Composing and Controlling Search in Reasoning Theories using Mappings

Alessandro Coglio[1], Fausto Giunchiglia[2], José Meseguer[3], and Carolyn L. Talcott[4]

[1] Kestrel Institute, Palo Alto, CA
coglio@kestrel.edu
[2] IRST and Università di Trento, Italy
fausto@irst.itc.it
[3] SRI International, Menlo Park, CA
meseguer@csl.sri.com
[4] Stanford University, CA
clt@cs.stanford.edu

Abstract. Reasoning theories can be used to specify heterogeneous reasoning systems. In this paper we present an equational version of reasoning theories, and we study their structuring and composition, and the use of annotated assertions for the control of search, as mappings between reasoning theories. We define composability and composition using the notion of *faithful inclusion mapping*, we define *annotated reasoning theories* using the notion of *erasing mapping*, and we lift composability and composition to consider also annotations. As an example, we give a modular specification of the top-level control (known as *waterfall*) of NQTHM, the Boyer-Moore theorem prover.

1 Introduction

Reasoning theories are the core of the OMRS (Open Mechanized Reasoning System(s)) framework [14] for the specification of complex, heterogeneous reasoning systems. The long-term goal of OMRS is to provide a technology which will allow the development, integration and interoperation of theorem provers (and similar systems, e.g., computer algebra systems and model checkers) in a principled manner, with minimal changes to the existing modules. Reasoning theories are the *Logic* layer of OMRS, specifying the assertions and the rules used to derive new assertions and to build derivations. There are two other layers: *Control*, specifying the search strategies, and *Interaction*, specifying the primitives which allow an OMRS to interact with its environment. The reasoning theory framework has been used to analyze the NQTHM prover [11] and to develop a modular reconstruction of the high-level structure of the ACL2 prover [3]. The latter work augments ACL2 with the ability to construct proofs. In addition, the reasoning theory framework has been used as the basis of an experiment to re-engineer the integration of the NQTHM linear arithmetic procedure implemented as a re-usable separate process [1]. A formalism for the control component of OMRS was developed and used to specify the control component of NQTHM [10]. The OMRS framework has also been used to build provably correct systems [13], and it has been used as the basis for integration of the Isabelle proof system and the Maple symbolic algebra system [4].

In this paper we study structuring and composition mechanisms for reasoning theories. We focus on the special case of equationally presented reasoning theories (ERThs). In passing to ERThs we gain concreteness, executability, and the availability of equational and rewriting tools, while keeping the originally envisioned generality and avoiding making unnecessary ontological commitments. In particular, ERThs have a strong link to equational and rewriting logic [21]. There are increasing levels of *structuring* of an equationally presented reasoning theory: ERths, nested ERThs (NERThs), and also ERThs and NERThs with annotated assertions for the control of search for derivations. Furthermore, there are horizontal *theory composition operations* at each level. The structuring and composition are formalized in terms of mappings between reasoning theories. Therefore, our work is in the spirit of "putting theories together" by means of theory mappings as advocated by Burstall and Goguen [7], that is widely used in algebraic specifications [2]. In order to simplify the presentation, we treat a restricted subcase of horizontal composition that is none-the-less adequate for interesting examples. A more general treatment in terms of pushouts of order-sorted theories [16, 22, 23] can be given.

In Section 2 we define ERTHs and their composition using the notion of *faithful inclusion mapping*. We then define *annotated reasoning theories* using the notion of *erasing mapping*, and lift composability and composition to consider also annotations. In Section 3 this is extended to NERths. As an example, in Section 4 we give a modular specification of the NQTHM top-level control (known as *waterfall*). In Section 5 we summarize and outline the future work.

Notation

We use standard notation for sets and functions. We let $\bar{\emptyset}$ denote the empty function (the function with empty domain, and whatever codomain). X^* denotes the set of all finite lists of elements in X. $[X_0 \to X_1]$ is the set of total functions, f, with domain, $\mathrm{Dom}(f) = X_0$, and range, $\mathrm{Rng}(f)$, contained in the codomain X_1. Let $(X, \leq)$ be a partial order. If $X' \subseteq X$, then $\leq \lceil X'$ is the restriction of $\leq$ to X', i.e., $\leq \lceil X' = \{(x, y) \bigm| x \in X', y \in X', x \leq y\}$. X' is *downwards closed* in X if whenever $x \leq y$ and $y \in X'$, then $x \in X'$. X' is *upwards closed* in X if whenever $y \leq x$ and $y \in X'$, then $x \in X'$.

A family Z of sets over a set I is an assignment of a set Z_i for each $i \in I$. This is notated by $Z = \{Z_i\}_{i \in I}$ and we say Z is an I-family. If $I = \{i\}$ is a singleton, then we let Z_i stand for $\{Z_i\}_{i \in I}$. If $I' \subseteq I$, the restriction of Z to I' is defined as $Z\lceil I' = \{Z_i\}_{i \in I'}$. If $I' \subseteq I$, Z is an I-family, Z' is an I'-family, then $Z' \subseteq Z$ means $Z'_i \subseteq Z_i$ for $i \in I'$. If Z_j are I_j-families for $j \in \{1, 2\}$, then we define the $(I_1 \cap I_2)$-family $Z_1 \cap Z_2 = \{(Z_1)_i \cap (Z_2)_i\}_{i \in I_1 \cap I_2}$ and the $(I_1 \cup I_2)$-family $Z_1 \cup Z_2 = \{(Z_1)_i \cup (Z_2)_i\}_{i \in I_1 \cup I_2}$ (where in the last expression we consider $(Z_1)_i = \emptyset$ if $i \in I_2 - I_1$, and $(Z_2)_i = \emptyset$ if $i \in I_1 - I_2$).

2 Equational Reasoning Theories

A reasoning theory specifies a set of *inference rules* of the form

$$l : \{\vec{cn}\}\ \vec{sq} \Rightarrow sq$$

where l is the rule label, sq is the conclusion, $\vec{sq}$ is a list of premisses, and $\vec{cn}$ is a list of side conditions. sq and the elements of $\vec{sq}$ are elements of the set, Sq, of *sequents* specified by the reasoning theory. Sequents represent the assertions to be considered. We use "sequent" in a very broad sense which includes sequent in the Gentzen/Prawitz sense [12, 24], simple formulas and judgements in the sense of Martin-Löf [19], as well as the complex data structures (carrying logical information) manipulated by the NQTHM waterfall. Elements of $\vec{cn}$ are elements of the set, Cn, of *constraints* specified by the reasoning theory. If this set is non-empty, then the reasoning theory also provides an inference machinery for deriving constraint assertions. Sequents and constraints can be schematic in the sense that they may contain place holders for missing parts. A reasoning theory also provides a mechanism for filling in missing parts: a set, I, of instantiations and an operation, $\_[\_]$, for applying instantiations to schematic entities. There is an identity instantiation and instantiations can be composed. A reasoning theory determines a set of *derivation structures* (the OMRS notion of proof) formed by instantiating inference rules and linking them by matching premisses to conclusions. We write $ds : \{\vec{cn}\}\ \vec{sq} \Rightarrow \vec{sq}'$ to indicate that ds is a derivation structure with conclusions the list of sequents $\vec{sq}'$, premisses the list $\vec{sq}$, and side conditions the list of constraints $\vec{cn}$. (See [14] for a detailed description of reasoning theories.)

