Electronic Notes in Theoretical Computer Science 23 No. 3 (1999)
URL: `http://www.elsevier.nl/locate/entcs/volume23.html` 18 pages

The Control Component of Open Mechanized Reasoning Systems [1]

Alessandro Armando [a,b] Alessandro Coglio [c]
Fausto Giunchiglia [d,e]

[a] *DIST, University of Genova, 16145 Genova, Italy*

[b] *LORIA-INRIA, 54602 Villers les Nancy, France*

[c] *Kestrel Institute, 94304 Palo Alto, California (U.S.A.)*

[d] *DISA, University of Trento, 38100 Trento, Italy*

[e] *IRST (Inst. for Scient. and Techn. Research), 38050 Trento, Italy*

Abstract

We are interested in integrating mechanized reasoning systems such as, e.g., Theorem Provers and Computer Algebra Systems. Our approach to the problem is to provide a framework for specifying mechanized reasoning systems and to use specifications as a starting point for integration. We build on top of the work presented in [9] which introduces the notion of Open Mechanized Reasoning Systems (OMRS) as a specification framework for integrating reasoning systems. An OMRS specification consists of three components: the logic component, the control component, and the interaction component. In this paper we focus on the control level and propose to specify the control component by first adding control knowledge to the data structures representing the logic by means of *annotations*, and then by specifying proof strategies via tactics. To show the adequacy of the approach we present and discuss a structured specification of the top-level inference strategy of NQTHM as a set of cooperating specialized reasoning modules.

1 Introduction

We are interested in integrating mechanized reasoning systems such as, e.g., Theorem Provers (TPs) and Computer Algebra Systems (CASs). The interest in this problem stems from the consideration that even though a variety of reasoning systems capable of very sophisticated reasoning activities in specific domains are now available, the services provided by each single system hardly

[1] We especially thank Paolo Pecchiari for his initial contributions to this work, and Carolyn Talcott for contributing to the technical development of the theory.

encompass the wide range of functionalities needed in real-world applications (e.g., the development of a mathematical theory, the development and validation of hardware and software components). However, it is often the case that functionalities missing in a system are available in another. This is particularly true for TPs and CASs due to the complementary nature of the services provided (proving and computing, respectively).

By looking at the relevant literature it turns out that there are essentially two possible strategies to cope with the problem: system extension and system integration.

- *System extension* amounts to the implementation and addition of new functionalities to the system of choice. This solution has the advantage that new functionalities can be programmed so to fit the requirements of the host system. The obvious drawback is that a considerable implementation effort is often necessary. Moreover the level of sophistication of state-of-the-art implementations is very difficult to achieve. Exemplary instances of projects and experiences based on this approach is described in [6,5,12].
- *System integration* amounts to directly using the services provided by an existing system (seen as an external and independent reasoner). In this case no (or minor) implementational effort is necessary and the advanced features and optimizations of existing systems can be directly used. The difficulty is here due to the fact that reasoning systems are usually conceived and built as stand-alone entities and—as a consequence—an effective combination can be very difficult to achieve. A set of representative experiences showing the viability of the approach is described in [1,13,14,2].

In both cases the main source of difficulty is the complexity of the services provided by state-of-the-art implementations.

We build on top of the work presented in [9] which introduces the notion of Open Mechanized Reasoning Systems (OMRS) as a specification framework for extending or integrating reasoning systems. An OMRS specification consists of three components: the *logic* component (specifying the assertions manipulated by the system and the elementary deductions upon them), the *control* component (specifying the inference strategies of the system), and the *interaction* component (specifying the interaction of the system with the external world). While the problems arising in integrating reasoning systems at the logic level have been addressed in [9], in this paper we focus on the control level. We propose to specify the control component by:

(i) adding control knowledge to the data structures representing the logic by means of *annotations*; this leads naturally to an extended notion of inference which accounts for the simultaneous manipulation of logic and control information;

(ii) specifying proof strategies via *tactics*, i.e., programs whose execution can only deliver provable facts.

To give evidence of the adequacy of the approach we show how the resulting specification framework allows for a structured specification of the top-level inference strategy of NQTHM (called the *waterfall*) as a set of cooperating specialized reasoning modules. The approach allows us *(i)* to provide a detailed and formal account of all the information exchanged by the modules, *(ii)* to neatly distinguish between the logic and the control components of the information exchanged, and *(iii)* to describe in detail the way the specialized modules are marshalled by NQTHM.

Even though the case study we consider does not address directly the integration of CASs and TPs, it exemplifies an important integration problem arising in such a context, i.e. the interplay between a general purpose reasoner provided by a TP and a library of specialized procedures provided by a CAS. Furthermore—as shown in [2]—the OMRS framework can be easily adapted to support both the specification of the computational services provided by CASs and their interaction with TPs. More in general, an OMRS specification can be used to support a variety of fundamental activities, ranging from the design and the implementation phases up to the formal analysis of the properties of reasoning systems and the synthesis of provably correct reasoning components. For instance, [3] shows the application of the OMRS framework to support the activity of building a provable correct version of the ACL2 prover [15].

The paper is organized as follows. Section 2 gives an overview of our theoretical framework: we start by giving a formal account of the logic component by introducing the notion of reasoning theory (Sect. 2.1); we then focus on the control component by defining the notion of annotated reasoning theory (Sect. 2.2) and the notion of tactic theory (Sect. 2.3); we finally illustrate how reasoning theories, annotated reasoning theories, and tactic theories for different modules can glued together yielding a composite specification (Sect. 2.4). Section 3 gives a brief (but precise) description of the *waterfall* of NQTHM. The OMRS specification of NQTHM is presented in Sect. 4.

Notations. We will make use of the following mathematical notations. Given a set I, an I-indexed family S is an assignment of a set to each $i \in I$. S_i denotes the set assigned to index i. If $I' \subseteq I$, then $S_{I'}$ denotes the set $\bigcup_{i \in I'} S_i$, and $S \mid I'$ denotes the restriction of S to I' (which is a family). If S is an I-indexed family and S' is an I'-indexed family: we have $S \subseteq S'$ if and only if $I \subseteq I'$ and $S_i \subseteq S'_i$ for all $i \in I$; furthermore $S \cap S'$ is the $(I \cap I')$-family defined by $(S \cap S')_i = S_i \cap S'_i$ for all $i \in I \cap I'$; finally $S \cup S'$ is the $(I \cup I')$-family defined by $(S \cup S')_i = S_i \cup S'_i$ for all $i \in I \cup I'$ (where $S_i = \emptyset$ if $i \in I' - I$ and $S'_i = \emptyset$ if $i \in I - I'$). Given a set A, A^* is the set of all finite sequences of elements of A. We use $[\,]$, $[h|t]$, $s@t$ to denote the empty sequence, the sequence with head h and tail the sequence t, and the concatenation of the sequences s and t, respectively.

2 Overview of the Theory

2.1 Reasoning Theories

The logic component of an OMRS is specified by a *reasoning theory* (*RTh*) [9]. An RTh is a mathematical object that, roughly speaking, consists of a set of *sequents* (i.e., assertions) and a set of *inference rules* over such sequents. Sequents can be schematic, i.e. contain place-holders for pieces of syntax. An RTh in fact comes equipped with a set of *instantiations* that can be applied to sequents to fill in schematic parts. An RTh defines a set of *reasoning structures*, i.e. graphs labeled by sequents and rules. The notion of reasoning structure generalizes the standard concept of derivation so as to capture, e.g., provisional reasoning and sub-proof sharing. In this paper we focus on a subclass of RThs called *equational* RThs [7]. Equational RThs are characterized by sequents being equivalence classes (w.r.t. a set of equations) of terms of an order-sorted signature [10]. In the sequel, since we will only deal with equational RThs, we will omit the adjective “equational”, for brevity.

