J. Symbolic Computation (2001) **32**, 305–332
doi:10.1006/jsco.2000.0464
Available online at http://www.idealibrary.com on IDEAL®

The Control Layer in Open Mechanized Reasoning Systems: Annotations and Tactics

ALESSANDRO ARMANDO[†], ALESSANDRO COGLIO[‡],
FAUSTO GIUNCHIGLIA[§¶] AND SILVIO RANISE[†‖]

[†] *DIST, University of Genova, 16145 Genova, Italy*
[‡] *Kestrel Institute, Palo Alto, CA 94304, U.S.A.*
[§] *DISA, University of Trento, 38100 Trento, Italy*
[¶] *IRST (Inst. for Scient. and Techn. Research), 38050 Trento, Italy*
[‖] *LORIA – Université Henri Poincaré, 54506 Nancy, France*

We are interested in developing a methodology for integrating mechanized reasoning systems such as Theorem Provers, Computer Algebra Systems, and Model Checkers. Our approach is to provide a framework for specifying mechanized reasoning systems and to use specifications as a starting point for integration. We build on the work presented by Giunchiglia *et al.* (1994) 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. We 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 constraint contextual rewriting as a set of cooperating specialized reasoning modules.

1. Introduction

We are interested in developing a methodology for integrating mechanized reasoning systems such as, e.g. Theorem Provers (TPs), Computer Algebra Systems (CASs), and Model Checkers (MCs). 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 encompass the wide range of functionalities needed in real-world applications (e.g. the development of a mathematical theory, the design and validation of hardware and software components). However, it is often the case that functionalities missing in a system are available in another.

By looking at the relevant literature it turns out that there are essentially two possible strategies to cope with the problem: system extension (Clarke and Zhao, 1992; Buchberger *et al.*, 1997; Harrison, 1998) and system integration (Jackson, 1994; Ballarin *et al.*, 1995; Armando and Ranise, 1998b; Bertoli *et al.*, 1998; Harrison and Théry, 1998). In both cases the main source of difficulty is the complexity of the services provided by state-of-the-art implementations. One critical issue in this endeavor is the integration

0747–7171/01/100305 + 28 $35.00/0

of the systems' underlying logics in a meaningful and semantically sound way. However there is more to integration than combining logics. As Boyer and Moore experienced when they integrated a decision procedure for linear arithmetic within their prover NQTHM (Boyer and Moore, 1979), the problem of integration is also a problem of control:

> *"The view of the decision procedure as a 'black box' is frequently destroyed by the need to pass large amounts of search strategic information back and forth between the two components."* (Boyer and Moore, 1988)

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. Specifications play a crucial role if properties such as, e.g. soundness and termination of the compound system need to be established. We build on the work presented in Giunchiglia *et al.* (1994) 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 layers: the *logic* layer (specifying the assertions manipulated by the system and the elementary deductions upon them), the *control* layer (specifying the inference strategies), and the *interaction* layer (specifying the interaction of the system with the environment). Notice that this layering allows for an additional and complementary way to structure the specifications w.r.t. the standard approach based on modularity. As a consequence, OMRS specifications are therefore more structured than conventional specifications. This domain-specific feature of the OMRS specification framework is fundamental to cope with the complexity of functionalities provided by state-of-the-art implementations. While the problem of specifying reasoning systems at the logic level has been addressed in Giunchiglia *et al.* (1994), in this paper we focus on the control level. We propose to specify the control layer by:

(1) 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;
(2) specifying proof strategies via *tactics*, i.e. expressions denoting sets of admissible derivations.

As a case study we give the OMRS specification of (a simplified form of) Constraint Contextual Rewriting. *Constraint Contextual Rewriting* (Armando and Ranise, 1998a) (CCR(X) for short) is a generalized form of contextual rewriting (Zhang, 1995) which incorporates (and is parametric in) the services provided by an external decision procedure. The case study is non-trivial since CCR(X) results from the combination of three distinguished reasoning modules: a simplifier, a rewrite engine, and a decision procedure. The OMRS specification we propose reflects this modularity and provides us with a detailed and formal account of the logic and the control aspects of the functionalities provided by each module as well as of their interplay.

The paper is organized as follows. In Section 2 we give an overview of the OMRS framework. We start in Section 2.1 by giving a formal account of the logic layer by introducing the notion of reasoning theory;[†] we then focus on the control layer by defining the notion of annotated reasoning theory (Section 2.2) and that of tactic theory (Section 2.3). In

[†] The concepts presented in Section 2.1 are a simplified account of analogous notions introduced in Giunchiglia *et al.* (1994) and Coglio *et al.* (2000).

Section 2.1 we also illustrate how reasoning theories, annotated reasoning theories, and tactic theories can be composed and made parametric. The complexity of the notions and notations we introduce are necessary to give a rigorous and reasonably concise account of the services provided by state-of-the-art reasoning systems. In Section 3 we substantiate this claim by outlining a structured specification of CCR(X). Section 4 is devoted to a comparison with the related work. Finally, in Section 5, we give some concluding remarks.

In this paper we focus on the validity and flexibility of the proposed framework to specify state-of-the-art mechanized reasoning systems. As a consequence, the discussion of general results following from the proposed theory (although there are some interesting ones) are outside the scope of the present paper. However, the paper shows that the OMRS framework provides the necessary concepts which allow to formally state and prove important properties of the specified systems as illustrated in the case study.