2.1 Sequent signatures and ERThs

In this paper we focus on *equationally presented reasoning theories* (*ERThs*). An ERTh consists of a sequent signature and a labelled set of rules of inference. A sequent signature consists of an equational signature, Σ, (the choice of equational logic is not so important, here for concreteness we take it to be order-sorted equational logic [15]) together with a sort-indexed family of variables, a set of equations and a distinguished set of sorts – the sequent sorts. Variables are the place holders and instantiations are simply sort preserving maps from variables to terms, applied and composed in the usual way. The equational theory just serves to define the sequents; deduction over such sequents is specified by the rules of inference of the ERTh. Therefore, the expressivity of ERThs is by no means limited to first-order equational logic: other logics can be easily expressed (e.g., higher-order) by using suitable rules of inference (see for example formalizations of Lambda calculus [18], HOL and NuPrl [25], and the Calculus of Constructions [26] in the Maude Rewriting Logic implementation).

Definition: sequent signatures. A sequent signature is a tuple $eseq = (\Sigma, X, E, Q)$ where $\Sigma = (S, \leq, O)$ is an order-sorted equational signature in which S is the set of sorts, O is an $S^* \times S$-indexed family of operations, and $\leq$ is a partial order on S; X is an S-indexed family of variables; E is a set of Σ-equations; and $Q \subseteq S$ is the set of sequent sorts. We require that Q be downwards and upwards closed in S. Elements of $O_{\vec{s},s}$ are said to have rank $\vec{s}$, s, where $\vec{s}$ is the list of argument sorts (the arity) and s is the result sort, and we write $o : \vec{s} \to s$ to indicate that o is an element of $O_{\vec{s},s}$. We assume that each set X_s for $s \in S$ is countably infinite. $T_{eseq} = T_{\Sigma}(X)$ is the free order-sorted Σ-algebra over X whose elements are terms formed from elements of X by applying operations of O to lists of terms of the appropriate argument sorts. For $s \in S$, $T_{eseq,s}$ is the set of $eseq$ terms of sort s and for $S' \subseteq S$, $T_{eseq,S'}$ is the set of terms of sort s for

$s \in S'$: $T_{eseq,S'} = \bigcup_{s \in S'} T_{eseq,s}$. We further require that Σ be *coherent* [15]. For our purposes this can be taken to mean that every connected component of the sort poset has a top element, and that there is a function $ls : T_{eseq} \to S$ mapping each term to its least sort in the partial order $\leq$ on S.

An equation is a pair of terms of the same sort, thus $E \subseteq \bigcup_{s \in S} T_{eseq,s} \times T_{eseq,s}$. The rules of deduction of order-sorted equational logic [15] then define an order-sorted Σ-congruence $\equiv_E$ on T_{eseq}. We say that t, t' are E-equivalent if $t \equiv_E t'$. For an $eseq$ term t we let $[t]_E$ be the E-equivalence class of t and for any set $T \subseteq T_{eseq,S}$ we let $\{T\}_E$ be the set of E-equivalence classes of terms in T.

The set of sequents, Sq_{eseq}, presented by $eseq$, is the set of equivalence classes of terms with sorts in Q: $Sq_{eseq} = \{T_{eseq,Q}\}_E$. Instantiations are sort-preserving mappings from X to T_{eseq}, with identity and composition defined in the obvious way. Their application to sequents consists in substituting terms for variables in members of the equivalence classes.

Definition: rules and ERThs. The set of rules $Rules_{eseq}$ over a sequent signature $eseq$ is the set of pairs $(\vec{sq}, sq)$ where $\vec{sq} \in Sq^*_{eseq}$ and $sq \in Sq_{eseq}$. We fix a set of labels, Lab, to use for rule names.

An ERTh, $erth$, is a pair $(eseq, R)$ where $eseq$ is a sequent signature and $R : L \to Rules_{eseq}$ is a function from a set of labels $L \subseteq Lab$ to the set of rules over $eseq$. We write $l : \vec{sq} \Rightarrow sq \in R$ if $l \in L$ and $R(l) = (\vec{sq}, sq)$.

2.2 Composing sequent signatures and ERThs

Specifying composition of systems at the logical level amounts to specifying how deductions performed by the different systems relate to each other (e.g., how an inequality proved by a linear arithmetic decider can be used by a rewriter to rewrite a term involving an arithmetic expression). This demands that the ERThs of the systems to be composed share some language, and that the composition preserves the shared language and its meaning. We represent the sharing by theory mappings from the shared part to each component. We restrict the mappings used for composition to guarantee that composition makes sense: there should be a uniquely defined composite with mappings from the component theories that agree on the shared part, and the theory mappings should induce mappings that extend/restrict instantiations in a well-behaved manner, so that the result is a mapping of ERThs. Here we consider a very simple but natural restriction with the desired properties. The general situation will be treated in a forthcoming paper [20].

Definition: faithful inclusions. Let $eseq_j = ((S_j, \leq_j, O_j), X_j, E_j, Q_j)$ be sequent signatures for $j \in \{0, 1\}$. A faithful inclusion arrow exists from $eseq_0$ to $eseq_1$ (written $eseq_0 \hookrightarrow eseq_1$) if

1. $S_0 \subseteq S_1$, $O_0 \subseteq O_1$, $X_0 \subseteq X_1$, and $Q_0 \subseteq Q_1$;
2. $X_1 \lceil S_0 = X_0$, and $Q_1 \cap S_0 = Q_0$;
3. $\leq_1 \lceil S_0 = \leq_0$, and S_0 is downwards and upwards closed in S_1;
4. any operation in O_1 with result sort in S_0, is in O_0;

5. for terms t, t' in T_{eseq_0} having sorts in the same connected component of $\leq_0$, $t' \in [t]_{E_1}$ iff $t' \in [t]_{E_0}$.

Lemma (term.restriction): If $eseq_0 \hookrightarrow eseq_1$, then $T_{eseq_1}\lceil S_0 = T_{eseq_0}$. Therefore any sub-term of a term $t \in T_{eseq_1,S_0}$ has sort in S_0. In fact t is an $eseq_0$-term.

Lemma (incl.transitivity): If $eseq_0 \hookrightarrow eseq_1$ and $eseq_1 \hookrightarrow eseq_2$, then $eseq_0 \hookrightarrow eseq_2$.

Definition: sequent signature composition. Let $eseq_j = ((S_j, \leq_j, O_j), X_j, E_j, Q_j)$ for $j \in \{0,1,2\}$. $eseq_1$ is composable with $eseq_2$ sharing $eseq_0$ if $eseq_0 \hookrightarrow eseq_j$ for $j \in \{1,2\}$ are faithful inclusions such that $S_1 \cap S_2 = S_0$, $O_1 \cap O_2 = O_0$, $X_1 \cap X_2 = X_0$, $Q_1 \cap Q_2 = Q_0$. The composition of $eseq_1$ and $eseq_2$ sharing $eseq_0$ (written $eseq_1 +_{eseq_0} eseq_2$) is the 'least' sequent signature including $eseq_1$ and $eseq_2$. More precisely,

$$eseq_1 +_{eseq_0} eseq_2 = ((S_1 \cup S_2, \leq_1 \cup \leq_2, O_1 \cup O_2), X_1 \cup X_2, Q_1 \cup Q_2).$$

Lemma (incl.composition): If $eseq_j$ for $j \in \{0,1,2\}$ satisfy the conditions for composability, then $eseq_1 \hookrightarrow eseq_1 +_{eseq_0} eseq_2$, $eseq_2 \hookrightarrow eseq_1 +_{eseq_0} eseq_2$. Also (by transitivity) $eseq_0 \hookrightarrow eseq_1 +_{eseq_0} eseq_2$.