We first define a *sequent system*, as a quadruple $Ssys = \langle \Sigma, X, E, Q\rangle$. $\Sigma = \langle S, \leq, O\rangle$ is an order-sorted signature: S is a set of sort symbols, $\leq$ a partial order over S, and O an $(S^* \times S)$-indexed family of operation symbols. X is an S-indexed family of variable symbols, where X_s is countably infinite for each $s \in S$. $\mathcal{T}$ is then the S-indexed family of Σ-terms over X, $\mathcal{T}_\Sigma(X)$, defined in the usual way. E is a set of equations $t_1 = t_2$, where $t_1, t_2 \in \mathcal{T}_s$ for some $s \in S$. E induces a congruence relation $\equiv_E$ over $\mathcal{T}$, which defines the S-indexed family $\widetilde{\mathcal{T}}$ of equivalence classes of terms. Q is a subset of S, i.e. $Q \subseteq S$, which is both downward and upward closed in S (i.e., if $s \in Q$ and either $s' \leq s$ or $s \leq s'$, then also $s' \in Q$). The set of sequents defined by $Ssys$ is $Sq = \widetilde{\mathcal{T}}_Q$. In fact, Q identifies the sorts whose (equivalence classes of) terms constitute the sequents. The set I of instantiations consists of sort-preserving mappings from variables to terms. The application of an instantiation ι to a sequent sq (i.e., variable substitution) is denoted by $sq[\iota]$, and the composition of ι and ι' is denoted by $\iota \circ \iota'$.

An (inference) rule over a sequent system $Ssys$ is a pair $\langle \vec{sq}, sq\rangle \in Sq^* \times Sq$, where $\vec{sq}$ are the *premises*, and sq the *conclusion*. An RTh is a pair $Rth = \langle Ssys, R\rangle$ where $Ssys$ is a sequent system, and $R : L \to Sq^* \times Sq$ is a labeled set of rules over $Ssys$ (i.e., a function from a set L of labels to the set of all rules over $Ssys$).

Given an RTh Rth, we consider a subclass of reasoning structures, called *derivation structures* [7], that for the purposes of this paper is defined to be the set Δ of proof trees of sequents in Rth. More precisely, Δ is defined to be the smallest set satisfying the following conditions, where we also define the functions $src : \Delta \to Sq^*$ (standing for ‘source’) and $tgt : \Delta \to Sq$ (standing for ‘target’) so as to yield respectively the leaves (called *open sequents*) and the root of the derivation given as input:

- $\forall sq \in Sq : \mathit{one}(sq) \in \Delta, \mathit{src}(\mathit{one}(sq)) = [sq], \mathit{tgt}(\mathit{one}(sq)) = sq$;
- $\forall \delta_1, \ldots, \delta_n \in \Delta, \ell \in L, \iota \in I, R(\ell)[\iota] = \langle [\mathit{tgt}(\delta_1), \ldots, \mathit{tgt}(\delta_n)], sq \rangle$:
  $\delta = \mathit{dcons}([\delta_1, \ldots, \delta_n], \ell, \iota) \in \Delta$, $\mathit{src}(\delta) = \mathit{src}(\delta_1)@\cdots@\mathit{src}(\delta_n)$, $\mathit{tgt}(\delta) = sq$.

(In the above definition, *one* and *dcons* play the role of free constructors of derivations.) Finally, if $[\delta_1, \ldots, \delta_n] \in \Delta^*$ and $\delta \in \Delta$ are such that $\mathit{src}(\delta) = [\mathit{tgt}(\delta_1), \ldots, \mathit{tgt}(\delta_n)]$, then $([\delta_1, \ldots, \delta_n]; \delta)$ is a derivation obtained from δ by replacing the open sequents of δ with the corresponding derivation in $[\delta_1, \ldots, \delta_n]$. It readily follows that $\mathit{src}([\delta_1, \ldots, \delta_n]; \delta) = \mathit{src}(\delta_1)@\cdots@\mathit{src}(\delta_n)$ and $\mathit{tgt}([\delta_1, \ldots, \delta_n]; \delta) = \mathit{tgt}(\delta)$.

2.2 Annotated Reasoning Theories

The data structures manipulated by a reasoning system include both logical and control information. This is formalized by the notion of (equational) *annotated* RTh (*ARTh*) [7]. Given an RTh *Rth*, an ARTh over *Rth* is a pair $\mathit{ARth} = \langle \mathit{Rth}^{\mathrm{A}}, \epsilon \rangle$. $\mathit{Rth}^{\mathrm{A}}$ is an RTh, whose sequents encode both logical and control information, and whose rules express how these two kinds of information are manipulated together. Intuitively, $\mathit{Rth}^{\mathrm{A}}$ constitutes an "annotated" version of *Rth*, where annotations represent control information. ϵ is an *erasing* mapping from $\mathit{Rth}^{\mathrm{A}}$ to *Rth*, written $\epsilon : \mathit{Rth}^{\mathrm{A}} \to \mathit{Rth}$. Intuitively speaking, ϵ maps the (annotated) sequents and rules of $\mathit{Rth}^{\mathrm{A}}$ to their (non-annotated) counterparts of *Rth*. In other words, ϵ "erases" the control information, leaving the logical information untouched. More precisely, ϵ acts as a special case of mapping from lists of terms to lists of terms (in the Lawvere theories associated to the equational signature [16]) such that singleton lists of terms are mapped either to singleton lists of terms or to the empty sequence, whereas a non-singleton list, say $[e_1, \ldots, e_n]$, is mapped to the concatenation of the images of the singleton lists of the components, i.e. $\epsilon([e_1])@\cdots@\epsilon([e_n])$. By identifying a singleton with its element, we can say that ϵ maps a term to either a term or $[\,]$. The intuition is that the terms of $\mathit{Rth}^{\mathrm{A}}$ contain both logical and control information, while the terms of *Rth* only contain logical information: ϵ maps a term t^{A} of $\mathit{Rth}^{\mathrm{A}}$ either to a term t of *Rth* with exactly the same logical content, or to $[\,]$ (when t^{A} encodes control information only). In more detail, ϵ maps: each sort in S^{A} either to a sort in S or to $[\,]$; each variable in X^{A} either to a variable in X or to $[\,]$, consistently with the sort mapping; each term in $\mathit{Rth}^{\mathrm{A}}$ of the form $o(x_1, \ldots, x_n)$, with $n \geq 0$, either to a term in *Rth* or to $[\,]$, consistently with the sort and variable mapping. ϵ is extended homomorphically to all terms in $\mathit{Rth}^{\mathrm{A}}$. This naturally induces a mapping from Sq^{A} to Sq.

It is required that for each rule $R^{\mathrm{A}}(\ell^{\mathrm{A}}) = \langle \vec{sq}^{\mathrm{A}}, sq^{\mathrm{A}} \rangle$ in $\mathit{Rth}^{\mathrm{A}}$ there is a derivation structure δ in *Rth* such that $\mathit{src}(\delta) = \epsilon(\vec{sq}^{\mathrm{A}})$ and $\mathit{tgt}(\delta) = \epsilon(sq^{\mathrm{A}})$. This requirement ensures that the rules in $\mathit{Rth}^{\mathrm{A}}$ are "sound" w.r.t. those in *Rth*, because they "agree" on the logical content. Often, an annotated rule has a non-annotated counterpart, in the sense that the corresponding derivation structure consists of just the application of such non-annotated rule. It is also common the case where the corresponding derivation structure turns out to consist of only one sequent: this means that the annotated rule does not modify the logical content but only the control one.

