\end{aligned}
$$

Rule (Modus Ponens). If $\vdash \varphi$ and $\vdash \varphi \to \psi$, then $\vdash \psi$.

$$\mathit{Kdeduce}(T, \varphi) \wedge \mathit{Kdeduce}(T, \varphi \to_s \psi) \to \mathit{Kdeduce}(T, \psi).$$

Rule (Universal Generalization). If $\vdash \varphi$, then $\vdash \forall x. \varphi$.

$$\mathit{Kdeduce}(T, \varphi) \to \mathit{Kdeduce}(T, \forall_{s_1}^{s_2} x. \varphi).$$

Rule (Membership Introduction). If $\vdash \varphi$ and x does not occur free in φ, then $\vdash x \in \varphi$.

$$\mathit{Kdeduce}(T, \varphi) \wedge x \notin \mathit{KgetFvs}(\varphi) \to \mathit{Kdeduce}(T, x \in_{s_1}^{s_2} \varphi).$$

Rule (Membership Elimination). If $\vdash x \in \varphi$ and x does not occur free in φ, then $\vdash \varphi$.

$$\mathit{Kdeduce}(T, x \in_{s_1}^{s_2} \varphi) \wedge x \notin \mathit{KgetFvs}(\varphi) \to \mathit{Kdeduce}(T, \varphi).$$

Theorem 13 (Faithfulness of K)**.** $T \vdash \varphi$ *iff* $K \vdash \mathit{Kdeduce}(T, \varphi)$.

Proof. TBC. □

3 The Kore Syntax

The Kore syntax that we propose here is the *minimal* one, in the sense that it supports only the five basic matching logic constructors (variables, symbol applications, conjunction, negation, and existential quantifications), plus parametricity, which allows to define matching logic theories with infinite sorts, symbols, and axioms.

$\langle\mathit{definition}\rangle$ ::= $\langle\mathit{declaration}\rangle^*$

$\langle\mathit{declaration}\rangle$ ::= `sort` $\langle\mathit{sort}\rangle$
  | `hooked-sort` $\langle\mathit{sort}\rangle$
  | `symbol` $\langle\mathit{symbol}\rangle$ `(` $\langle\mathit{sort}\rangle^*$ `)` `:` $\langle\mathit{sort}\rangle$
  | `hooked-symbol` $\langle\mathit{symbol}\rangle$ `(` $\langle\mathit{sort}\rangle^*$ `)` `:` $\langle\mathit{sort}\rangle$
  | `axiom` $\langle\mathit{pattern}\rangle$

$\langle\mathit{sort}\rangle$ ::= $\langle\mathit{atomic\text{-}sort}\rangle$ | $\langle\mathit{parametric\text{-}sort}\rangle$

⟨*parametric-sort*⟩ ::= `{` ⟨*sort-variable*⟩ `}`
  | ⟨*sort-constructor*⟩ `{` ⟨*sort-variable*⟩$^+$ `}`

⟨*pattern*⟩ ::= ⟨*pattern-variable*⟩
  | ⟨*symbol-id*⟩ `(` ⟨*pattern-list*⟩ `)`
  | `\and (` ⟨*pattern*⟩ `,` ⟨*pattern*⟩ `)`
  | `\not (` ⟨*pattern*⟩ `)`
  | `\exists (` ⟨*pattern-variable*⟩ `,` ⟨*pattern*⟩ `)`

Sorts are declared using the `sort` keyword, symbols are declared using the `symbol` keyword, and axioms are defined using the `axiom` keyword.

```
/* Example 1: One-Element */
sort Element1
symbol e() : Element1
axiom  e()

/* Example 2: Two-Element */
sort Element2
symbol e1() : Element2
symbol e2() : Element2
axiom  \not(\and(\not(e1()), \not(e2())))
axiom  \not(\and(e1(), e2()))
```

Symbols can be parametric on sort variables, which are wrapped by curly brackets. Sort variables can be instantiated by any sort.

```
/* Example 3: Identity-Function */
symbol id({S}) : {S}
axiom  \not(\and(X:{S},     \not(id(X:{S}))))
axiom  \not(\and(id(X:{S}), \not(X:{S})))
```