1.1. MATHEMATICAL NOTATIONS

Let A be a set, then A^* is the set of all finite sequences of elements of A; we write $[\,]$, $[a|a^*]$, $a_1^*@a_2^*$, and $|a^*|$ to denote the empty sequence, the sequence with head the element a and tail the sequence a^*, the concatenation of the sequences a_1^* and a_2^*, and the length of the sequence a^*, respectively. If A is a set then $\mathcal{P}_\omega(A)$ denotes the set of all finite subsets of A. If A is a set and $\approx$ an equivalence relation over A, $A/_{\approx}$ is the set of all equivalence classes of A, i.e. $A/_{\approx} = \{[\![a]\!] \mid a \in A\}$, where $[\![a]\!] = \{a' \in A \mid a' \approx a\}$. If A and B are sets, we write $A \uplus B$ to denote the disjoint union of A and B. If $f : A \to A'$ and $g : B \to B'$, $(f \uplus g) : A \uplus B \to A' \uplus B'$ is defined by $(f \uplus g)(a) = f(a)$ for $a \in A$ and $(f \uplus g)(b) = g(b)$ for $b \in B$.

Given a set T, a T-typed set is a pair $S = \langle S_0, \tau\rangle$ where S_0 is a set and $\tau : S_0 \to T$.[†] If S is a typed set, $\lfloor S \rfloor = S_0$. We lift $\in$, $\subseteq$, $\cap$, $\cup$, $\uplus$, and $\times$ to typed sets as follows:

(elem) $s \in S$ iff $s \in S_0$;
(sub) $S \subseteq S'$ iff $S_0 \subseteq S_0'$ and $\tau(s) = \tau'(s)$ for $s \in S_0$;
(int) $S = S' \cap S''$ iff $S_0 = S_0' \cap S_0''$ and $\tau(s) = \tau'(s) = \tau''(s)$ for $s \in S_0$;
(un) $S = S' \cup S''$ iff $S_0 = S_0' \cup S_0''$, $\tau(s) = \tau'(s) = \tau''(s)$ for $s \in S_0' \cap S_0''$, $\tau(s) = \tau'(s)$ for $s \in S_0' - S_0''$, and $\tau(s) = \tau''(s)$ for $s \in S_0'' - S_0'$;
(djun) $S = S' \uplus S''$ iff $S_0 = S_0' \uplus S_0''$ and $\tau = \tau' \uplus \tau''$;
(prod) $S = S' \times S''$ iff $S_0 = S_0' \times S_0''$ and $\tau(\langle s', s''\rangle) = \langle \tau'(s'), \tau''(s'')\rangle$ for $\langle s', s''\rangle \in S_0$.

If S' and S'' are T-typed sets, the T-typed set $S = S' \otimes S''$ is defined by $S_0 = \{\langle s', s''\rangle \in S_0' \times S_0'' \mid \tau'(s') = \tau''(s'')\}$ and $\tau(\langle s', s''\rangle) = \tau'(s') = \tau''(s'')$ for $\langle s', s''\rangle \in S_0$. If S is a typed set, we write $s{:}t \in S$ as an abbreviation for $(s \in S_0 \ \wedge\ \tau(s) = t)$. For any set T, we write $\emptyset$ to denote the empty T-typed set S characterized by $\lfloor S \rfloor = \emptyset$. If S is a T-typed set and $T' \subseteq T$, the T'-typed set $S' = S|_{T'}$ is defined by $S_0' = \{s \in S_0 \mid \tau(s) \in T'\}$ and $\tau'(s) = \tau(s)$ for $s \in S_0'$. If S is a T-typed set, the T^*-typed set $S' = S^*$ is defined by $S_0' = S_0^*$ and $\tau'([s_1, \ldots, s_n]) = [\tau(s_1), \ldots, \tau(s_n)]$ for $[s_1, \ldots, s_n] \in S_0'$. We write $\{s{:}t \mid \ldots\}$ to denote a typed set whose elements s and corresponding types t are defined as indicated by the expression in "…". When we write a typed set S where an ordinary set is

[†] In other words, a T-typed set is a set whose elements are uniquely labeled by elements of T. The definition implies that the same S is a T-typed set for each T that includes the range of τ (i.e. for each T such that $T \supseteq \{\tau(s) \mid s \in S_0\}$).

expected (i.e. where the typed set would make the expression not defined), S just stands for $\lfloor S \rfloor$.

If $g : T \to T'$, a g-typed function f from a T-typed set S to a T'-typed set S', also written $f : S \to_g S'$, is a function $f : S_0 \to S'_0$ such that $\tau'(f(s)) = g(\tau(s))$ for $s \in S_0$. A T-typed function f from a T-typed set S to a T-typed set S', also written $f : S \to_T S'$, is an id-typed function $f : S \to_{id} S'$, where $id : T \to T$ and $id(t) = t$ for $t \in T$. When we define an (ordinary or typed) function $f : A \to B$, we assume it is automatically lifted to $f : A^* \times A \to B^* \times B$ by $f(\langle [a_1, \ldots, a_n], a \rangle) = \langle [f(a_1), \ldots, f(a_n)], f(a) \rangle$. A T-typed relation r over a T-typed set S is a T-typed set $r \subseteq S$.[†] If S is a T-typed set, a T-typed relation $\approx$ over $S \otimes S$ is a T-typed equivalence over S iff $\approx_0$ is an equivalence over S_0; in this case, we have $S/_{\approx} = S'$ iff $T' = T$, $S'_0 = S_0/_{\approx}$, and $\tau'([\![s]\!]) = \tau(s)$ for $s \in S_0$.