2.3 Tactic Theories

While an ARTh specifies how control information is encoded via annotations, an ARTh does not specify the strategies employed by the system. In this paper we focus on an important class of proof strategies specified as tactics. Tactics, first introduced in LCF [11] and later adopted in many popular theorem provers such as NuPrl [8] and Isabelle [17], are an effective means to specify backward proof strategies in a modular fashion.

We first define an (equational) *tactic system* as a quadruple $Tsys = \langle \Sigma', E', T', F' \rangle$. $\Sigma' = \langle S', \leq', O' \rangle$ is an order-sorted signature (as in a sequent system). $\mathcal{T}'$ is then the S'-indexed family of ground Σ'-terms, $\mathcal{T}_{\Sigma'}$. E' is a set of equations $t_1 = t_2$ where $t_1, t_2 \in \mathcal{T}'_s$ for some $s \in S'$. E' induces a congruence relation $\equiv_{E'}$ over $\mathcal{T}'$, which defines the S'-indexed family $\widetilde{\mathcal{T}}'$ of equivalence classes of terms. T' and F' are subsets of S', i.e. $T' \subseteq S'$ and $F' \subseteq S'$, which are both downward and upward closed in S'. $Tac = \widetilde{\mathcal{T}}'_{T'}$ and $Fail = \widetilde{\mathcal{T}}'_{F'}$ are the set of *tactics* and *failures* (resp.) associated to $Tsys$.

Given an RTh Rth, an (equational) *tactic theory* (*TTh*) is a triple $Tth = \langle Tsys, \pi, TR \rangle$. $Tsys$ is a tactic system. The operation symbols of Σ' of arity $\langle \vec{s}, s \rangle$ with $s \in T'$ and at least one element of $\vec{s}$ also in T', are called *tacticals*. π is a (partial) injective mapping from $\bigcup_{s \in T'} \widetilde{O'}_{\langle [], s \rangle}$ (i.e., the equivalence classes of terms that are operation symbols of Σ' of arity $\langle [], s \rangle$ with $s \in T'$) to L (the rule labels of Rth). The elements of the domain of π are called *primitive tactics*. TR is a set of pairs of the form $\langle [\tau_1 \lhd sq_1 \Rightarrow r_1, \ldots, \tau_n \lhd sq_n \Rightarrow r_n], \tau_0 \lhd sq_0 \Rightarrow r_0 \rangle$ called *tactic rules* and written as

$$\frac{\tau_1 \lhd sq_1 \Rightarrow r_1 \quad \cdots \quad \tau_n \lhd sq_n \Rightarrow r_n}{\tau_0 \lhd sq_0 \Rightarrow r_0}$$

where, for $i = 0, 1, \ldots, n$, $\tau_i \in \widetilde{\mathcal{T}}'_{T'}$, sq_i is a sequent of Rth, and r_i is either a failure of Tth or a pair $\langle \delta, \iota \rangle \in \Delta \times I$.

If $\tau \in Tac$, $sq \in Sq$, and $r \in Fail \cup (\Delta \times I)$, we say that the *application* of τ to sq yields r, in symbols $\tau \lhd sq \Rightarrow^* r$, if and only if one of the following cases holds:

(i) $\exists \ell \in L, \iota \in I, R(\ell) = \langle [sq_1, \ldots, sq_n], sq' \rangle : \pi(\tau) = \ell, sq'[\iota] = sq[\iota]$,
$r = \langle dcons([one(sq_1[\iota]), \ldots, one(sq_n[\iota])], \ell, \iota), \iota \rangle$;[2]

(ii) $\exists \ell \in L, R(\ell) = \langle \vec{sq}, sq' \rangle : \pi(\tau) = \ell, (\forall \iota \in I : sq'[\iota] \neq sq[\iota]), r \in Fail$;

(iii) $\exists \langle [\tau_1 \lhd sq_1 \Rightarrow r_1, \ldots, \tau_n \lhd sq_n \Rightarrow r_n], \tau \lhd sq \Rightarrow r \rangle \in TR : \forall i = 1, \ldots, n : \tau_i \lhd sq_i \Rightarrow^* r_i$.

[2] We implicitly assume that the variables in $R(\ell)$ are standardized apart in the usual way, in order to avoid accidental "conflicts" with those in sq.

It is required that $tgt(\delta) = sq[\iota]$ whenever $\tau \lhd sq \Rightarrow^* \langle \delta, \iota \rangle$.

The intuition captured by the above definitions is that a tactic system defines a vocabulary of tactics and failures, and tactics represent non-deterministic proof strategies. If the application of a tactic to a sequent yields a derivation structure without open sequents, the tactic has succeeded in proving the sequent. If it has some open sequents, the problem of proving the sequent has been reduced to the (hopefully simpler) sub-problem of proving such open sequents. If the returned instantiation is not the identity instantiation, an instance of the original sequent is proved (or reduced to sub-problems): this mechanism can be used to support provisional reasoning, where schematic parts of sequents get instantiated during the deduction process. On the other hand, when the application of the tactic to the sequent yields a failure, this means that the tactic is unable to prove (or reduce) the sequent. By using tacticals, applications of tactics can be combined together in various ways. A TTh specifies the results of applying tactics to sequents. π identifies the primitive tactics, i.e. those that correspond to the application of a single rule of an RTh. The tactic rules in *TR* specify how applications of tactics are combined together.

As a final remark, note that the notion of TTh has been given w.r.t. an RTh. Indeed, such RTh is usually meant to be the component of an ARTh, i.e. the one with annotated sequents and rules. Therefore, the specification of the logic and control components of an OMRS is given by: an RTh *Rth* for the logic component, and an ARTh $ARth = \langle Rth^{\mathrm{A}}, \epsilon \rangle$ over *Rth* together with a TTh *Tth* over Rth^{A} for the control component.

2.4 Gluing

Specifying composition of OMRSs amounts to specifying how deductions carried out by different components relate to each other. For example, a rewriter calls a linear arithmetic decider that produces an inequality, and the inequality is used by the rewriter to rewrite a term involving an arithmetic expression. In order to specify relationships among deductions, it is required that the RThs, ARThs, and TThs of the various components share some language and that composition preserves the shared language and its meaning.

We first define composition of RThs [7]. Given sequent systems $Ssys_0$ and $Ssys_1$, we say that $Ssys_0$ is *faithfully included* into $Ssys_1$, written $Ssys_0 \hookrightarrow Ssys_1$, iff S_0, $\leq_0$, O_0, X_0, E_0, and Q_0 are included into S_1, $\leq_1$, O_1, X_1, E_1, and Q_1 respectively, and $\mathcal{T}_1 \mid S_0 = \mathcal{T}_0$. In other words, $Ssys_0 \hookrightarrow Ssys_1$ iff $Ssys_1$ "adds to" $Ssys_0$ without "changing" it. Now, given sequent systems $Ssys_1$ and $Ssys_2$, we define $shared(Ssys_1, Ssys_2)$ to be the sequent system given by the intersection of $Ssys_1$ and $Ssys_2$ (i.e., $S_0 = S_1 \cap S_2$, etc.) We say that $Ssys_1$ and $Ssys_2$ are *composable*, written $Ssys_1 \bowtie Ssys_2$, iff $shared(Ssys_1, Ssys_2) \hookrightarrow Ssys_1$ and $shared(Ssys_1, Ssys_2) \hookrightarrow Ssys_2$. If $Ssys_1 \bowtie Ssys_2$, we define their composition, $Ssys_1 + Ssys_2$, as the union of $Ssys_1$ and $Ssys_2$ (i.e., $S = S_1 \cup S_2$, etc.). It can be shown that both $\mathit{Ssys}_1 \hookrightarrow \mathit{Ssys}_1 + \mathit{Ssys}_2$ and $\mathit{Ssys}_2 \hookrightarrow \mathit{Ssys}_1 + \mathit{Ssys}_2$ (and $\mathit{shared}(\mathit{Ssys}_1, \mathit{Ssys}_2) \hookrightarrow \mathit{Ssys}_1 + \mathit{Ssys}_2$ as well). Given RThs Rth_1 and Rth_2, we say that Rth_1 and Rth_2 are *composable*, written $\mathit{Rth}_1 \bowtie \mathit{Rth}_2$, iff $\mathit{Ssys}_1 \bowtie \mathit{Ssys}_2$ (i.e., the underlying sequent systems are composable) and $L_1 \cap L_2 = \emptyset$ (i.e., the rule labels are disjoint). If $\mathit{Rth}_1 \bowtie \mathit{Rth}_2$, we define the composition $\mathit{Rth}_1 + \mathit{Rth}_2$ to be the RTh $\langle \mathit{Ssys}_1 + \mathit{Ssys}_2, R_1 \cup R_2 \rangle$.