To avoid the need to make the shared part of a composition explicit we define a function that computes the shared part of a pair of sequent signatures and say that two sequent signatures are composable just if their shared part is faithfully included in both signatures.

Definition: shared part. Let $eseq_j = ((S_j, \leq_j, O_j), X_j, E_j, Q_j)$ be sequent signatures for $j \in \{1,2\}$. Define $shared(eseq_1, eseq_2)$ by

$$shared(eseq_1, eseq_2) = (\Sigma_0, X_0, E_0, Q_0)$$

where $\Sigma_0 = (S_0, \leq_0, O_0)$, $S_0 = S_1 \cap S_2$, $\leq_0 = \leq_1 \cap \leq_2$, $O_0 = (O_1 \cap O_2)\lceil(S_0^* \times S_0)$, $X_0 = X_1 \cap X_2\lceil S_0$, $E_0 = \{e \in (E_1 \cup E_2) \mid e \text{ is a } \Sigma_0\text{-equation}\}$, and $Q_0 = Q_1 \cap Q_2$.

Definition: composability and composition. $eseq_1$ is composable with $eseq_2$ (written $eseq_1 \bowtie eseq_2$) if

1. $shared(eseq_1, eseq_2) \hookrightarrow eseq_1$
2. $shared(eseq_1, eseq_2) \hookrightarrow eseq_2$

If $eseq_1 \bowtie eseq_2$, then their composition $eseq_1 + eseq_2$ is defined by

$$eseq_1 + eseq_2 = eseq_1 +_{shared(eseq_1, eseq_2)} eseq_2$$

The two notions of composability and composition are related as follows.

Lemma (composing): $eseq_1$ is composable with $eseq_2$ sharing $eseq_0$ if $eseq_0 = shared(eseq_1, eseq_2)$ and $eseq_1 \bowtie eseq_2$.

Lemma (ACI): Composition of sequent signatures is associative, commutative and idempotent. (Note that since composition is partial it is necessary to check both definedness and equality.)

(C) if $eseq_1 \bowtie eseq_2$ then $shared(eseq_1, eseq_2) = shared(eseq_2, eseq_1)$ and $eseq_1 + eseq_2 = eseq_2 + eseq_1$

(A) if $eseq_1 \bowtie eseq_2$ and $eseq_0 \bowtie (eseq_1 + eseq_2)$, then $eseq_0 \bowtie eseq_1$ and $(eseq_0 + eseq_1) \bowtie eseq_2$ and $eseq_0 + (eseq_1 + eseq_2) = (eseq_0 + eseq_1) + eseq_2$.

(I) $eseq \bowtie eseq$ and $eseq + eseq = eseq$

Definition: multi-composition. A special case of multiple composition is when there is a common shared part among several sequent signatures. We say that a set of sequent signatures, $\{eseq_i \mid 1 \leq i \leq k\}$, is composable if there is some $eseq_0$ such that $shared(eseq_i, eseq_j) = eseq_0$ for $1 \leq i \neq j \leq k$ and $eseq_0 \hookrightarrow eseq_i$ for $1 \leq i \leq k$. In this case, the composition of $\{eseq_i \mid 1 \leq i \leq k\}$ is just the repeated binary composition: $eseq_1 + \ldots + eseq_k$. By (**ACI**) the order in which the elements are listed does not matter.

Definition: composing ERThs. Composability and composition lift naturally to ERThs. Let $erth_j = (eseq_j, R_j)$ for $j \in \{1, 2\}$ be ERThs. Then $erth_1$ is composable with $erth_2$ (written $erth_1 \bowtie erth_2$) if $eseq_1 \bowtie eseq_2$ and the domains of R_1 and R_2 are disjoint. If $erth_1 \bowtie erth_2$, then the composition of $erth_1$ with $erth_2$ is defined by

$$erth_1 + erth_2 = (eseq_1 + eseq_2, R_1 \cup R_2)$$

2.3 Annotated sequent signatures and ERThs

Informally an *annotated sequent signature* is a pair of sequent signatures, $(eseq^a, eseq)$ together with an *erasing* mapping $\varepsilon : eseq^a \to eseq$ which "removes annotations", extracting the logical content of annotated assertions. An *annotated ERTh* is a pair of ERThs $(erth^a, erth)$ together with an erasing mapping $\varepsilon : erth^a \to erth$ such that the mapping on the underlying sequent signatures gives an annotated sequent signature and annotated rules map to derivation structures over $erth$.

An erasing map $\varepsilon : eseq^a \to eseq$ is a map of theories that maps each sort of $eseq^a$ (the annotated theory) to either a sort of $eseq$ or to the empty list of sorts (erasing it in that case). The conditions for erasing maps are given below. Erasing maps are more general than the usual maps of theories (that map sorts to sorts). Categorically, an erasing map extends to a functor $\overline{\varepsilon} : \mathcal{L}_{eseq^a} \to \mathcal{L}_{eseq}$ that preserves products and inclusions between the "Lawvere theories" associated to the order-sorted theories $eseq^a$ and $eseq$ [17].

Annotations are very useful for controlling the search for derivations, as will be illustrated in Section 4. The mapping from annotated rules to derivation structures gives for each derivation at the annotated level, a corresponding, possibly more detailed derivation in the original theory.

Definition: annotated sequent signature. Let $eseq^a = (\Sigma^a, X^a, E^a, Q^a)$ and $eseq = (\Sigma, X, E, Q)$ be sequent signatures, where $\Sigma^a = (S^a, \leq^a, O^a)$ and $\Sigma = (S, \leq, O)$. Then $\varepsilon : eseq^a \to eseq$ is an annotated sequent signature if ε, with the action on sorts and terms lifted homomorphically to lists, satisfies the following:

1. $\varepsilon : S^a \to S \cup \{[\,]\}$ such that:
   - if $s_0^a \leq^a s_1^a$, then $\varepsilon(s_0^a) \leq \varepsilon(s_1^a)$
   - if $q^a \in Q^a$, then $\varepsilon(q^a) \in Q$
   - if $s^a \in S^a$ and $\varepsilon(s^a) \in Q$ then $s^a \in Q^a$
2. $\varepsilon : X^a \to X \cup \{[\,]\}$ such that if $x^a \in X^a_{s^a}$ and $\varepsilon(s^a) = s$, then $\varepsilon(x^a) \in X_s$ and if $\varepsilon(s^a) = [\,]$, then $\varepsilon(x^a) = [\,]$ (i.e., the empty list of variables).
3. If $o^a : \vec{s}^a \to s^a$, and $\varepsilon(s^a) = s$ (not $[\,]$), then $\varepsilon(o^a(x_1^a, \ldots, x_n^a))$ is a Σ-term with sort s and free variables list $\varepsilon(x_1^a, \ldots, x_n^a)$. ε is extended homomorphically to $eseq^a$-terms.
4. If $t_0^a \in [t_1^a]_{E^a}$ then $\varepsilon(t_0^a) \in [\varepsilon(t_1^a)]_E$

If $erth^a = (eseq^a, R^a)$ and $erth = (eseq, R)$, then $\varepsilon : erth^a \to erth$ is an annotated ERTh if $\varepsilon : eseq^a \to eseq$ is an annotated sequent signature and for $l : \vec{sq}^a \Rightarrow sq^a \in R^a$ there is a derivation structure $ds : \varepsilon(\vec{sq}^a) \Rightarrow \varepsilon(sq^a)$ in the set of $erth$ derivation structures (such derivation structures are described in detail in [20]).