2. Theory

2.1. Logic Layer

The logic layer of an OMRS specification describes the assertions manipulated by the system as well as the elementary deduction steps the system performs upon such assertions. For example, a resolution-based theorem prover may manipulate first-order clauses by resolving and factorizing them. As another example, a decider for linear arithmetic may manipulate polynomial inequalities by cross-multiplication and sum. At the logical level, the computations carried out by the system amount to constructing and manipulating structures consisting of assertions connected through elementary deduction steps (like proof trees).

There are two basic mechanisms to compose OMRS specifications at the logical level. The first mechanism consists in putting together the constituent elements (assertions and elementary deduction steps) of the specifications to form a larger specification. This form of composition is "well-defined" if the components satisfy some conditions relative to each other, namely that their common constituent elements are defined "in the same way". For example, a (logical) specification for term rewriting and one for clause resolution, which use the same entities as terms and atoms (respectively), can be put together yielding a specification for both rewriting and resolution over the common terms/atoms. The second mechanism is parameterization: the logic layer of an OMRS specification can contain some "replaceable" template parts. For each possible replacement, we obtain a (slightly) different specification. For example, the (logical) specification for a propositional decider may have propositional atoms as replaceable parts: such atoms can be replaced by first-order atomic formulas, polynomial inequalities, term equalities, and so on.

2.1.1. Sequent Systems

A *sequent system* is a quadruple $Ssys = \langle \Sigma, X, E, Q \rangle$. $\Sigma = \langle S, O \rangle$ is a *signature* where S is a set of *sorts*, and O is an $(S^* \times S)$-typed set of *operations*. If $o : \langle [s_1, \ldots, s_n], s \rangle \in O$, then $\langle [s_1, \ldots, s_n], s \rangle$, also written $[s_1, \ldots, s_n] \to s$ or $s_1 \cdots s_n \to s$, is the *arity* of o; $s_1, \ldots, s_n$ are the *argument sorts* of o, and s is the *result sort* of o. X is an S-typed set of *variables*, such that $\lfloor X \rfloor \cap \lfloor O|_{\{[\,]\} \times S} \rfloor = \emptyset$ and $X|_{\{s\}}$ is infinite for any $s \in S$. The S-typed set $\mathcal{T}$ of *terms* is the smallest one satisfying:

[†] Note that r_0 is an ordinary relation over S_0.

(var) $X \subseteq \mathcal{T}$;

(op) $o\!:\!s_1 \cdots s_n \!\rightarrow\! s \in O \;\;\wedge\;\; t_1\!:\!s_1,\ldots,t_n\!:\!s_n \in \mathcal{T} \;\;\Rightarrow\;\; o(t_1,\ldots,t_n)\!:\!s \in \mathcal{T}$.

We may write o instead of $o()$. The S-typed set $\mathcal{OT}$ of *operation terms* is $\mathcal{OT} = \{o(x_1,\ldots,x_n)\!:\!s \mid o\!:\!s_1\cdots s_n \!\rightarrow\! s \in O \;\wedge\; x_1\!:\!s_1,\ldots,x_n\!:\!s_n \!\in\! X \;\wedge\; x_1 \neq \cdots \neq x_n\}$. An *instantiation* is an S-typed function $\iota : X \rightarrow_S \mathcal{T}$, which is lifted to terms, $\iota : \mathcal{T} \rightarrow_S \mathcal{T}$, by $\iota(o(t_1,\ldots,t_n)) = o(\iota(t_1),\ldots,\iota(t_n))$. I is the set of all instantiations. The *instantiation composition* function $\circ : I \times I \rightarrow I$ is defined by $(\iota \circ \iota')(x) = \iota(\iota'(x))$. The *identity instantiation* $\mathtt{idi} \in I$ is defined by $\mathtt{idi}(x) = x$ for all $x \in X$. The S-typed set of *equations* is $\mathcal{E} = \mathcal{T} \otimes \mathcal{T}$. For each $\langle t_1, t_2\rangle \in \mathcal{E}$, we may write $t_1 = t_2$ instead of $\langle t_1, t_2\rangle$.[†] The consequence relation $\vdash \subseteq \mathcal{P}_\omega(\mathcal{E}) \times \mathcal{E}$ over equations is the usual entailment relation for equational logic[‡]—see, e.g., Ehrig and Mahr (1985). Let E be an S-typed set of equations, i.e. $E \subseteq \mathcal{E}$. The S-typed equivalence relation $\equiv\, \subseteq \mathcal{T} \otimes \mathcal{T}$ is defined by $(t_1 \equiv t_2 \Leftrightarrow E \vdash t_1 = t_2)$. Q is a set of sorts in S, i.e. $Q \subseteq S$. The Q-typed set of *sequents* is $Sq = \widetilde{\mathcal{T}}|_Q$, where $\widetilde{\mathcal{T}} = \mathcal{T}/_{\equiv}$. An instantiation $\iota \in I$ is lifted to sequents, $\iota : Sq \rightarrow_Q Sq$, by $\iota(sq) = [\![\iota(t)]\!]$ where $t \in \mathcal{T}|_Q$ and $[\![t]\!] = sq$. We may write $sq[\iota]$ instead of $\iota(sq)$.

Sequents represent the logical assertions manipulated by the reasoning system being specified and provide a more general concept than the notion of sequent used in sequent calculi as, e.g. in Gentzen (1934). Sorts identify kinds of syntactical entities (e.g. literals, atoms, clauses, polynomials, polynomial inequalities) used, directly or indirectly, to build sequents. Operations identify constructions and manipulations of such entities (e.g. building a unary clause from a literal, multiplying a polynomial by a coefficient, conjoining two clauses). Equations express properties of these constructions and manipulations (e.g. that conjoining clauses are commutative, associative, and idempotent; that multiplying a polynomial by a coefficient amounts to multiplying all monomial coefficients by such a coefficient). Q indicates which kinds of entities count as sequents (e.g. clauses, polynomial inequalities), as opposed to the others (e.g. literals, atoms, polynomials) that are typically used as component parts of sequents. Sequents are defined as equivalence classes of terms in order to take equations into account: for example, if the operation of conjoining clauses is commutative, associative, and idempotent (through suitable equations), a clause can be effectively regarded as a (finite) set of literals. Because of the presence of variables, sequents can be regarded as "schematic", i.e. containing placeholders for unspecified pieces of syntax; instantiations serve to fill in such placeholders.