Composition of ARThs is defined in terms of composition of the underlying RThs in a natural way [7]. Given ARThs $\mathit{ARth}_1 = \langle \mathit{Rth}_1^{\mathrm{A}}, \epsilon_1 \rangle$ over Rth_1, and $\mathit{ARth}_2 = \langle \mathit{Rth}_2^{\mathrm{A}}, \epsilon_2 \rangle$ over Rth_2, we say that ARth_1 and ARth_2 are *composable*, written $\mathit{ARth}_1 \bowtie \mathit{ARth}_2$, iff $\mathit{Rth}_1 \bowtie \mathit{Rth}_2$, $\mathit{Rth}_1^{\mathrm{A}} \bowtie \mathit{Rth}_2^{\mathrm{A}}$, and ϵ_1 and ϵ_2 agree on $\mathit{shared}(\mathit{Rth}_1^{\mathrm{A}}, \mathit{Rth}_2^{\mathrm{A}})$. If $\mathit{ARth}_1 \bowtie \mathit{ARth}_2$, we define $\mathit{ARth}_1 + \mathit{ARth}_2 = \langle \mathit{Rth}_1^{\mathrm{A}} + \mathit{Rth}_2^{\mathrm{A}}, \epsilon_1 \cup \epsilon_2 \rangle$, which is an ARTh over $\mathit{Rth}_1 + \mathit{Rth}_2$.

Finally, composition of TThs is defined as follows. Given tactic systems Tsys_0 and Tsys_1, we say that Tsys_0 is *faithfully included* into Tsys_1, written $\mathit{Tsys}_0 \hookrightarrow \mathit{Tsys}_1$, iff Tsys_0 is included into Tsys_1, and in addition $\mathcal{T}_1' \mid S_0 = \mathcal{T}_0'$ (analogous to sequent systems). Given tactic systems Tsys_1 and Tsys_2, we define $\mathit{shared}(\mathit{Tsys}_1, \mathit{Tsys}_2)$ to be the intersection of Tsys_1 and Tsys_2 (analogous to sequent systems). We say that Tsys_1 and Tsys_2 are *composable*, written $\mathit{Tsys}_1 \bowtie \mathit{Tsys}_2$, iff $\mathit{shared}(\mathit{Tsys}_1, \mathit{Tsys}_2) \hookrightarrow \mathit{Tsys}_1$ and $\mathit{shared}(\mathit{Tsys}_1, \mathit{Tsys}_2) \hookrightarrow \mathit{Tsys}_2$. If $\mathit{Tsys}_1 \bowtie \mathit{Tsys}_2$, we define their composition, $\mathit{Tsys}_1 + \mathit{Tsys}_2$, as the union of Tsys_1 and Tsys_2 (again, analogous to sequent systems). It can be shown that both $\mathit{Tsys}_1 \hookrightarrow \mathit{Tsys}_1 + \mathit{Tsys}_2$ and $\mathit{Tsys}_2 \hookrightarrow \mathit{Tsys}_1 + \mathit{Tsys}_2$. Given TThs Tth_1 and Tth_2, we say that Tth_1 and Tth_2 are *composable*, written $\mathit{Tth}_1 \bowtie \mathit{Tth}_2$, iff $\mathit{Tsys}_1 \bowtie \mathit{Tsys}_2$ (i.e., the underlying tactic systems are composable) and π_1 and π_2 have disjoint domains. If $\mathit{Tth}_1 \bowtie \mathit{Tth}_2$, we define the composition $\mathit{Tth}_1 + \mathit{Tth}_2 = \langle \mathit{Tsys}_1 + \mathit{Tsys}_2, \pi_1 \cup \pi_2, \mathit{TR}_1 \cup \mathit{TR}_2 \rangle$.

It is easy to show that the composition operators + over RThs, ARThs, and TThs are commutative, associative, and idempotent.

3 NQTHM as a Case Study

The following description of the top-level inference of NQTHM is derived from [4] and the analysis of the actual LISP code of the system. Figure 1 depicts the flows of terms, clauses and clause sequences among the data structures and inference processes.

The user asks the system to prove a conjecture by supplying an NQTHM term. (In NQTHM a term t, used where a formula is expected, stands for the formula $(t \neq \mathtt{F})$ [4, p. 31].) This term is first *pre-processed*: some of the available definitions are unfolded (*pre-expansion*) and then the resulting term is turned into clausal form (*clausification*), as shown in the upper part of Fig. 1 (for now, ignore the leftmost arrow from the conjecture to the clausification). NQTHM 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

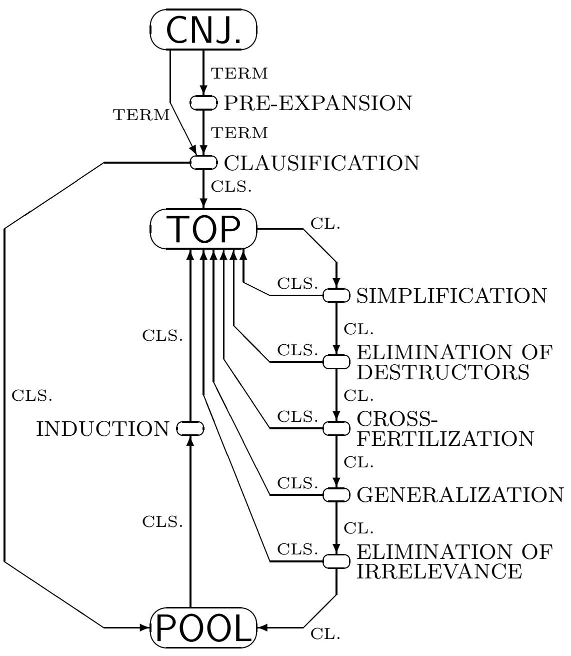

Fig. 1. The top-level inference of NQTHM.

clause in turn, until no clauses are left. Each process returns zero or more (supposedly simpler) 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 output clauses are produced by these processes by means of different inference techniques, e.g. the simplifier performs rewriting and typeset reasoning [4, Chapt. VI]; since here we only describe the top-level inference of NQTHM, we do not go into the details of these techniques.

The clauses manipulated by NQTHM are stored in two data structures: the Top and the Pool (shown in Fig. 1). The *Top* is a finite sequence of pairs whose first element is a clause and second element is the history of the clause. A *history* of a clause *cl* is a finite sequence of triples $[\langle hid_1, cl_1, hel_1\rangle, \ldots, \langle]hid_n, cl_n, hel_n\rangle$, where each triple is called *history component*, each $hid_i \in \{\mathsf{SI}, \mathsf{ED}, \mathsf{CF}, \mathsf{GE}, \mathsf{EI}\}$ and is called *history identifier* (it identifies one of the first five main inference processes) each cl_i is a clause, and each hel_i is a *history element* (since they are not relevant to the top-level inference we do not specify them here): such a history expresses that clause cl_{i+1} has been obtained from cl_i by applying the process identified by hid_i, where $cl = cl_{n+1}$; histories are updated by NQTHM as the proof progresses. The *Pool* is a finite sequence 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 sequence of resulting clauses, each one equipped with the empty history (for now, ignore the leftmost arrow from the clausification to the Pool in Fig. 1). The Pool is instead set to the empty sequence.