Given the context where two sorts `Element1` and `Element2` are declared, the above is just a shorthand of

```
/* Example 3': Identity-Function */
symbol id(Element1) : Element1
symbol id(Element2) : Element2
axiom  \not(\and(X:Element1,     \not(id(X:Element1))))
axiom  \not(\and(id(X:Element1), \not(X:Element1)))
axiom  \not(\and(X:Element2,     \not(id(X:Element2))))
axiom  \not(\and(id(X:Element1), \not(X:Element1)))
```

If a sort variable appears multiple times in a statement, only one of the appearances needs curly brackets. Therefore, an alternative way to write Example 3 is

```
/* Example 3'': Identity-Function */
symbol id({S}) : S
axiom  \not(\and(X:{S},     \not(id(X:S))))
axiom  \not(\and(id(X:{S}), \not(X:S)))
```

If a variable is decorated with a sort variable, as in `X:{S}`, the sort decoration can be omitted. Therefore, an alternative way to write Example 3 is

```
/* Example 3''': Identity-Function */
symbol id({S}) : S
axiom  \not(\and(X,     \not(id(X))))
axiom  \not(\and(id(X), \not(X)))
```

Symbols can be parametric on more than one sort variables.

```
/* Example 4: Definedness */
symbol ceil({S}) : {S'}
axiom  ceil(X)
```

Use sort constructors and sort variables to declare parametric sorts.

```
/* Example 5: Lists */
sort List{S}
symbol nil() : List{S}
symbol cons(S, List{S}) : List{S}
symbol length(List{S}) : Nat
```

Parametric sorts can take multiple sort variables.

```
/* Example 6: Pairs */
sort Pair{S, S'}
symbol pairOf(S, S') : Pair{S, S'}
symbol fst(Pair{S, S'}) : S
symbol snd(Pair{S, S'}) : S'
```

Example (Boolean Algebra).

```
sort Bool
symbol true() : Bool
symbol false() : Bool
axiom  \or(true(), false())
axiom  \not(\and(true(), false()))
symbol andBool(Bool, Bool) : Bool
axiom  \equals(andBool(true(), B), B)
axiom  \equals(andBool(false(), B), false())
```

4 Ignore Me

4.1 Syntax and Semantics of Kore

```
// Namespaces for sorts, variables, metavariables,
// symbols, and Kore modules.
Sort           = String
VariableId     = String
MetaVariableId = String
Symbol         = String
```

```
ModuleId       = String

Variable       = VariableId:Sort
MetaVariable   = MetaVariableId::Sort

Pattern        = Variable | MetaVariable
               | \and(Pattern, Pattern)
               | \not(Pattern)
               | \exists(Variable, Pattern)
               | Symbol(PatternList)

Sentence       = import ModuleId
               | syntax Sort
               | syntax Sort ::= Symbol(SortList)
               | axiom Pattern
Sentences      = Sentence | Sentences Sentences

Module         = module ModuleId
                   Sentences
                 endmodule
```

In Kore syntax, the backslash "\" is reserved for matching logic connectives and the sharp "#" is reserved for the meta-level, i.e., the K sorts and symbols. Therefore, the sorts *KBool*, *KString*, *KSymbol*, *KSort*, and *KPattern* in the calculus K are denoted as `#Bool`, `#String`, `#Symbol`, `#Sort`, and `#Pattern` in Kore respectively. Symbols in K are denoted in the similar way, too. For example, the constructor symbol $\mathit{Kvariable}\colon \mathit{KString} \times \mathit{KSort} \to \mathit{KPattern}$ is denoted as `#variable` in Kore.

A Kore module definition begins with the keyword `module` followed by the name of the module-being-defined, and ends with the keyword `endmodule`. The body of the definition consists of some *sentences*, whose meaning are introduced in the following.