2.1.2. REASONING THEORIES

A *reasoning theory* (*RTh*) is a pair $Rth = \langle Ssys, R\rangle$, where $Ssys$ is a sequent system, and R is an $(Sq^* \times Sq)$-typed set whose elements are called *rules.* If $r\!:\!\langle [sq_1,\ldots,sq_n], sq\rangle \in R$, we may write $r\!:\![sq_1,\ldots,sq_n] \!\rightarrow\! sq$, $r\!:\!sq_1\cdots sq_n \!\rightarrow\! sq$, or

$$\frac{sq_1 \qquad \cdots \qquad sq_n}{sq}\; r \tag{1}$$

[†] Context will always disambiguate these object-level equations from the meta-level equations we use in formal definitions.

[‡] Note that $\vdash$ is not a typed relation, but just an ordinary relation.

instead of $r\!:\!\langle [sq_1,\ldots,sq_n], sq\rangle$, and we may write terms (of sorts in Q) instead of sequents (i.e. instead of terms' equivalence classes); $sq_1,\ldots,sq_n$ are the *premises* of r, and sq the *conclusion* of r.[†]

An RTh constitutes the logic layer of an OMRS specification. The sequent system describes the assertions manipulated by the reasoning system by means of sequents and some auxiliary information (instantiations, etc.). The rules describe the elementary deduction steps over the assertions. A rule $r\!:\!sq_1\cdots sq_n \rightarrow sq$ expresses the fact that validity of $sq[\iota]$ is implied by the validity of $sq_1[\iota],\ldots,sq_n[\iota]$ for any $\iota \in I$. For example, we may have a rule with two (terms representing) polynomial inequalities as premises, and as conclusion the result of cross-multiplying and adding them (expressed symbolically as a term with suitable operations applied to the polynomials). Such a rule expresses that the result of cross-multiplying and adding two polynomial inequalities logically follows from them: it is typically used to derive simpler polynomials by canceling monomials.

2.1.3. DERIVATION STRUCTURES

Let *Rth* be an RTh. The $(Sq^* \times Sq)$-typed set Δ of *derivation structures* is the smallest one satisfying:

$$\begin{array}{ll}
\text{(sq)} & sq \in Sq \quad\Rightarrow\quad sq\!:\!\langle [sq], sq\rangle \in \Delta;\\
\text{(rul)} & r\!:\!sq_1\cdots sq_n \rightarrow sq \in R \quad\wedge\quad \iota \in I \quad\wedge\quad \delta_1\!:\!\langle \vec{sq}_1, sq_1[\iota]\rangle,\ldots,\delta_n\!:\!\langle \vec{sq}_n, sq_n[\iota]\rangle \in \Delta \quad\Rightarrow\\
 & \langle [\delta_1,\ldots,\delta_n], r, \iota\rangle\!:\!\langle \vec{sq}_1 @ \cdots @ \vec{sq}_n, sq[\iota]\rangle \in \Delta.
\end{array}$$

If $\delta\!:\!\langle [sq_1,\ldots,sq_n], sq\rangle \in \Delta$, we may write $\delta\!:\![sq_1,\ldots,sq_n] \rightarrow sq$ or $\delta\!:\!sq_1\cdots sq_n \rightarrow sq$ instead of $\delta\!:\!\langle [sq_1,\ldots,sq_n], sq\rangle$; $sq_1,\ldots,sq_n$ are the *open sequents* of δ, and sq is the *conclusion* of δ.[‡] An instantiation $\iota \in I$ is lifted to derivation structures, $\iota : \Delta \rightarrow_\iota \Delta$, by $\iota(\langle [\delta_1,\ldots,\delta_n], r, \iota'\rangle) = \langle [\iota(\delta_1),\ldots,\iota(\delta_n)], r, \iota \circ \iota'\rangle$.[§] We may write $\delta[\iota]$ instead of $\iota(\delta)$. The (partial) *derivation structure composition* function $\_\,;\_ : \Delta^* \times \Delta \xrightarrow{\text{p}} \Delta$ is the smallest one satisfying:

$$\begin{array}{ll}
\text{(sq)} & sq \in Sq \quad\wedge\quad \delta\!:\!\vec{sq} \rightarrow sq \in \Delta \quad\Rightarrow\quad [\delta]; sq = \delta;\\
\text{(rul)} & \langle [\delta_1,\ldots,\delta_n], r, \iota\rangle \in \Delta \quad\wedge\quad \vec{\delta}_1;\delta_1,\ldots,\vec{\delta}_n;\delta_n \in \Delta \quad\Rightarrow\\
 & [\vec{\delta}_1 @ \cdots @ \vec{\delta}_n]; \langle [\delta_1,\ldots,\delta_n], r, \iota\rangle = \langle [\vec{\delta}_1;\delta_1,\ldots,\vec{\delta}_n;\delta_n], r, \iota\rangle.
\end{array}$$

A derivation structure corresponds to a proof tree. A derivation structure consisting of a single sequent corresponds to a tree with a single node that is both the root and the (only) leaf. A derivation structure of the form $\langle [\delta_1,\ldots,\delta_n], r, \iota\rangle$, with $r\!:\!sq_1\cdots sq_n \rightarrow sq \in R$, corresponds to a tree with $sq[\iota]$ as root, and the n trees corresponding to $\delta_1,\ldots,\delta_n$ as

[†] In this definition, we call "rules" the elements r of the (untyped) set $\lfloor R \rfloor$, and use typing (R is a typed set) to associate premises $\vec{sq}$ and a conclusion sq with a rule r. In similar formalisms in the literature, r is called the "rule label" and the "rule" is considered to be the whole triple $\langle r, \vec{sq}, sq\rangle$. Our approach of expressing premises and conclusion as information that is "external", but closely connected (through the typing), to a rule, is consistent with our treatment of operations, variables, etc., whose arities and sorts are also given "externally" through typing.