Now we consider the relation between annotated theories and composition.

Lemma (compose.erase): Let $\varepsilon_j : erth_j^a \to erth_j$ be annotated ERThs for $j \in \{1, 2\}$, where $erth_j^a = (eseq_j^a, R_j^a)$ and $erth_j = (eseq_j, R_j)$. Suppose that $erth_1^a \bowtie erth_2^a$, $erth_1 \bowtie erth_2$, and that ε_1, ε_2 agree on $shared(eseq_1^a, eseq_2^a)$. Then we have annotated ERThs

$$\varepsilon_1 \cap \varepsilon_2 : shared(eseq_1^a, eseq_2^a) \to shared(eseq_1, eseq_2)$$
$$\varepsilon_1 + \varepsilon_2 : eseq_1^a + eseq_2^a \to eseq_1 + eseq_2$$

where $\varepsilon_1 \cap \varepsilon_2$ is the common action of the two mappings on the shared part, and $\varepsilon_1 + \varepsilon_2$ acts as ε_1 on elements coming from $eseq_1$ and as ε_2 on elements coming from $eseq_2$ (and as $\varepsilon_1 \cap \varepsilon_2$ on the shared elements).

3 Nested Equational Reasoning Theories

To represent proof systems with conditional inference rules and other subsidiary deductions that we may want to treat as side conditions, we represent the side conditions and their rules for verification by a sub-reasoning theory called the constraint theory. This gives rise to a *nested ERTh* (*NERTh* for short). Note that this nesting organization can be further applied to the constraint theory, giving rise to higher levels of nesting. For simplicity we consider only one level here, as all the features and the necessary machinery already appear at this stage.

Definition: NERThs. A *nested sequent signature* is a tuple $neseq = (eseq, (eseq_c, R_c))$ where $eseq$ is a sequent signature with sequent sorts Q and $(eseq_c, R_c)$ is an ERTh, with sequent sorts Q_c such that $eseq \bowtie eseq_c$ and the shared part has no sequent sorts, i.e., $Q \cap Q_c = \emptyset$.

A NERTh is a pair $nerth = (neseq, R)$ where $neseq$ is a nested sequent signature and, letting $Sq = Sq_{eseq}$ and $Cn = Sq_{eseq_c}$, we have $Rules : L \to Cn^* \times Sq^* \times Sq$. Thus, labelled rules of a nested reasoning structure have the form $l : \vec{cn}, \vec{sq} \Rightarrow sq$ where

$\vec{cn}$ are the side conditions or premisses to be established by deduction in the constraint theory.

Definition: composition of NERThs. Let $neseq_j = (eseq_j, (eseq_{j,c}, R_{j,c}))$ be nested sequent signatures for $j \in \{1, 2\}$. $neseq_1$ is composable with $neseq_2$ (written $neseq_0 \bowtie neseq_1$) if:

1. $eseq_1 \bowtie eseq_2$, $eseq_{1,c} \bowtie eseq_{2,c}$, and $eseq_1{+}eseq_2 \bowtie eseq_{1,c}{+}eseq_{2,c}$;
2. $shared(eseq_1, eseq_{2,c})$ has no sequent sorts, and $shared(eseq_2, eseq_{1,c})$ has no sequent sorts.

Note that 2. together with the assumption that $neseq$ is a nested sequent signature implies that there are no sequent sorts in $shared(eseq_1{+}eseq_2, eseq_{1,c}{+}eseq_{2,c})$.

If $neseq_1 \bowtie neseq_2$, then the composition, $neseq_1{+}neseq_2$ is defined by

$$neseq_1{+}neseq_2 = (eseq_1{+}eseq_2, erth_{1,c}{+}erth_{2,c})$$

Let $nerth_j = (neseq_j, R_j)$ be NERThs for $j \in \{1, 2\}$. $nerth_1$ is composable with $nerth_2$ ($nerth_1 \bowtie nerth_2$) if $neseq_1 \bowtie neseq_2$ and the domains of R_1 and R_2 are disjoint. If $nerth_1 \bowtie nerth_2$, then the composition is defined by

$$nerth_1{+}nerth_2 = (neseq_1{+}neseq_2, R_1 \cup R_2)$$

Definition: annotated NERThs. Annotated nested sequent signatures and NERThs are defined analogously to the non-nested case. Let $nerth^a = (neseq^a, R^a)$ and $nerth = (neseq, R)$ where $neseq^a = (eseq^a, erth^a_c)$, $neseq = (eseq, erth_c)$, $erth^a_c = (eseq^a_c, R^a_c)$, and $erth_c = (eseq_c, R_c)$. Then $\varepsilon : nerth^a \to nerth$ is an annotated nested reasoning theory if there are mappings ε' and ε_c such that $\varepsilon' : eseq^a \to eseq$ is an annotated sequent signature, $\varepsilon_c : erth^a_c \to erth_c$ is an annotateed reasoning theory, ε' and ε_c agree on $shared(eseq^a, eseq^a_c)$, and $\varepsilon = \varepsilon' + \varepsilon_c$ is such that $\varepsilon : (eseq^a + eseq^a_c, R^a \cup R^a_c) \to (eseq + eseq_c, R \cup R_c)$ is an annotated reasoning theory. The final clause is needed to ensure that the annotated rules are mapped to derivations in $nerth$. In fact all that needs to be checked is that the rules in R^a map to derivations, the rest follows by the disjointness of sequents and constraints and the requiement that ε_c is an annotation erasing map on $erth^a_c$.

4 An Annotated Reasoning Theory for the NQTHM Waterfall

We demonstrate the usefulness of the concepts just presented by applying them to a non trivial case study. We describe salient aspects of a specification of the NQTHM [5, 6] waterfall as an annotated NERTh $\varepsilon_{\mathrm{W}} : nerWat^a \to nerWat$. Such an annotated NERTh can be composed with annotated NERThs for the other sub-systems of NQTHM, to produce an annotated NERTh for the whole system. To provide some background, we first describe the features of the waterfall that are relevant to our specification.

4.1 The NQTHM Waterfall

When the user asks NQTHM to prove an assertion, this is first *pre-processed*. While the details of pre-processing are not relevant here, the important fact is that a list of *clauses* (to be considered conjunctively) is produced and is fed into the waterfall. This is shown in the upper part of Fig. 1. The waterfall then tries to prove all the resulting clauses by means of six main inference processes (*simplification*, *elimination of destructors*, *cross-fertilization*, *generalization*, *elimination of irrelevance*, and *induction*), called upon each clause in turn, until no clauses are left. Each process returns a list of zero or more clauses which replace the input clause; the provability of the input clause is implied by that of the returned ones (i.e., from a logical point of view these inference processes work backwards).

The clauses manipulated by the waterfall are stored in two data structures: the *Top* and the *Pool* (shown in Fig. 1). The Top is a list of pairs where the first element is a clause and the second element is the (current) *history* of the clause. A history of a clause cl_0 is a list of triples $(hid_1, cl_1, hnfo_1), \ldots, (hid_n, cl_n, hnfo_n)$, where each triple is called a *history element*, each $hid_i \in \{\mathtt{SI}, \mathtt{ED}, \mathtt{CF}, \mathtt{GE}, \mathtt{EI}\}$ and is called *history identifier* (it identifies one of the first five main inference processes) each cl_i is a clause, and each $hnfo_i$ is a *history info*. Such a history expresses that, for $1 \leq i \leq n$, clause cl_{i-1} has been obtained from cl_i by applying the process identified by hid_i; $hnfo$ contains (process-dependent) details about such backward derivation. Histories are updated by the waterfall as the proof progresses. The Pool is a list, treated as a stack, of pairs whose first component is a clause and second component, called *tag*, is either TBP (To Be Proved) or BP (Being Proved). Just after the pre-processing, the Top is set to the list of resulting clauses, each one equipped with the empty history. The Pool is instead set to the empty list.