While the Top is not empty, its first clause and history are picked up, and fed into the simplifier, which returns a sequence of clauses and a history element. If the returned sequence of clauses is equal to the singleton sequence having as unique element the input clause then the simplifier has failed. Otherwise the returned clauses are added to the head of the Top, each paired with the history obtained by extending the input history with the history component consisting of history identifier SI, the input clause, and the history element 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 (unless it is the empty clause, 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 NQTHM terminates with success, because the conjecture has been proved; otherwise, the following actions are performed. First, while the clause at the head of the Pool is tagged by BP, the clause and the tag are removed (because the clause has been proved). When the clause at the head is tagged by TBP, another clause in the Pool is searched, which subsumes it. If one is found, then in case it is tagged by BP NQTHM stops with a failure (because a loop in inventing inductions is likely to be taking place), in case it is tagged by TBP the clause and tag at the head are just removed and things proceed as above, when the Top gets empty. Eliminating a subsumed clause is called *cleaning* the Pool. If no subsuming clause is found, the tag is changed to BP and the clause is fed into the induction process, which either cannot invent any induction, in which case NQTHM stops with a failure, or produces a sequence of clauses which are added to the head of the Top, each paired with the empty history (see the central-lower part of Fig. 1; for now, ignore the fact that really clause sequences, and not just clauses, go into the induction process); in this second case, then the clause and history at the head of the Top are fed into the simplifier, and things go as explained above.

For ease of understanding, in the above description we have omitted two important features, which now we explain in detail, thus completing our description.

Just before pushing a clause onto the Pool, if either (1) the Pool is empty and the history of the clause contains at least one history identifier other than SI, or (2) the Pool is not empty and no BP is present in it, then a backtracking is performed: Top and Pool are emptied, the conjecture is clausified (without pre-expanding it), and the resulting sequence of clauses is pushed onto the Pool as a whole, tagged by TBP (so that induction is performed upon the original conjecture, see footnote at p. 90 of [4]), as shown in the leftmost part of

Fig. 1. In fact, the Pool really contains clause sequences (and not just clauses, as we approximated above) paired with tags, and the induction process really receives clause sequences as inputs. If none of the two backtracking conditions is satisfied, then the singleton sequence of the clause, paired with TBP, is pushed onto the Pool.

The second feature is motivated as follows: after inventing an induction scheme, it is heuristically convenient both 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 that purpose, the induction process also produces two sets of terms as outputs (those appearing in the hypothesis and 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 the induction: to achieve that, as soon as the simplifier fails upon a clause, its history is extended with a history component consisting of a (sixth) *settle-down* history identifier SD (indicating that the clause has settled down with respect to the special treatment of terms appearing in induction hypothesis and conclusion), the clause itself, and the null history element `nil`, and the clause and new history are fed into the simplifier; however, this does not happen if SD is already present in the history, and in that case the clause and history are just fed into the elimination of destructors process. The simplifier ignores the two input sets of terms iff SD is present in the input history.

4 Specifying NQTHM as an OMRS

We now provide a modular specification of the top-level logic and control of NQTHM by composing specifications for the main modules of the system, i.e.:

- user (U);
- waterfall (W);
- simplification (SI);
- elimination of destructors (ED);
- cross-fertilization (CF);
- generalization (GE);
- elimination of irrelevance (EI);
- induction (IN);
- pre-expansion (PE);
- clausification (CL);
- subsumption (SB).

The user module is responsible of pre-processing the user-supplied conjecture. The waterfall module manages the distribution of the goal clauses between the Top and the Pool, including the calls to the inference processes [3]. The purposes of the other modules can be readily inferred from the description given in Sect. 3. By composing RThs, ARThs, and TThs for the individual modules,

[3] The name derives from a metaphorical analogy with a waterfall described in [4]

we obtain an RTh, an ARTh, and a TTh for the whole system. For the lack of space we confine ourselves to specifying what is relevant to the top-level inference of NQTHM. In particular, we do not consider further modularizations (e.g., simplification can be partitioned into rewriting, linear arithmetic, type reasoning, etc.), even though the approach can be readily applied to the specification of sub-modules. To simplify notation we introduce the following set of labels $\mathcal{N} = \{\mathrm{U}, \mathrm{W}, \mathrm{SI}, \mathrm{ED}, \mathrm{CF}, \mathrm{GE}, \mathrm{EI}, \mathrm{IN}, \mathrm{PE}, \mathrm{CL}, \mathrm{SB}\}$.

4.1 A Reasoning Theory for NQTHM

The RThs of the component modules are Rth_i for $i \in \mathcal{N}$ which are described below. All these RThs are composable together, and their composition, $\sum_{i \in \mathcal{N}} Rth_i$, is an RTh for the whole system. Rth_{U} contains sequents of the form $(th \vdash t)$, where th represents the current NQTHM *theory* (i.e. an unordered collection of type and function definitions, axioms, and previously proved conjectures), and t represents an NQTHM term[4]. Each such sequent asserts that t is a logical consequence of the axioms in th. Rth_{U} also contains sequents of the form $(th \vdash \widetilde{cl})$, asserting that the axioms in th entail the clauses in $\widetilde{cl}$ (a clause is a set of NQTHM terms, considered disjunctively). In addition, Rth_{U} contains sequents of the form $(th \vdash t = t')$, asserting that the term t' obtained by unfolding some definitions in t is equal to the original term t, and sequents of the form $(t \leftrightarrow \widetilde{cl})$, asserting the logical equivalence between a term t and its CNF translation $\widetilde{cl}$. The rules of Rth_{U} are:

$$\frac{th \vdash t = t' \quad th \vdash t'}{th \vdash t}\ \mathsf{CallPreExp} \qquad \frac{t \leftrightarrow \widetilde{cl} \quad th \vdash \widetilde{cl}}{th \vdash t}\ \mathsf{CallClaus}$$

(The backward application of) CallPreExp formalizes the activity of invoking the pre-expander. It specifies that the provability of a term follows from the provability of another term obtained by unfolding some definitions. Similarly CallClaus formalizes the activity of invoking the clausifier.

Rth_{W} has sequents of the form $(th \vdash \widetilde{cl})$, $(th \vdash \widetilde{cl} \to \widetilde{cl}')$, and $(\widetilde{cl} \prec \widetilde{cl}')$. They respectively assert that the clauses in $\widetilde{cl}$ follow from the axioms of th, that under the axioms in th the clauses in $\widetilde{cl}$ imply the clauses in $\widetilde{cl}'$, and that $\widetilde{cl}$ subsumes $\widetilde{cl}'$. Note that the sequents $(th \vdash \widetilde{cl})$ are shared with Rth_{U}. The rules of Rth_{W} are the following:

$$\frac{\widetilde{cl}' \prec \widetilde{cl}'' \quad th \vdash \widetilde{cl} \wedge \widetilde{cl}'}{th \vdash \widetilde{cl} \wedge \widetilde{cl}' \wedge \widetilde{cl}''}\ \mathsf{ElimSubsum} \quad \frac{th \vdash \widetilde{cl}' \to cl \quad th \vdash \widetilde{cl} \wedge \widetilde{cl}'}{th \vdash \widetilde{cl} \wedge cl}\ \mathsf{CallInfProc}$$

The rule ElimSubsum states that subsumed clauses can be eliminated. The rule CallInfProc formalizes the activity of invoking the inference processes[5].