[‡] Note that the typing of a derivation structure δ can be determined from δ itself. The reason why we define Δ as a typed set, as opposed to an ordinary set, is to enable some conveniently compact definitions for annotated reasoning theories (see Section 2.2).

[§] Note that ι is indeed a ι-typed function over derivation structures, because it changes the open sequents and conclusion (by applying ι to them).

subtrees (which have $sq_1[\iota], \ldots, sq_n[\iota]$ as roots); the root is labeled by r and ι, which provide the "justification" for the connection between the root and the subtrees. If $\delta : sq_1 \cdots sq_n \to sq \in \Delta$, then sq and $sq_1, \ldots, sq_n$ are respectively the root and leaves of the tree corresponding to δ. Applying an instantiation to a derivation structure amounts to applying it to all the constituent sequents, and updating the justifications accordingly. The derivation structure composition ";" corresponds to replacing the leaves of a tree (open sequents) with subtrees having such sequents as roots.

At the logical level, the computations performed by a system amount to creating and manipulating derivation structures. For example, a resolution-based theorem prover may start with some sequents (the clauses given as input), and incrementally generate new sequents (resolvents) by resolution: this amounts to building a derivation structure from leaves to root. As another example, a goal-directed theorem prover may start with a single sequent and incrementally generate new sequents by backward inference: this amounts to building a derivation structure from root to leaves (possibly, having no leaves in the end and thus proving the initial goal). In other cases, derivation structures may be built in a mixed fashion (i.e. partly from leaves to root, partly from root to leaves). In addition, during construction instantiations may be applied to the whole derivation structure: this may happen, for instance, when a system eliminates an existential quantifier from a formula, and later in the proof it finds (and applies) a suitable substitution for the variable that was eliminated.

2.1.4. FAITHFUL INCLUSIONS

An RTh Rth_0 is *faithfully included* in an RTh Rth_1, also written $Rth_0 \hookrightarrow Rth_1$, iff $S_0 \subseteq S_1$, $O_1|_{S_1^* \times S_0} = O_0$, $X_1|_{S_0} = X_0$, $E_1|_{S_0} = E_0$, $Q_1 \cap S_0 = Q_0$, and $R_0 \subseteq R_1$.[†] If $Rth_0 \hookrightarrow Rth_1$ then $\mathcal{T}_1|_{S_0} = \mathcal{T}_0$, $\widetilde{\mathcal{T}_1}|_{S_0} = \widetilde{\mathcal{T}_0}$, $Sq_1|_{Q_0} = Sq_0$, and there exist functions $\phi : I_0 \to I_1$ and $\psi : I_1 \to I_0$ such that $\phi(\iota)(x) = \textbf{if } x \in X_0 \textbf{ then } \iota(x) \textbf{ else } x$ and $\psi(\iota)(x) = \iota(x)$. $\hookrightarrow$, as a binary relation over RThs, is a partial order.

The notion of faithful inclusion formally captures the intuition of an RTh being part of another RTh. If $Rth_0 \hookrightarrow Rth_1$, all the sorts, operations, variables, terms, equations, sequents, and rules of Rth_0 are also in Rth_1. In order for sequents of Rth_0 to be also sequents of Rth_1, it is necessary that all equations in Rth_1 of sort in Rth_0 are also equations of Rth_0, and that all terms in Rth_1 of sort in Rth_0 are also terms of Rth_0 (otherwise, sequents would be different equivalence classes in Rth_0 and in Rth_1). In order for the latter requirement to be satisfied, it is necessary that all variables in Rth_1 of sort in Rth_0 are also variables of Rth_0, and that all operations in Rth_1 with result sort in Rth_0 are also operations in Rth_0 (and therefore all their argument sorts must be in Rth_0). All these requirements are indeed enforced in the formal definition above. A faithful inclusion also guarantees that instantiations can be "extended" from Rth_0 to Rth_1 and "restricted" from Rth_1 to Rth_0.

An example of faithful inclusion is that of an RTh specifying propositional reasoning over first-order atomic formulas, into an RTh specifying a complex first-order theorem prover that employs various reasoning techniques (propositional being one of them). Conceivably, RThs specifying the other reasoning techniques are faithfully included in it as well. Another example is that of an RTh specifying polynomials, into an RTh specifying arithmetical reasoning. The RTh for polynomials has no sequents and no rules (i.e. $Q = \emptyset$

[†] Note that R_1 may include "new" (i.e. not in R_0) rules involving sequents of Rth_0 only.

and $R = \emptyset$), but just specifies terms representing polynomials; this is indeed a perfectly legal RTh, which represents the part of the RTh for arithmetic reasoning that defines the polynomials. Faithful inclusions of RThs with no sequents and no rules frequently arise in practice when composing RThs together (see examples below).