When the Top is non-empty, its first clause and history are fed into the simplifier, which returns a list of clauses and a history info. If the returned list is equal to the singleton list having as unique element the input clause, then the simplifier has *failed*. Otherwise, it has *succeeded*, and the returned clauses are prepended to the Top, each paired with the history obtained by extending the input history with the history element consisting of SI, the input clause, and the history info returned by the simplifier. If the simplifier fails, the clause and history are fed into the elimination of destructors, cross-fertilization, generalization, and elimination of irrelevance processes (in this order), until one succeeds. As soon as one process applies successfully, the output clauses are added to the Top analogously to the case of the simplifier. If none of the above processes succeeds, then the clause, paired with the tag TBP, is pushed onto the Pool (and the history is discarded), unless it is the empty clause (i.e., with no literals), in which case NQTHM stops with a failure. See the rightmost part of Fig. 1.

When the Top is empty, if also the Pool is empty, then NQTHM terminates with success (the conjecture has been proved). Otherwise, the waterfall performs the following actions. If the clause at the head of the Pool is tagged by BP, then the clause and the tag are popped (that means that the clause has been proved; see below). When the clause at the head is tagged by TBP, another clause in the Pool is searched, which subsumes it (i.e., the clause at the head is an instance of the subsuming one). In case one clause is found, then if it is tagged by BP, then NQTHM stops with a failure (because a loop in

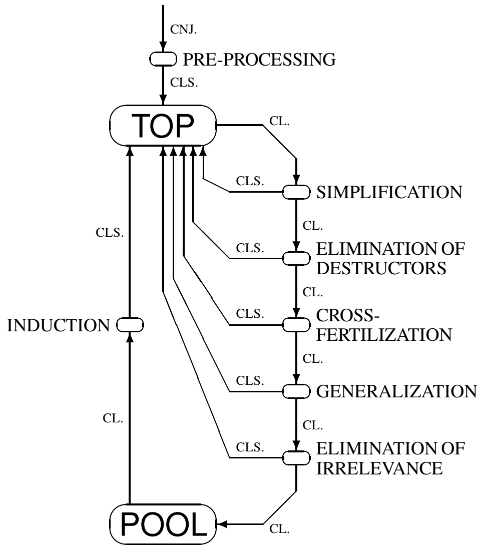

Fig. 1. The NQTHM waterfall.

inventing inductions is likely to be taking place); if instead it is tagged by `TBP`, then the clause and tag at the head are just popped (because if the subsuming clause eventually gets proved, also the subsumed one is provable), and things go on as above (i.e., when the head clause is tagged by `BP`, it is popped, etc.). If no subsuming clause is found, the tag is changed to `BP` and the clause is fed into the induction process. In case the induction process cannot invent any induction, NQTHM stops with a failure. Otherwise the induction process produces a list of clauses which are prepended to the Top, each paired with the empty history. See the leftmost part of Fig. 1. In this second case, the clause and history at the head of the Top are fed into the simplifier, and things go on as previously explained. So, note that when the Top is empty and the head clause of the Pool is tagged by `BP`, this means that it has been really proved (as said above).

We now explain in detail an important feature which we have omitted above for ease of understanding. The feature is motivated as follows: after inventing an induction scheme, it is heuristically convenient to prevent the rewriting of terms appearing in the induction hypothesis, and to force the rewriting of terms appearing in the induction conclusion (so that the hypothesis can be used to prove the conclusion). For this purpose, the induction process also produces two sets of terms as outputs (those appearing in the hypothesis and those in the conclusion), which are fed into the simplifier. Anyway, this special treatment of the terms is only performed the first time the simplifier is called upon a clause produced by induction. To achieve that, as soon as the simplifier fails upon a clause, its history is extended with a history element consisting of a (sixth) *settle-down* history identifier `SD`, the clause itself, and the empty history info (i.e., carrying no information). `SD` indicates that the clause has "settled down" with respect to the special treatment of terms appearing in induction hypothesis and conclusion. The clause and the new history are then fed into the simplifier. However, this does not happen if `SD` is already present in the history; in that case the clause and the history are just fed into the elimination of destructors process. The simplifier ignores the two input sets of terms if and only if `SD` is present in the input history.

4.2 The NERTh *nerWat*

Clauses are specified by the sequent signature $esCl$, which includes a sort Cl for clauses, as well as other sorts and operations to build clauses (e.g., specifying literals and disjunction), which we do not describe here. The waterfall manipulates lists (logically, conjunctions) of clauses. These are specified by the sequent signature $esCls$, with $esCl \hookrightarrow esCls$. $esCls$ includes a sort Cls for conjunctions of clauses, with $Cl \leq Cls$, and two operations, $\top : \rightarrow Cls$ and $(\_ \wedge \_) : Cls, Cls \rightarrow Cls$, for the empty clause conjunction and to conjoin clause conjunctions. $esCls$ contains equations stating that $(\_ \wedge \_)$ is commutative, associative, idempotent, and has $\top$ as identity. In other words, clause conjunctions are treated as sets.

Although not explicitly mentioned in the previous subsection, all the proving activity performed by NQTHM happens in the context of a database of axioms (of various kinds) with heuristic information (e.g., by which inference techniques a certain axiom should be used). From the logical point of view, this database represents a theory in which theorems can be proved. The sequent signature $esTh$ includes a sort Th for theories, as well as other sorts and operations to build theories (not presented here).

$esTh$ and $esCls$ are composable, and their composition $esThCl = esTh + esCls$ constitutes the base to build sequents formalizing the logical assertions manipulated by the waterfall, by means of the sequent signatures $esWat_0$ and $esWat_1$, with $esThCl \hookrightarrow esWat_0$ and $esThCl \hookrightarrow esWat_1$. $esWat_0$ contains a sequent sort Sq_{W} (i.e., $Sq_{\mathrm{W}} \in Q$) and an operation $(\_ \vdash_{\mathrm{W}} \_) : Th, Cls \rightarrow Sq_{\mathrm{W}}$. A sequent of the form $(th \vdash_{\mathrm{W}} \widehat{cl})$ asserts that all the clauses in $\widehat{cl}$ are provable from axioms in th. $esWat_1$ contains sequent sorts Sq_{SI}, Sq_{ED}, Sq_{CF}, Sq_{GE}, Sq_{EI}, and Sq_{IN}, and operations $(\_ \vdash_{\mathrm{SI}} \_ \rightarrow \_) : Th, Cls, Cl \rightarrow Sq_{\mathrm{SI}}$, $(\_ \vdash_{\mathrm{ED}} \_ \rightarrow \_) : Th, Cls, Cl \rightarrow Sq_{\mathrm{ED}}$, $(\_ \vdash_{\mathrm{CF}} \_ \rightarrow \_) : Th, Cls, Cl \rightarrow Sq_{\mathrm{CF}}$, $(\_ \vdash_{\mathrm{GE}} \_ \rightarrow \_) : Th, Cls, Cl \rightarrow Sq_{\mathrm{GE}}$, $(\_ \vdash_{\mathrm{EI}} \_ \rightarrow \_) : Th, Cls, Cl \rightarrow Sq_{\mathrm{EI}}$, and $(\_ \vdash_{\mathrm{IN}} \_ \rightarrow \_) : Th, Cls, Cl \rightarrow Sq_{\mathrm{IN}}$. A sequent of the form $(th \vdash_{\mathrm{SI}} \widehat{cl} \rightarrow cl)$ asserts that the provability of the clauses in $\widehat{cl}$ implies that of cl, where $\widehat{cl}$ are obtained by simplification upon cl (in the context of theory th). The other kinds of sequents have analogous meaning (SI, ED, etc. identify the six main inference processes of NQTHM).