[4] More precisely, sequents are equivalence classes of terms in $\mathcal{T}_{\mathrm{U}}$ of the form $(th \vdash t)$, where th and t are also terms in $\mathcal{T}_{\mathrm{U}}$ that describe theories and NQTHM terms, respectively. However, for the sake of brevity, we neglect the distinction.

[5] The sort for clauses is a subsort of the one for clause conjunctions, so it is legitimate to write $\widetilde{cl} \wedge cl$.

Logically, they specify that the provability of a conjunction of clauses is implied by the provability of the conjunction of clauses obtained by replacing a clause with a conjunction that entails it (e.g., the result of simplifying the clause).

$\mathit{Rth}_{\mathrm{PE}}$ and $\mathit{Rth}_{\mathrm{CL}}$ include sequents of the form $(\mathit{th} \vdash t = t)$ and $(t \leftrightarrow \widetilde{cl})$, respectively, which are shared with $\mathit{Rth}_{\mathrm{U}}$. $\mathit{Rth}_{\mathrm{SI}}$, $\mathit{Rth}_{\mathrm{ED}}$, $\mathit{Rth}_{\mathrm{CF}}$, $\mathit{Rth}_{\mathrm{GE}}$, $\mathit{Rth}_{\mathrm{EI}}$, and $\mathit{Rth}_{\mathrm{IN}}$ include sequents of the form $(\mathit{th} \vdash \widetilde{cl} \rightarrow \widetilde{cl}')$, shared with $\mathit{Rth}_{\mathrm{W}}$. $\mathit{Rth}_{\mathrm{SB}}$ includes sequents of the form $(\widetilde{cl} \prec \widetilde{cl}')$, shared with $\mathit{Rth}_{\mathrm{W}}$. These RThs contain other kind of sequents, and contain rules to perform deductions upon all their sequents. Since we are only interested in modeling the top-level inference, we do not describe them here.

4.2 An Annotated Reasoning Theory for NQTHM

There is an ARTh for each RTh introduced in Sect. 4.1, namely $\mathit{ARth}_i = \langle \mathit{Rth}_i^{\mathrm{A}}, \epsilon_i \rangle$ for $i \in \mathcal{N}$. These ARThs are all composable together, and their composition is an ARTh, $\sum_{i \in \mathcal{N}} \mathit{Rth}_i^{\mathrm{A}}$, for the whole system. $\mathit{Rth}_{\mathrm{U}}^{\mathrm{A}}$ has sequents of the forms $(\mathit{evh} \vdash t)$, $(\mathit{evh} \vdash t = t')$, $(t \leftrightarrow \overline{ocl})$, and $(\mathit{evh} \vdash \mathit{top}, \mathit{pool}; \widetilde{\mathit{iht}}, \widetilde{\mathit{ict}})$, which constitute the annotated counterparts of the logical sequents $(\mathit{th} \vdash t)$, $(\mathit{th} \vdash t = t')$, $(t \leftrightarrow \widetilde{cl})$, and $(\mathit{th} \vdash \widetilde{cl})$ of $\mathit{Rth}_{\mathrm{U}}$. *evh* is the current NQTHM *event history*, i.e. a sequence of type and function definitions, axioms and proved conjectures, all annotated with heuristic information. $\epsilon_{\mathrm{U}}(\mathit{evh})$ yields the underlying NQTHM theory, where all the heuristic information (as well as the relative ordering of events) has been removed. $\overline{ocl}$ is a finite sequence of ordered clauses considered conjunctively, and an ordered clause is a finite sequence of NQTHM terms considered disjunctively[6]. $\epsilon_{\mathrm{U}}(\overline{ocl})$ yields a set of sets of NQTHM terms (an unordered conjunction of unordered clauses), i.e. removes the ordering information. *top* represents the current Top: it is a finite sequence of expressions of the form $(\mathit{ocl} : \mathit{hst})$, where *hst* is a history. *pool* represents the current Pool: it is a finite sequence of expressions of the form $(\overline{ocl} : \mathit{tag})$, where *tag* is either BP or TBP. ϵ_{U} maps *top* to the conjunction of its underlying clauses, and maps *pool* to the conjunction of its underlying clauses tagged by TBP. $\widetilde{\mathit{iht}}$ and $\widetilde{\mathit{ict}}$ are finite sets of terms, respectively those occurring in the induction hypothesis and conclusion (see Sect. 3). They are erased by ϵ_{U}, as they only constitute control information. The rules of $\mathit{Rth}_{\mathrm{U}}^{\mathrm{A}}$ are the following:

$$\frac{\mathit{evh} \vdash t = t' \quad \mathit{evh} \vdash t'}{\mathit{evh} \vdash t}\ \mathsf{CallPreExp}^{\mathrm{A}} \qquad \frac{t \leftrightarrow \overline{ocl} \quad \mathit{evh} \vdash \mathit{sethst}(\overline{ocl}, [\,]), [\,]; \emptyset, \emptyset}{\mathit{evh} \vdash t}\ \mathsf{CallClaus}^{\mathrm{A}}$$

where $\mathit{sethst}(\overline{ocl}, \mathit{hst})$ denotes the sequence of clauses in $\overline{ocl}$ each paired with history *hst* (the empty history in rule $\mathsf{CallClaus}^{\mathrm{A}}$ above). Note that $\mathsf{CallPreExp}^{\mathrm{A}}$ and $\mathsf{CallClaus}^{\mathrm{A}}$ are annotated counterparts of $\mathsf{CallPreExp}$ and $\mathsf{CallClaus}$ respectively.

[6] In the sequel we neglect the distinction between clauses and ordered clauses. The context will make clear what is intended.

$Rth^{\mathrm{A}}_{\mathrm{W}}$ contains sequents $(evh \vdash top, pool; \widetilde{iht}, \widetilde{ict})$ (shared with $Rth^{\mathrm{A}}_{\mathrm{U}}$) and $(pool \prec \overline{ocl})$ as annotated counterparts of $(th \vdash \widetilde{cl})$ and $(\widetilde{cl} \prec \widetilde{cl}')$, respectively. A sequent $(pool \prec \overline{ocl})$ asserts that the clauses in $\overline{ocl}$ are subsumed by clauses in $pool$ that are tagged by TBP (and not BP). Sequents $(th \vdash \widetilde{cl} \to \widetilde{cl}')$ in Rth_{W} indeed have three annotated counterparts, namely $(evh \vdash ocl, hst; \widetilde{iht}, \widetilde{ict} \rightsquigarrow \overline{ocl}, hel)$, $(evh \vdash ocl, hst \rightsquigarrow \overline{ocl}, hel)$, and $(evh \vdash \overline{ocl} \rightsquigarrow \overline{ocl}'; \widetilde{iht}, \widetilde{ict})$ (where hel is a history element). For all three, ϵ_{W} collects the clauses at the left and right of $\rightsquigarrow$ and puts them respectively to the right and left of $\to$ (e.g., $\epsilon_{\mathrm{W}}(evh \vdash \overline{ocl} \rightsquigarrow \overline{ocl}', \widetilde{iht}, \widetilde{ict}) = \epsilon_{\mathrm{W}}(evh) \vdash \epsilon_{\mathrm{W}}(\overline{ocl}') \to \epsilon_{\mathrm{W}}(\overline{ocl}))$. In addition, $Rth^{\mathrm{A}}_{\mathrm{W}}$ also contains sequents $NoSomeHstId(hst, hid)$, $AllHstId(hst, hid)$, and $SomeTag(pool, tag)$. They respectively assert that the history hst does not contain the history identifier hid, that all history elements of hst contain hid as history identifier, and that there is at least one tag tag in $pool$. Such sequents only consist of control information, in fact they are erased by ϵ_{W}. Rule $\mathsf{CallInfProc}$ has five annotated counterparts, one for each inference process, namely $\mathsf{CallSimp}^{\mathrm{A}}$, $\mathsf{CallElDes}^{\mathrm{A}}$, $\mathsf{CallCrFer}^{\mathrm{A}}$, $\mathsf{CallGen}^{\mathrm{A}}$, and $\mathsf{CallElIrr}^{\mathrm{A}}$. To illustrate here is the rule $\mathsf{CallElDes}^{\mathrm{A}}$:

$$\frac{evh \vdash ocl, hst \rightsquigarrow \overline{ocl}, hel \qquad evh \vdash sethst(\overline{ocl}, [\langle \mathsf{ED}, ocl, hel\rangle])@top, pool; \widetilde{iht}, \widetilde{ict}}{evh \vdash [(ocl\!:\!hst) \mid top], pool; \widetilde{iht}, \widetilde{ict}}\ \mathsf{CallElDes}^{\mathrm{A}}$$