2.1.5. GLUING

Let Rth_1 and Rth_2 be RThs. Let $\mathit{shared}(Rth_1, Rth_2) = Rth_0$, where $S_0 = S_1 \cap S_2$, $O_0 = O_1 \cap O_2$, $X_0 = X_1 \cap X_2$, $E_0 = E_1 \cap E_2$, $Q_0 = Q_1 \cap Q_2$, and $R_0 = R_1 \cap R_2$. If Rth_0 is defined[†] and if $X_0|_{\{s\}}$ is infinite for any $s \in S_0$, then Rth_0 is an RTh. Rth_1 and Rth_2 are *glueable*, also written $Rth_1 \bowtie Rth_2$, iff Rth_0 is an RTh, $Rth_0 \hookrightarrow Rth_1$, and $Rth_0 \hookrightarrow Rth_2$. If $Rth_1 \bowtie Rth_2$, the result of *gluing* Rth_1 and Rth_2 is the RTh $Rth = Rth_1 + Rth_2$ defined by $S = S_1 \cup S_2$, $O = O_1 \cup O_2$, $X = X_1 \cup X_2$, $E = E_1 \cup E_2$, $Q = Q_1 \cup Q_2$, $R = R_1 \cup R_2$. We have $Rth_1 \hookrightarrow Rth$, $Rth_2 \hookrightarrow Rth$, $\mathcal{T}_0 = \mathcal{T}_1 \cap \mathcal{T}_2$, $Sq_0 = Sq_1 \cap Sq_2$, $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$, and $Sq = Sq_1 \cup Sq_2$. Gluing of RThs is associative, commutative, and idempotent.

Gluing formalizes the intuition of "putting together" the constituent elements (sorts, operations, etc.) of two or more RThs. In order for this to make sense, it is required that the "intersection" of the RThs is a well-defined RTh and that it is a well-defined part of (i.e. faithfully included in) each of the RThs. If these conditions are met, the result is indeed a well-defined RTh, in which the components are faithfully included. The associativity, commutativity, and idempotence properties of gluing expose the fact that given two or more RThs, if they are glueable then they can be glued in any relative order counting each RTh any number of times, and the same result is obtained in all cases. This amounts to saying that + can be lifted to a (partial) operator over sets of RThs.

As an example of gluing, consider an RTh for term rewriting, and an RTh for clause resolution. If these two RThs define the same entities as terms and atoms (respectively) (i.e. if their intersection is a well-defined RTh that defines the common terms/atoms), then we can glue them together and obtain a new RTh specifying both term rewriting and clause resolution over the same terms/atoms. Note that the intersection RTh defining the terms/atoms has no rules.

2.1.6. PARAMETERIZATION

A *parameterized RTh* (*pRTh*) is a pair $PRth = \langle Rth_\pi, Rth_\beta \rangle$, where Rth_π and Rth_β are RThs, and $Rth_\pi \hookrightarrow Rth_\beta$. Rth_π and Rth_β are, respectively, the *parameter* and *body* of *PRth*. We may write $Rth_\beta[Rth_\pi]$ instead of $\langle Rth_\pi, Rth_\beta \rangle$.

A *replacement mapping* ρ from an RTh Rth_1 to an RTh Rth_2, also written $\rho : Rth_1 \to Rth_2$, is a quadruple $\rho = \langle \rho_S, \rho_O, \rho_X, \rho_R \rangle$, where:

(srt) $\rho_S : S_1 \to S_2$;
(op) $\rho_O : O_1 \to_{\rho_S} O_2$;
(var) $\rho_X : X_1 \to_{\rho_S} X_2$, such that if $\rho_X(x) = \rho_X(x')$ then $x = x'$;
(eq) if $(t_1 = t_2) \in E_1$ then $E_2 \vdash (\rho_T(t_1) = \rho_T(t_2))$, where $\rho_T : \mathcal{T}_1 \to_{\rho_S} \mathcal{T}_2$ is defined by $\rho_T(x) = \rho_X(x)$ for $x \in X_1$, and $\rho_T(o(t_1, \ldots, t_n)) = \rho_O(o)(\rho_T(t_1), \ldots, \rho_T(t_n))$;
(sqsrt) $s \in Q_1 \;\Rightarrow\; \rho_S(s) \in Q_2$;

[†] Recall that the intersection of two typed sets is defined only if the common elements of the sets have the same type; see Section 1. For example, R_0 is defined only if each rule r belonging to both R_1 and R_2 has the same type $\langle \vec{sq}, sq \rangle$ in both R_1 and R_2.

(rul) $\rho_{\mathrm{R}} : R_1 \rightarrow_{\rho_{\mathrm{Q}}} R_2$, where $\rho_{\mathrm{Q}} : Sq_1 \rightarrow_{\rho_{\mathrm{S}}} Sq_2$ is defined by $\rho_{\mathrm{Q}}(sq) = [\![\rho_{\mathrm{T}}(t)]\!]$ where $t \in \mathcal{T}_1|_{Q_1}$ and $[\![t]\!] = sq$.

$\rho_{\mathrm{I}} : I_1 \rightarrow I_2$ is defined by

$$\rho_{\mathrm{I}}(\iota)(x) = \textbf{if } (x = \rho_{\mathrm{X}}(x')) \textbf{ then } \rho_{\mathrm{T}}(\iota(x')) \textbf{ else } x.$$

$\rho_\Delta : \Delta_1 \rightarrow_{\rho_{\mathrm{Q}}} \Delta_2$ is defined by

(sq) $sq \in Sq \quad\Rightarrow\quad \rho_\Delta(sq) = \rho_{\mathrm{Q}}(sq)$;
(rul) $\rho_\Delta(\langle[\delta_1, \ldots, \delta_n], r, \iota\rangle) = \langle[\rho_\Delta(\delta_1), \ldots, \rho_\Delta(\delta_n)], \rho_{\mathrm{R}}(r), \rho_{\mathrm{I}}(\iota)\rangle$.