The logical inferences performed by the waterfall are specified by suitable rules of inference over the nested sequent systems $nesWat_0 = (esWat_0, erSbsm)$ and $nesWat_1 = (esWat_1, \mathtt{mterth})$. $\mathtt{mtrth}$ is the empty ERTh (i.e., with no sorts, no operations, etc.). $erSbsm$ is an ERTh including a sequent sort Cn_{SBSM} and an operation $Subsumes(\_, \_) : Cl, Cl \rightarrow Cn_{\mathrm{SBSM}}$. A sequent of the form $Subsumes(cl, cl')$ asserts that clause cl subsumes clause cl', and $erSbsm$ contains rules that specify subsumption (i.e., when $Subsumes(cl, cl')$ can be derived). The NERTh $nerWat_0$ consists of $nesWat_0$ and the following rules:

$$\mathsf{Qed}: \quad \Longrightarrow \quad th \vdash_{\mathrm{W}} \top\,,$$

$$\mathsf{ElimSubsumed}: \quad \begin{array}{l} \{Subsumes(cl, cl')\} \\ th \vdash_{\mathrm{W}} \widehat{cl} \wedge cl \quad \Longrightarrow \quad th \vdash_{\mathrm{W}} \widehat{cl} \wedge cl \wedge cl'\,. \end{array}$$

Qed specifies that the empty conjunction of clauses is provable in any theory, and models the end of the proof when there no more clauses to prove in the Top or Pool. ElimSubsumed specifies that to prove a conjunction of clauses it is sufficient to prove all except one, provided this one is subsumed by another one in the conjunction; note the constraint (sequent of $erSbsm$) used as a side condition. This rule models the elimination of a sumsumed clause from the Pool. The NERTh $nerWat_1$ consists of $nesWat_1$ and six rules, one of which is

$$\mathsf{CallSimp}: \quad th \vdash_{\mathrm{SI}} \widehat{cl}' \rightarrow cl \qquad th \vdash_{\mathrm{W}} \widehat{cl} \wedge \widehat{cl}' \quad \Longrightarrow \quad th \vdash_{\mathrm{W}} \widehat{cl} \wedge cl\,.$$

CallSimp specifies that to prove the conjunction of cl and $\widehat{cl}$ it is sufficient to prove the conjunction of $\widehat{cl}'$ and $\widehat{cl}$, where the $\widehat{cl}'$ are the result of simplifying cl. Its backward application models the invocation of the simplification process upon a clause and the replacement of the clause with those resulting from simplification. The other five rules, CallElDes, CallCrFer, CallGen, CallElIrr, and CallInd, are analogous. $nerWat_0$ and $nerWat_1$ are composable, and $nerWat = nerWat_0 + nerWat_1$.

4.3 The NERTh *nerWat*^{a}

The sequents and rules just presented do not specify anything about clause histories, division between Top and Pool, and so on. That, in fact, constitutes non-logical (i.e., control) information, and is specified, along with its manipulation, by annotated sequents and rules in $nerWat^a$.

Histories are specified by the sequent signature $esHst$, with $esCls \hookrightarrow esHst$. $esHst$ includes a sort Hid and seven constants $\mathtt{SI}, \mathtt{ED}, \mathtt{CF}, \mathtt{GE}, \mathtt{EI}, \mathtt{IN}, \mathtt{SD} : \rightarrow Hid$ for history identifiers, as well as a sort $Hinfo$ and operations for process-specific history information. It also includes a sort $Helem$ and an operation $[\_,\_,\_] : Hid, Cl, Hinfo \rightarrow Helem$ for history elements; a sort Hst for histories, with $Helem \leq Hst$, and operations $\emptyset : \rightarrow Hst$ and $(\_ \cdot \_) : Hst, Hst \rightarrow Hst$ for the empty history and to concatenate histories, plus equations stating that $(\_ \cdot \_)$ is associative and has identity $\emptyset$ (i.e., histories are treated as lists). Clauses paired with histories are specified by a sort ClH and an operation $(\_/\_) : Cl, Hst \rightarrow ClH$; their conjunctions are specified by a sort $ClHs$, with and $ClH \leq ClHs$, and operations $\top : \rightarrow ClHs$ and $(\_ \wedge \_) : ClHs, ClHs \rightarrow ClHs$, which is associative and has identity $\top$ (these conjunctions are thus treated as lists, as opposed to sets; in fact, the ordering of the elements of a list constitutes control information).

Pool tags are formalized by the sequent signature $esTag$, with $esCls \hookrightarrow esTag$, which includes a sort ClT for tagged clauses, and two operations $\mathtt{TBP}(\_) : Cl \rightarrow ClT$ and $\mathtt{BP}(\_) : Cl \rightarrow ClT$ to tag clauses. Conjunctions of tagged clauses are specified by a sort $ClTs$, with $ClT \leq ClTs$, and two operations $\top : \to ClTs$ and $(\_ \wedge \_) : ClTs, ClTs \to ClTs$; the latter is associative and has identity $\top$.

The sequent signature $esDB$ contains a sort DB for databases of axioms (with heuristic information), as well as sorts and operations to build them. $esDB$, $esHst$, and $esTag$ are composable, and $esDBHstTag = esDB + esHst + esTag$ constitutes the base to build the annotated sequents manipulated by the waterfall. These are specified by sequent signatures $esWat^a_0$ and $esWat^a_1$, with $esDBHstTag \hookrightarrow esWat^a_0$ and $esDBHstTag \hookrightarrow esWat^a_1$. $esWat^a_0$ contains a sequent sort $Sq^{\mathrm{A}}_{\mathrm{W}}$ and an operation $(\_ \vdash_{\mathrm{W}} \_;\_) : DB, ClHs, ClTs \to Sq^{\mathrm{A}}_{\mathrm{W}}$. An (annotated) sequent of the form $(db \vdash_{\mathrm{W}} \widehat{clh}; \widehat{clt})$ asserts that the clauses in $\widehat{clh}$ and $\widehat{clt}$ are provable from the axioms in db. It also embeds control information: $\widehat{clh}$ and $\widehat{clt}$ constitute the Top and the Pool, with histories and tags; db contains heuristic information for the axioms. $esWat^a_1$ contains sequent sorts $Sq^{\mathrm{A}}_{\mathrm{SI}}$, $Sq^{\mathrm{A}}_{\mathrm{ED}}$, $Sq^{\mathrm{A}}_{\mathrm{CF}}$, $Sq^{\mathrm{A}}_{\mathrm{GE}}$, $Sq^{\mathrm{A}}_{\mathrm{EI}}$, and $Sq^{\mathrm{A}}_{\mathrm{IN}}$, and operations $(\_ \vdash_{\mathrm{SI}} \_,\_ \to \_,\_) : DB, Cls, Hinfo, Cl, Hst \to Sq^{\mathrm{A}}_{\mathrm{SI}}$, $(\_ \vdash_{\mathrm{ED}} \_,\_ \to \_,\_) : DB, Cls, Hinfo, Cl, Hst \to Sq^{\mathrm{A}}_{\mathrm{ED}}$, $(\_ \vdash_{\mathrm{CF}} \_,\_ \to \_,\_) : DB, Cls, Hinfo, Cl, Hst \to Sq^{\mathrm{A}}_{\mathrm{CF}}$, $(\_ \vdash_{\mathrm{GE}} \_,\_ \to \_,\_) : DB, Cls, Hinfo, Cl, Hst \to Sq^{\mathrm{A}}_{\mathrm{GE}}$, $(\_ \vdash_{\mathrm{EI}} \_,\_ \to \_,\_) : DB, Cls, Hinfo, Cl, Hst \to Sq^{\mathrm{A}}_{\mathrm{EI}}$, $(\_ \vdash_{\mathrm{IN}} \_,\_ \to \_) : DB, Cls, Hinfo, Cl \to Sq^{\mathrm{A}}_{\mathrm{IN}}$. An (annotated) sequent of the form $(db \vdash_{\mathrm{SI}} \widehat{cl}, hnfo \to cl, hst)$ asserts that $\widehat{cl}$ and $hnfo$ are the result of the simplification process when called upon cl with history hst. The other kinds of (annotated) sequents have analogous meaning.