The keyword `import` takes an argument as the name of the module-being-imported, and looks for that module in previous definitions. If the module is found, the body of that module is copied to the current module. Otherwise, nothing happens. The keyword `syntax` leads a *syntax declaration*, which can be either a *sort declaration* or a *symbol declaration*. Sorts declared by sort declarations are called *object-sorts*, in comparison to the five *meta-sorts*, `#Bool`, `#String`, `#Symbol`, `#Sort`, and `#Pattern`, in K. Symbols whose argument sorts and return sort are all object-sorts (meta-sorts) are called *object-symbols* (*meta-sorts*).

Patterns are written in prefix forms. A pattern is called an *object-pattern* (*meta-pattern*) if all sorts and symbols in it are object (meta) ones. Meta-symbols will be added to the calculus K, while object-sorts and object-symbols will not. They only serve for the purpose to parse an object pattern.

The keyword `axiom` takes a pattern and adds an axiom to the calculus K. If the pattern is a meta-pattern, it adds the pattern itself as an axiom. If the pattern φ is an object-pattern, it adds $[\![\varphi]\!]$ as an axiom to the calculus K.

Recall that we have defined the semantics bracket as

$$[\![\varphi]\!] \equiv (\mathit{deducible}\,(\mathit{lift}[\varphi]) = \mathit{true})\,,$$

where φ is a pattern of the grammar in Figure 1. However, here in Kore we allow φ containing *meta-variables*. As a result, we modify the definition of the semantics bracket as

$$[\![\varphi]\!] \equiv \mathit{mvsc}[\varphi] \to (\mathit{deducible}\,(\mathit{lift}[\varphi]) = \mathit{true}),$$

where the lifting function $\mathit{lift}[\_]$ and the meta-variable sort constraint $\mathit{mvsc}[\_]$ are defined in Algorithm 1 and 2, respectively. Intuitively, meta-variables in an object-pattern φ are lifted to variables of the sort *KPattern* with the corresponding sort constraints. For example, the meta-variable $x{::}s$ is lifted to a variable $x{:}\mathit{KPattern}$ in K with the constraint that $\mathit{KgetSort}(x{:}\mathit{KPattern}) = \mathit{sort}(s)$. The function $\mathit{mvsc}[\_]$ collects all such meta-variable sort constraint in an object-pattern is implemented in Algorithm 2.

Algorithm 1: Lifting Function $\mathit{lift}[\_]$

Input: An object-pattern φ.
Output: The meta-representation (ASTs) of φ in K

if φ *is* $x{:}s$ **then**
  Return $\mathit{variable}(x, \mathit{sort}(s))$
else if φ *is* $x{::}s$ **then**
  Return $x{:}\mathit{KPattern} \wedge (\mathit{sort}(s) = \mathit{KgetSort}(x{:}\mathit{KPattern}))$
else if φ *is* $\varphi_1 \wedge \varphi_2$ **then**
  Return $\mathit{Kand}(\mathit{lift}[\varphi_1], \mathit{lift}[\varphi_2]$
else if φ *is* $\neg\varphi_1$ **then**
  Return $\mathit{Knot}(\mathit{lift}[\varphi_1])$
else if φ *is* $\exists x{:}s.\varphi_1$ **then**
  Return $\mathit{Kexists}(x, \mathit{sort}(s), \mathit{lift}[\varphi_1])$
else if φ *is* $\sigma(\varphi_1, \ldots, \varphi_n)$ *and* $\sigma \in \Sigma_{s_1,\ldots,s_n,s}$ **then**
  Return $\mathit{Kapplication}(\mathit{symbol}(\sigma, (\mathit{Ksort}(s_1), \ldots, \mathit{Ksort}(s_n)), \mathit{Ksort}(s)), \mathit{lift}[\varphi_1], \ldots, \mathit{lift}[\varphi_n])$

Algorithm 2: Meta-Variable Sort Constraint Collection *mvsc*

Input: An object-pattern φ
Output: The meta-variable sort constraint of φ

Collect in set W all meta-variables appearing in φ;
Let $C = \emptyset$;
foreach $x{::}s \in W$ **do**
  $C = C \cup (\mathit{sort}(s) = \mathit{KgetSort}(x{:}\mathit{KPattern}))$
Return $\bigwedge C$;

4.2 Examples of Kore

Xiaohong: Add more examples and texts here.