We may drop the indices and just write ρ instead of ρ_{S}, ρ_{O}, etc.

Let *PRth* be a pRTh. Let Rth_0 be an RTh. Let $\rho : Rth_\pi \rightarrow Rth_0$. The result of *replacing* the parameter of *PRth* with Rth_0 by ρ is the RTh *Rth* defined as follows, where we also lift ρ to Rth_β, $\rho : Rth_\beta \rightarrow Rth$:

(srt) $S = S_0 \uplus (S_\beta - S_\pi)$;
(srplc) $s \in S_\beta - S_\pi \quad\Rightarrow\quad \rho(s) = s$;
(op) $O = O_0 \uplus \{o : \rho(\vec{s} \rightarrow s) \mid o : \vec{s} \rightarrow s \in O_\beta - O_\pi\}$;
(orplc) $o \in O_\beta - O_\pi \quad\Rightarrow\quad \rho(o) = o$;
(var) $X = X_0 \uplus (X_\beta - X_\pi)$;
(vrplc) $x \in X_\beta - X_\pi \quad\Rightarrow\quad \rho(x) = x$;
(eq) $E = E_0 \uplus \{(\rho(t_1) = \rho(t_2)) : s \mid (t_1 = t_2) : s \in E_\beta - E_\pi\}$;
(sqsrt) $Q = Q_0 \uplus (Q_\beta - Q_\pi)$;
(rul) $R = R_0 \uplus \{r : \rho(\vec{sq} \rightarrow sq) \mid r : \vec{sq} \rightarrow sq \in R_\beta - R_\pi\}$;
(rrplc) $r \in R_\beta - R_\pi \quad\Rightarrow\quad \rho(r) = r$.

We have $Rth_0 \hookrightarrow Rth$. When ρ is clear from context, we may write $Rth_\beta[Rth_0/Rth_\pi]$ to denote *Rth*.

A pRTh is substantially an RTh (the body Rth_β) with a distinguished well-defined (i.e. faithfully included) part (the parameter Rth_π). In order to replace the parameter with another RTh Rth_0, it is necessary to indicate, for each constituent element of Rth_π (sorts, operations, etc.), the element of Rth_0 that replaces it. This is expressed by a replacement mapping from Rth_π to Rth_0. $Rth_\beta[Rth_0/Rth_\pi]$ is obtained by taking the disjoint union (to avoid unintended "name conflicts") of the elements of Rth_0 and the elements of Rth_β that are not in Rth_π (because those in Rth_π have been replaced by those in Rth_0); the latter elements must be suitably changed to reflect the replacement.

A pRTh constitutes the logic layer of an OMRS specification for a system that is parameterized over some aspect(s), or, in other words, that contains some "generic" parts, with explicit, visible "hooks" to these generic parts. By suitably connecting the hooks to another system (that "fits" the hooks), a new, more specific system is obtained. Parameterization is indeed a key to building open, re-usable, and compositional systems. A simple example of pRTh is one whose body specifies propositional reasoning, where atoms are generic in the sense that their structure is not specified; there is just a sort for atoms. The parameter of the pRTh basically consists of the sort for atoms only. Now, if we have an RTh for arithmetic reasoning over polynomial inequalities, we can replace the sort for atoms with the sort of such inequalities. The result is an RTh specifying both arithmetic and propositional reasoning over polynomial inequalities.

2.2. CONTROL LAYER: ANNOTATIONS

The logic layer of an OMRS specification (an RTh or pRTh) describes how the system may manipulate the logical information (i.e. the logical assertions). The control layer must specify how the system actually manipulates such information, i.e. which strategies are used to select and apply the inference steps at each point of the computation. Most real-world systems carry out their control strategies by making use of (often extensive) non-logical information, used exactly for control purposes. Examples of such control information are some history about how an assertion was produced, the number of times a certain inference step has been applied, the order in which some assertions must be selected for applying some reasoning steps, etc. Control information is used and modified during computation, at the same time as logical inferences are performed.

In OMRS specifications, we represent control information by enriching the sequents with annotations carrying this additional information. The use and manipulations of these annotations are expressed by lifting rules to also consider annotations (i.e. express how annotations are used and modified). More precisely, given an RTh for the logic layer of an OMRS specification, the control layer contains another RTh whose sequents and rules constitute the "annotated counterpart" of the first RTh. There is a formal relation between the two RThs: intuitively, that by discarding the annotations from the second RTh we obtain the first RTh. The RTh with annotations is just like any other RTh, but its sequents and rules deal with both logical and control information. This allows to nicely lift to control the formal notions developed for RThs (e.g. derivation structures, gluing, parameterization).