The nested sequent signature $nesWat^a_0$ consists of $esWat^a_0$ and nested ERThs for various constraints (obtained by composition) whose details we omit here. The NERTh $nerWat^a_0$ consists of $nesWat^a_0$ and some (annotated) rules of inference, including

$$\mathsf{Qed}^a : \quad \Longrightarrow \quad db \vdash_{\mathrm{W}} \top; \top \,,$$

$$\begin{array}{ll}\mathsf{ElimSubsumed}^a : & \{Subsumes(cl, cl')\} \\ & \quad db \vdash_{\mathrm{W}} \widehat{clh}; \widehat{clt} \wedge \mathtt{TBP}(cl') \wedge \widehat{clt}' \quad \Longrightarrow \\ & \quad db \vdash_{\mathrm{W}} \widehat{clh}; \mathtt{TBP}(cl) \wedge \widehat{clt} \wedge \mathtt{TBP}(cl') \wedge \widehat{clt}' \,,\end{array}$$

which constitute the annotated counterparts of rules Qed and ElimSubsumed previously shown. Note that the subsuming clause cl' is required to be tagged by TBP. The elimination of a proved clause from the Pool is formalized by

$$\mathsf{ElimProved}^a : \quad db \vdash_{\mathrm{W}} \top; \widehat{clt} \quad \Longrightarrow \quad db \vdash_{\mathrm{W}} \top; \mathtt{BP}(cl) \wedge \widehat{clt} \,,$$

which eliminates the clause at the head of the Pool if it is tagged by BP and the Top is empty (all these rules must be read backwards). Finally, the pushing of a clause from the Top onto the Pool, and the settling-down of a clause are specified by

$$\begin{array}{ll}\mathsf{Push}^a : & \{NonEmpty(cl)\} \\ & \quad db \vdash_{\mathrm{W}} \widehat{clh}; \mathtt{TBP}(cl) \wedge \widehat{clt} \quad \Longrightarrow \quad db \vdash_{\mathrm{W}} (cl/hst) \wedge \widehat{clh}; \widehat{clt} \,,\end{array}$$

$$\begin{array}{ll}\mathsf{SettleDown}^a : & \{NoneSD(hst)\} \\ & \quad db \vdash_{\mathrm{W}} (cl/([\mathtt{SD}, cl, \emptyset] \cdot hst)) \wedge \widehat{clh}; \widehat{clt} \quad \Longrightarrow \\ & \quad db \vdash_{\mathrm{W}} (cl/hst) \wedge \widehat{clh}; \widehat{clt} \,.\end{array}$$

The side conditions $NonEmpty(cl)$ and $NoneSD(hst)$ respectively require that cl is non-empty (i.e., it has at least one literal) and that hst does not contain the history identifier SD. $\emptyset : \rightarrow Hinfo$ is the empty history information.

The nested sequents signature $nesWat_1^a$ consists of $esWat_1^a$ and a nested ERThs for various constraints whose details we omit here. The NERTh $nerWat_1^a$ consists of $nesWat_1^a$ and some (annotated) rules of inference, including

$$\mathsf{CallSimp}^a : \quad \left\{\widehat{cl} \neq cl\right\} \quad \begin{array}{l} db \vdash_{\mathrm{SI}} \widehat{cl}, hnfo \rightarrow cl, hst \\ db \vdash_{\mathrm{W}} atchst(\widehat{cl}, [\mathtt{SI}, cl, hnfo] \cdot hst) \wedge \widehat{clh}; \widehat{clt} \quad \Longrightarrow \\ db \vdash_{\mathrm{W}} (cl/hst) \wedge \widehat{clh}; \widehat{clt}\ , \end{array}$$

which constitutes the annotated counterpart of the CallSimp. Read backwards, it specifies that the clause and history at the head of the Top are fed into the simplifier and replaced by the clauses returned by it, all paired with the suitably extended history ($atchst(\_,\_) : Cls, Hst \rightarrow ClHs$ pairs each clause of a conjunction with a history, as specified by suitable equations)[1]. The side condition requires that the simplifier does not fail. The rules CallElDes^a, CallCrFer^a, CallGen^a, CallElIrr^a, and CallInd^a are analogous. $nerWat_0^a$ and $nerWat_1^a$ are composable, and $nerWat^a = nerWat_0^a + nerWat_1^a$.

4.4 The Erasing Mapping

The erasing mapping removes control information. Histories and Pool tags constitute control information. Therefore, we have, for instance, that $\varepsilon_{\mathrm{W}}(Hst) = [\,]$, $\varepsilon_{\mathrm{W}}(ClHs) = \varepsilon_{\mathrm{W}}(ClTs) = Cls$, and $\varepsilon_{\mathrm{W}}(cl/hst) = \varepsilon_{\mathrm{W}}(\mathtt{TBP}(cl)) = cl$. Since clauses tagged by BP are logically redundant, they constitute control information altogether, as specified by $\varepsilon_{\mathrm{W}}(\mathtt{BP}(cl)) = [\,]$. Databases are mapped to their underlying theories, $\varepsilon_{\mathrm{W}}(DB) = Th$. Annotated sequents are mapped to their non-annotated counterparts, e.g., $\varepsilon_{\mathrm{W}}(Sq_{\mathrm{W}}^{\mathrm{A}}) = Sq_{\mathrm{W}}$, $\varepsilon_{\mathrm{W}}(db \vdash_{\mathrm{W}} \widehat{clh}; \widehat{clt}) = (\varepsilon_{\mathrm{W}}(db) \vdash_{\mathrm{W}} \varepsilon_{\mathrm{W}}(\widehat{clh}) \wedge \varepsilon_{\mathrm{W}}(\widehat{clt}))$, $\varepsilon_{\mathrm{W}}(Sq_{\mathrm{SI}}^{\mathrm{A}}) = Sq_{\mathrm{SI}}$, and $\varepsilon_{\mathrm{W}}(db \vdash_{\mathrm{SI}} \widehat{cl}, hnfo \rightarrow cl, hst) = (\varepsilon_{\mathrm{W}}(db) \vdash_{\mathrm{SI}} \widehat{cl} \rightarrow cl)$.