The following rule formalizes the activity of marking the first clause of the Top as "settled down" [7]:

$$\frac{evh \vdash [(ocl\!:\![\langle \mathsf{SD}, ocl, \mathtt{nil}\rangle \mid hst]) \mid top], pool; \widetilde{iht}, \widetilde{ict} \qquad NoSomeHstId(hst, \mathsf{SD})}{evh \vdash [(ocl\!:\!hst) \mid top], pool; \widetilde{iht}, \widetilde{ict}}\ \mathsf{SettleDown}^{\mathrm{A}}$$

The derivation structure corresponding to $\mathsf{SettleDown}^{\mathrm{A}}$ consists of one sequent, i.e. the result of applying ϵ_{W} to either the first premise or the conclusion of $\mathsf{SettleDown}^{\mathrm{A}}$ (they yield the same result). In fact, this rule only modifies control information. The backward application of $\mathsf{Push}^{\mathrm{A}}_1$ and $\mathsf{Push}^{\mathrm{A}}_2$ models the act of pushing a clause into the Pool:

$$\frac{evh \vdash top, [(([t \mid ocl])\!:\!\mathsf{TBP})]; \widetilde{iht}, \widetilde{ict} \qquad AllHstId(hst, \mathsf{SI})}{evh \vdash [([t \mid ocl]\!:\!hst) \mid top], [\,]; \widetilde{iht}, \widetilde{ict}}\ \mathsf{Push}^{\mathrm{A}}_1$$

$$\frac{evh \vdash top, [(([t \mid ocl])\!:\!\mathsf{TBP}) \mid pool]; \widetilde{iht}, \widetilde{ict} \qquad SomeTag(pool, \mathsf{BP})}{evh \vdash [([t \mid ocl]\!:\!hst) \mid top], pool; \widetilde{iht}, \widetilde{ict}}\ \mathsf{Push}^{\mathrm{A}}_2$$

Either $\mathsf{Push}^{\mathrm{A}}_1$ or $\mathsf{Push}^{\mathrm{A}}_2$ is applicable exactly when no backtracking needs to be carried out — cf. end of Sect. 3. Finally, $ARth_{\mathrm{W}}$ contains rules over sequents of the form $NoSomeHstId(hst, hid)$, $AllHstId(hst, hid)$, and $SomeTag(pool, tag)$ which are not given here for the lack of space.

$ARth_{\mathrm{PE}}$ and $ARth_{\mathrm{CL}}$ include sequents of the form $(evh \vdash t = t')$ and $(t \leftrightarrow \overline{ocl})$ respectively, both shared with $ARth_{\mathrm{U}}$. $ARth_{\mathrm{SI}}$, $ARth_{\mathrm{IN}}$, and $ARth_{\mathrm{SB}}$ include sequents of the form $(evh \vdash ocl, hst; \widetilde{iht}, \widetilde{ict} \rightsquigarrow \overline{ocl}, hel)$,

[7] `nil` indicates a history element that substantially contains no information; in fact, such history element is never used by NQTHM.

$(\mathit{evh} \vdash \overline{\mathit{ocl}} \rightsquigarrow \overline{\mathit{ocl}}'; \widetilde{\mathit{iht}}, \widetilde{\mathit{ict}})$, and $(\mathit{pool} \prec \overline{\mathit{ocl}})$ respectively, all shared with $\mathit{ARth}_{\mathrm{W}}$. $\mathit{ARth}_{\mathrm{ED}}$, $\mathit{ARth}_{\mathrm{CF}}$, $\mathit{ARth}_{\mathrm{GE}}$, and $\mathit{ARth}_{\mathrm{EI}}$ all include sequents of the form $(\mathit{evh} \vdash \mathit{ocl}, \mathit{hst} \rightsquigarrow \overline{\mathit{ocl}}, \mathit{hel})$, shared with $\mathit{ARth}_{\mathrm{W}}$. These ARThs also contain other sequents, as well as rules over them, but we do not present them here as they are not part of the top-level inference.

4.3 A Tactic Theory for NQTHM

There is a TTh for each ARTh introduced in Sect. 4.2, namely $\mathit{Tth}_i = \langle \mathit{Tsys}_i, \pi_i, \mathit{TR}_i \rangle$ for $i \in \mathcal{N}$). These TThs are all composable together, and their composition is a TTh, $\sum_{i \in \mathcal{N}} \mathit{Tsys}_i$, for the whole system. Each TTh has a primitive tactic corresponding (via the π mapping) to an inference rule of the associated ARTh. For instance `CallPreExp`, `CallClaus`, and `PushOrig` are among the primitive tactics of $\mathit{Tsys}_{\mathrm{U}}$ and $\pi_{\mathrm{U}}(\texttt{CallPreExp}) = \mathsf{CallPreExp}^{\mathrm{A}}$, $\pi_{\mathrm{U}}(\texttt{CallClaus}) = \mathsf{CallClaus}^{\mathrm{A}}$, and $\pi_{\mathrm{U}}(\texttt{PushOrig}) = \mathsf{PushOrig}^{\mathrm{A}}$. In addition, each TTh has three tacticals `ORELSE`, `THEN`, and `REPEAT`, respectively of arities $\langle [s_{\mathrm{F}}, s_{\mathrm{T}}, s_{\mathrm{T}}], s_{\mathrm{T}} \rangle$, $\langle [s_{\mathrm{T}}, s_{\mathrm{T}}, s_{\mathrm{T}}], s_{\mathrm{T}} \rangle$, and $\langle [\vec{s}_{\mathrm{F}}, s_{\mathrm{T}}], s_{\mathrm{T}} \rangle$, where s_{F} is a failure sort, s_{T} is a tactic sort, and $\vec{s}_{\mathrm{F}}$ is a sort for lists of failures. For notational convenience we write $(\tau_1\ \texttt{ORELSE}(\mathit{fail})\ \tau_2)$, $(\tau\ \texttt{THEN}\ [\tau_1, \tau_2])$, and $(\texttt{REPEAT}(\overline{\mathit{fail}})\ \tau)$ in place of $\texttt{ORELSE}(\mathit{fail}, \tau_1, \tau_2)$, $\texttt{THENL}(\tau, \tau_1, \tau_2)$, and $\texttt{REPEAT}(\overline{\mathit{fail}}, \tau)$, respectively. We also assume that `ORELSE` associates to the right. Furthermore, each TTh has a tactic `idtac`. `ORELSE` is used to try a tactic and, if it fails with a specific failure, to try an alternative tactic. `THEN` is used to apply tactics in sequence. `REPEAT` is used to exhaustively apply a tactic in sequence until specific failures are returned. `idtac` is the "identity" tactic. To illustrate, here are the tactic rules for `ORELSE`:

$$\frac{\tau_1 \lhd \mathit{sq} \Rightarrow r}{(\tau_1\ \texttt{ORELSE}(\mathit{fail})\ \tau_2) \lhd \mathit{sq} \Rightarrow r}\ \text{if } r \neq \mathit{fail} \qquad \frac{\tau_1 \lhd \mathit{sq} \Rightarrow \mathit{fail} \quad \tau_2 \lhd \mathit{sq} \Rightarrow r}{(\tau_1\ \texttt{ORELSE}(\mathit{fail})\ \tau_2) \lhd \mathit{sq} \Rightarrow r}$$