The BOOL module.

```
module BOOL
  syntax Bool
  syntax Bool ::= true | false | notBool(Bool)
                | andBool(Bool, Bool) | orBool(Bool, Bool)
  axiom \or(true(), false())
  axiom \exists(X:Bool, \equals(X:Bool, true()))
  axiom \equals(andBool(B1::Bool, B2::Bool),
                andBool(B2::Bool, B1::Bool))
  axiom ... ...
endmodule
```

The BOOL module (desugared).

```
module BOOL
  axiom \equals(
    #true,
    #deducible(#or(#application(#symbol("true", #nilSort, #sort("Bool")),
                                #nilPattern),
                   #application(#symbol("false", #nilSort, #sort("Bool")),
                                #nilPattern))))
  axiom \equals(
    #true,
    #deducible(#exists("X", #sort("Bool"),
                 #equals(#variable("X", #sort("Bool")),
                         #application(#symbol("true", #nilSort, #sort("Bool")),
                                      #nilPattern)))))
  axiom \implies(
    \and(\equals(#getSort(B1:Pattern), #sort("Bool")),
         \equals(#getSort(B2:Pattern), #sort("Bool"))),
    \equals(
      #true,
      #deducible(#equals(#application(#symbol("andBool",
                                         (#sort("Bool"), #sort("Bool"))
                                         #sort("Bool")),
                                   (B1:Patern, B2:Pattern)), ---- TODO
                         #application(#symbol("andBool",
                                         (#sort("Bool"), #sort("Bool"))
                                         #sort("Bool")),
                                   (B2:Patern, B1:Pattern))))))
  axiom ... ...
endmodule
```

The LAMBDA module

```
module LAMBDA
  syntax Exp
  syntax Exp ::= app(Exp, Exp) | lambda0(Exp, Exp)
  syntax #Bool ::= isLTerm(#Pattern)

  axiom \equals(
    isLTerm(#variable(X:String, #sort("Exp"))),
    true)
  axiom \equals(
    isLTerm(#application(
              #symbol("app", (#sort("Exp"), #sort("Exp")), #sort("Exp")),
              (E:Pattern, E':Pattern))),
    andBool(isLTerm(E:Pattern), isLTerm(E':Pattern)))
  axiom \equals(
    isLTerm(#exists(X:String, #sort("Exp"),
                    #application(#symbol("lambda0",
                                         (#sort("Exp"), #sort("Exp")),
                                         #sort("Exp")),
                                 (#variable(X:String, #sort("Exp")),
                                  E:Pattern))),
    isLTerm(E:Pattern))
  axiom \implies(\equals(true,
                         andBool(isLTerm(E:Pattern),
                                 isLTerm(E':Pattern))),
                 \equals(true,
                         deducible(#equals(...1,
                                           ...2))))
endmodule
```

5 Ignore Me

A proof system is a theorem generator. In K, the proof system of matching logic is captured by the functional symbol $\mathit{deducible}\colon \mathit{KPattern} \to \mathit{KBool}$, which returns Ktrue iff the argument pattern is a theorem.

We introduce the double bracket $[\![\_]\!]$, known as the semantics bracket, as follows:

$$[\![\varphi]\!] \equiv (\mathit{deducible}\,(\mathit{lift}[\varphi]) = \mathit{true})\,.$$

Intuitively, $[\![\varphi]\!]$ means that "φ is deducible". Whenever there is an inference rule (axioms are considered as rules with zero premise)

$$\frac{\varphi_1, \dots, \varphi_n}{\psi}$$

in matching logic, there is a corresponding axiom in K:

$$[\![\varphi_1]\!] \wedge \dots \wedge [\![\varphi_n]\!] \to [\![\psi]\!].$$

Inference modulo theories can be considered in the same way. For any (syntactic) matching logic theory T whose axiom set is A, we add

$$[\![\varphi]\!] \quad \text{for all } \varphi \in A$$

as axioms to K. We sometimes denote the extended theory as $\mathit{lift}[T]$ and call it the *meta-theory for* T.