By applying the erasing mapping to the premisses, conclusion, and side condition (which gets erased altogether) of CallSimp^a, we obtain an instance of CallSimp, thereby guaranteeing the existence of a derivation structure as required by the formal definition in Sect. 2.3. If we apply the erasing mapping to the premiss and to the conclusion of Push^a, we respectively obtain $(th \vdash_{\mathrm{W}} \widehat{cl}' \wedge cl \wedge \widehat{cl}'')$ and $(th \vdash_{\mathrm{W}} cl \wedge \widehat{cl}' \wedge \widehat{cl}'')$ (where db, $\widehat{clh}$, and $\widehat{clt}$ are respectively mapped to th, $\widehat{cl}'$, and $\widehat{cl}''$), which are the same sequent. Therefore, this rule maps to a trivial derivation structure (the side condition is just erased). In other words, it only modifies control information, leaving logical content unaltered; the same holds for ElimProved^a and SettleDown^a.

[1] In the actual LISP code of NQTHM, the sets of terms appearing in the induction hypothesis and conclusion are passed as separate arguments to the simplifier. However, we believe they can be more nicely and elegantly specified (and implemented as well) as history information produced by the induction process. We have therefore introduced a seventh history identifier for induction, and specified that induction returns a history information together with the clauses constituting the induction schema.

5 Conclusions and Future Work

In this paper we have formalized the composition and nesting of ERThs by means of faithful inclusions. We have also formalized the concept of annotated (N)ERThs by means of erasing mappings. Working in the special case of ERThs, the mappings we need are well understood notions of mappings of theories in a logic. Mappings for a very general notion of reasoning theory are developed in [20]. The mappings used in this paper provide the starting point for developing a more general theory of composition and annotation. There is also work in progress to generalize the notion of faithful inclusion to allow a richer collection of mappings to be used directly for composition.

While annotations can be used to specify search information and its manipulation, in order to fully specify an OMRS we also need a formal way to specify the strategy of application of annotated rules. In [9] this is done by means of a tactical language, and a tactical language is also used in [3]. We are working towards the development of a general formalism to capture tactical languages and also more general mechanisms; given the close connections between reasoning theories and rewriting logic studied in [20], such formalisms may perhaps be fruitfully related to the internal strategy languages of rewriting logic [8]. What remains to give a full specification of an OMRS is its interaction level. A formalism for this has still to be developed.

Acknowledgements The authors would like to thank the anonymous referees for helpful criticisms. This research was partially supported by ONR grants N00014-94-1-0857 and N00014-99-C-0198, NSF grants CRR-9633363 and 9633363, DARPA/Rome Labs grant AF F30602-96-1-0300, and DARPA/NASA contract NAS2-98073.

References

1. A. Armando and S. Ranise. From Integrated Reasoning Specialists to "Plug-and-Play" Reasoning Components. In *Fourth International Conference Artificial Intelligence And Symbolic Computation (AISC'98)*, 1998. also available as DIST Technical Report 97-0049, University of Genova, Italy.
2. E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors. *Algebraic Foundations of Systems Specifications*. IFIP State-of-the-Art Reports. Springer, 1999.
3. P. G. Bertoli. *Using OMRS in Practice: A Case Study with ACL2.* PhD thesis, University of Rome 3, 1997.
4. P.G. Bertoli, J. Calmet, K. Homann, and F. Giunchiglia. Specification and Integration of Theorem Provers and Computer Algebra Systems. In *Fourth International Conference On Artificial Intelligence and Symbolic Computation (AISC'98)*, Lecture Notes in AI. Springer Verlag, 1998. also Technical Report 9804-03, IRST, Trento, Italy, April 1998.
5. R. S. Boyer and J. S. Moore. *A Computational Logic.* Academic Press, 1979.
6. R. S. Boyer and J. S. Moore. *A Computational Logic Handbook.* Academic Press, 1988.
7. R. M. Burstall and J. A. Goguen. The semantics of Clear, a specification language. In *Proc. 1979 Copenhagen Winter School on Abstract Software Specification*, volume 86 of *Lecture Notes in Computer Science*, pages 292–332. Springer-Verlag, 1980.
8. M. Clavel and J. Meseguer. Reflection and strategies in rewriting logic. In *Rewriting Logic Workshop'96*, number 4 in Electronic Notes in Theoretical Computer Science. Elsevier, 1996. URL: `http://www.elsevier.nl/locate/entcs/volume4.html`.

9. A. Coglio. The control component of OMRS. Master's thesis, University of Genova, Italy, 1996.
10. A. Coglio. Definizione di un formalismo per la specifica delle strategie di inferenza dei sistemi di ragionamento meccanizzato e sua applicazione ad un sistema allo stato dell'arte. Master's thesis, 1996. Master thesis, DIST - University of Genoa (Italy).
11. A. Coglio, F. Giunchiglia, P. Pecchiari, and C. Talcott. A logic level specification of the nqthm simplification process. Technical report, IRST, University of Genova, Stanford University, 1997.
12. G. Gentzen. *The Collected Papers of Gerhard Gentzen*. North-Holland, 1969. edited by Szabo, M. E.
13. F. Giunchiglia, P. Pecchiari, and A. Armando. Towards provably correct system synthesis and extension. *Future Generation Computer Systems*, 12(458):123–137, 1996.
14. F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning theories: Towards an architecture for open mechanized reasoning systems. In *Workshop on Frontiers of Combining Systems FROCOS'96*, 1996.
15. Joseph Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.
16. A. Haxthausen and F. Nickl. Pushouts of order-sorted algebraic specifications. In *Algebraic Methodology and Software Technology (AMAST'96)*, number 1101 in Lecture Notes in Computer Science, pages 132–148. Springer, 1996.
17. Narciso Martí-Oliet and José Meseguer. Inclusions and subtypes i: First-order case. *J. Logic and Computation*, 6(3):409–438, 1996.
18. Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. In D. Gabbay, editor, *Handbook of Philosophical Logic*. Kluwer Academic Publishers, 1997.
19. P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.
20. J. Meseguer and C. Talcott. Reasoning theories and rewriting logic. (in preparation).
21. J. Meseguer and C. Talcott. Mapping OMRS to Rewriting Logic. In C. Kirchner and H. Kirchner, editors, *2nd International Workshop on Rewriting Logic and its Applications, WRLA'98*, volume 15 of *Electronic Notes in Theoretical Computer Science*, 1998. URL: `http://www.elsevier.nl/locate/entcs/volume15.html`.
22. José Meseguer. Membership algebra as a semantic framework for equational specification. In F. Parisi-Presicce, editor, *12th International Workshop on Abstract Data Types (WADT'97)*, number 1376 in Lecture Notes in Computer Science, pages 18–61. Springer, 1998.
23. T. Mossakowski. Colimits of order-sorted specifications. In F. Parisi-Presicce, editor, *12th International Workshop on Abstract Data Types (WADT'97)*, number 1376 in Lecture Notes in Computer Science, pages 316–322. Springer, 1998.
24. D. Prawitz. *Natural Deduction: A Proof-theoretical Study*. Almquist and Wiksell, 1965.
25. M-O. Stehr and J. Meseguer. The HOL-Nuprl connection from the viewpoint of general logics. manuscript, SRI International, 1999.
26. M-O. Stehr and J. Meseguer. Pure type systems in rewriting logic. In *Workshop on Logical Frameworks and Meta-languages*, 1999.