$\mathit{Tth}_{\mathrm{U}}$ includes tactics `Nqthm`, `PreExp`, `Claus`, and `Waterfall`. They are related via the following tactic definition:

```
Nqthm = (CallPreExp THENL [PreExp,
                           CallClaus THENL [Claus, Waterfall]])
        ORELSE(bktfail) (PushOrig THENL [Claus, Waterfall])
```

`PreExp` and `Claus` also belong to $\mathit{Tth}_{\mathrm{PE}}$ and $\mathit{Tth}_{\mathrm{CL}}$, respectively, and their application models the actions of the pre-expansion and clausification module, respectively. `Waterfall` also belongs to $\mathit{Tth}_{\mathrm{W}}$, which includes `FiveProcesses`, as well as `Simp`, `ElDes`, `CrFer`, `Gen`, `ElIrr`, and `Ind`. The application of the last six tactics models the actions of the six main inference processes of NQTHM. These tactics of $\mathit{Tth}_{\mathrm{W}}$ are related by the following two tactic definitions:

```
Waterfall =
REPEAT([totfail, bktfail],
  (Qed
   ORELSE(fail) (FiveProcesses
                 ORELSE(fail)
                (Push1 THENL [idtac, AllHstId])
                ORELSE(fail)
               (Push2 THENL [idtac, SomeTag])))
   ORELSE(fail) ElimProved
   ORELSE(fail) (ElimSubsum THENL [idtac, Subsum])
   ORELSE(fail) (CallInd THENL [Ind, idtac]))

FiveProcesses =
    (CallSimp THENL [Simp, idtac])
    ORELSE(fail) (SettleDown THENL [idtac, NoSomeHstId])
    ORELSE(fail) (CallElDes  THENL [ElDes, idtac])
    ORELSE(fail) (CallCrFer  THENL [CrFer, idtac])
    ORELSE(fail) (CallGen    THENL [Gen,   idtac])
    ORELSE(fail) (CallElIrr  THENL [ElIrr, idtac])
```

Such equations contain tactics `AllHstId`, `NoSomeHstId`, and `SomeTag`, which are applied to sequents of the form $AllHstId(hst, hid)$, $SomeTag(pool, tag)$, and $NoSomeHstId(hst, hid)$. The application of the first two, in case of failure, yields `bktfail`, while the application of the third one yields `fail` in case of failure. Finally, `Subsum`, also in Tth_{W}, yields `fail` if no subsuming clause is found, `totfail` if a subsuming clause is found but it is tagged by BP. The other TThs contain some more tactics which are not described here for the lack of space.

The tactic `Nqthm` is meant to be invoked upon a sequent of the form $(evh \vdash t)$, which contains the current NQTHM event history and the user-supplied conjecture to be proved. A simple analysis shows that `Waterfall` can only fail with `bktfail` or `totfail`. In the first case, the `ORELSE` in the definition of `Nqthm` activates the backtracking tactic (i.e. `(PushOrig THEN [Claus, Waterfall]))`. If instead `Waterfall` fails with `totfail`, then `Nqthm` immediately fails with `totfail`. This reflects the actual behavior of the system (cf. Sect. 3). It is worth pointing out that the semantics of the `ORELSE` and `REPEAT` tacticals differs from the traditional one (as given, for instance, in [11,8]) in that they are sensitive to the identity of the failures generated by the component tactics. This is necessary in our case study as the outermost `ORELSE` must activate the backtracking tactic only in the event the first tactic fails with `bktfail`, and immediately fail if `totfail` is returned.

5 Conclusions

In this paper we have proposed an approach to specifying the control component of mechanized reasoning systems. The proposed approach amounts to annotating the data structures representing the logic with control information, and specifying proof strategies via tactics. We have showed the adequacy of the approach by discussing the specification of the top-level inference strategy of NQTHM as a set of cooperating specialized reasoning modules.

References

[1] Clemens Ballarin, Karsten Homann, and Jacques Calmet. Theorems and Algorithms: An Interface between Isabelle and Maple. In *A.H.M. Levelt, editor, Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'95)*, pages 150–157, 1995.

[2] P. G. Bertoli, J. Calmet, F. Giunchiglia, and K. Homann. Specification and integration of theorem provers and computer algebra systems. In Jaques Calmet and Jan Plaza, editors, *Proceedings of the International Conference on Artificial Intelligence and Symbolic Computation (AISC-98)*, volume 1476 of *LNAI*, pages 94–106, Berlin, September 16–18 1998. Springer.

[3] P.G. Bertoli. *Using OMRS in practice: a case study with* ACL2. PhD thesis, Computer Science Dept., University Rome 3, Rome, 1997.

[4] R.S. Boyer and J.S. Moore. *A Computational Logic.* Academic Press, 1979. ACM monograph series.

[5] Bruno Buchberger, Tudor Jebelean, Franz Kriftner, Mircea Marin, Elena Tomuţa, and Daniela Vāsaru. A survey of the theorema project. In Wolfgang W. Küchlin, editor, *ISSAC '97. Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, July 21–23, 1997, Maui, Hawaii*, pages 384–391, New York, NY 10036, USA, 1997. ACM Press.

[6] Edmund Clarke and Xudong Zhao. Analytica — A Theorem Prover in Mathematica. In *Proc. of the 11th Conference on Automated Deduction*, pages 761–765, 1992.

[7] Alessandro Coglio, Fausto Giunchiglia, José Meseguer, and Carolyn Talcott. Composing and Controlling Search in Reasoning Theories using Mappings. Technical report, DIST, University of Genoa, Italy, May 1998.

[8] R.L. Constable, S.F. Allen, H.M. Bromley, et al. *Implementing Mathematics with the NuPRL Proof Development System.* Prentice Hall, 1986.

[9] F. Giunchiglia, P. Pecchiari, and C. Talcott. Reasoning Theories: Towards an Architecture for Open Mechanized Reasoning Systems. Technical Report 9409-15, IRST, Trento, Italy, 1994. Also published as Stanford Computer Science Department Technical note number STAN-CS-TN-94-15, Stanford University. Short version published in Proc. of the First International Workshop on Frontiers of Combining Systems (FroCoS'96), Munich, Germany, March 1996.

[10] 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.

[11] M.J. Gordon, A.J. Milner, and C.P. Wadsworth. *Edinburgh LCF - A mechanized logic of computation*, volume 78 of *Lecture Notes in Computer Science.* Springer Verlag, 1979.

[12] John Harrison. *Theorem Proving with the Real Numbers.* Springer Verlag, 1998.

[13] John Harrison and Laurent Théry. A Skeptic's Approach to Combining Hol and Maple. *Journal of Automated Reasoning*, 21:279–294, 1998.

[14] Paul Jackson. Exploring Abstract Algebra in Constructive Type Theory. In *Proc. of the 12th Conference on Automated Deduction*, pages 590–604, 1994.

[15] M. Kaufmann and J S. Moore. acl2 Version 1.8 User's Manual. Available on line at `http://www.cs.utexas.edu/users/moore/acl2/index.html`.

[16] F. William Lawvere. Functorial semantics of algebraic theories. *Proceedings, National Academy of Sciences*, 50:869–873, 1963. Summary of Ph.D. Thesis, Columbia University.

[17] L. Paulson. The Foundation of a Generic Theorem Prover. *Journal of Automated Reasoning*, 5:363–396, 1989.