2.2.1. ANNOTATED REASONING THEORIES

An *annotated RTh* (*ARTh*) over an RTh *Rth* is a pair $ARth = \langle Rth^{\mathrm{A}}, \epsilon\rangle$, where Rth^{A} is an RTh, and ϵ is an *erasing mapping* from Rth^{A} to *Rth*, also written[†] $\epsilon : Rth^{\mathrm{A}} \not\rightarrow Rth$, i.e. a quadruple $\epsilon = \langle \epsilon_{\mathrm{S}}, \epsilon_{\mathrm{X}}, \epsilon_{\mathrm{O}}, \epsilon_{\mathrm{R}}\rangle$ where:

(srt) $\epsilon_{\mathrm{S}} : S^{\mathrm{A}} \rightarrow S \uplus \{\cdot\}$;

(var) $\epsilon_{\mathrm{X}} : X^{\mathrm{A}} \rightarrow_{\epsilon_{\mathrm{S}}} X \uplus \{\cdot{:}\cdot\}$[‡] such that if $\epsilon_{\mathrm{X}}(x) = \epsilon_{\mathrm{X}}(x') \neq \cdot$ then $x = x'$;

(op) $\epsilon_{\mathrm{O}} : \mathcal{OT}^{\mathrm{A}} \rightarrow_{\epsilon_{\mathrm{S}}} \mathcal{T} \uplus \{\cdot{:}\cdot\}$ such that if $\epsilon_{\mathrm{O}}(o(x_1, \ldots, x_n)) \neq \cdot$ then all the variables occurring in $\epsilon_{\mathrm{O}}(o(x_1, \ldots, x_n))$ are in $\{\epsilon_{\mathrm{X}}(x_i) \mid 1 \leq i \leq n \ \wedge \ \epsilon_{\mathrm{X}}(x_i) \neq \cdot\}$;

(eq) if $(t_1 = t_2){:}s \in E^{\mathrm{A}}$ and $\epsilon_{\mathrm{S}}(s) \neq \cdot$ then $E \vdash (\epsilon_{\mathrm{T}}(t_1) = \epsilon_{\mathrm{T}}(t_2))$, where $\epsilon_{\mathrm{T}} : \mathcal{T}^{\mathrm{A}} \rightarrow_{\epsilon_{\mathrm{S}}} \mathcal{T} \uplus \{\cdot{:}\cdot\}$ is defined by $\epsilon_{\mathrm{T}}(x) = \epsilon_{\mathrm{X}}(x)$ for $x \in X^{\mathrm{A}}$, and $\epsilon_{\mathrm{T}}(o(t_1, \ldots, t_n)) = \iota(\epsilon_{\mathrm{O}}(o(x_1, \ldots, x_n)))$ where $\iota(\epsilon_{\mathrm{X}}(x_i)) = t_i$ for $1 \leq i \leq n$ with $\epsilon_{\mathrm{X}}(x_i) \neq \cdot$;

(sqsrt) $s \in Q^{\mathrm{A}} \quad \wedge \quad \epsilon_{\mathrm{S}}(s) \neq \cdot \quad \Rightarrow \quad \epsilon_{\mathrm{S}}(s) \in Q$;

(rul) $\epsilon_{\mathrm{R}} : R^{\mathrm{A}} \rightarrow_{\epsilon_{\mathrm{Q}}} \Delta \uplus \{\cdot{:}\cdot\}$, where $\epsilon_{\mathrm{Q}} : Sq^{\mathrm{A}} \rightarrow_{\epsilon_{\mathrm{S}}} Sq \uplus \{\cdot{:}\cdot\}$ is defined by $\epsilon_{\mathrm{Q}}(sq) = [\![\epsilon_{\mathrm{T}}(t)]\!]$ where $t \in \mathcal{T}^{\mathrm{A}}|_{Q^{\mathrm{A}}}$ and $[\![t]\!] = sq$, where we consider $[\![\cdot]\!] = \cdot$, $\langle \vec{sq}, \cdot\rangle = \cdot$, and $[\cdot|\vec{sq}] = \vec{sq}$ for $\vec{sq} \in (Sq \uplus \{\cdot\})^*$.

[†] We use slashed arrows $\not\rightarrow$ in order to distinguish erasing mappings from replacement mappings between RThs.

[‡] By $\{\cdot{:}\cdot\}$ we denote the $\{\cdot\}$-typed singleton set containing $\cdot$ as the only element (whose type is obviously $\cdot$). We are using the same entity $\cdot$ to type itself, which is perfectly allowed by the definition of a typed set given in Section 1.

$\epsilon_{\mathrm{I}} : I^{\mathrm{A}} \to I$ is defined by

$$\epsilon_{\mathrm{I}}(\iota)(x) = \mathbf{if}\ (x = \epsilon_{\mathrm{X}}(x'))\ \mathbf{then}\ \epsilon_{\mathrm{T}}(\iota(x'))\ \mathbf{else}\ x.$$

$\epsilon_\Delta : \Delta^{\mathrm{A}} \to_{\epsilon_{\mathrm{Q}}} \Delta \uplus \{\cdot : \cdot\}$ is defined by:

(sq) $sq \in Sq^{\mathrm{A}} \quad \Rightarrow \quad \epsilon_\Delta(sq) = \epsilon_{\mathrm{Q}}(sq)$;

(rul) $\epsilon_\Delta(\langle[\delta_1, \dots, \delta_n], r, \iota\rangle) = [\epsilon_\Delta(\delta_1), \dots, \epsilon_\Delta(\delta_n)]; (\epsilon_{\mathrm{R}}(r)[\epsilon_{\mathrm{I}}(\iota)])$, where we consider $\vec{\delta}; \cdot = \cdot$ and $[\cdot|\vec{\delta}] = \vec{\delta}$ for $\vec{\delta} \in (\Delta \uplus \{\cdot\})^*$.

We may drop the indices and just write ϵ instead of ϵ_{S}, ϵ_{X}, etc.

An ARTh is just an RTh Rth^{A}, plus an erasing mapping ϵ from Rth^{A} to its non-annotated counterpart *Rth*. The terms of Rth^{A} encode both logical and control information, while the terms of *Rth* encode logical information only. The action of ϵ on terms consists of erasing the control content, leaving the logical content untouched. Some terms of Rth^{A} might contain only control (i.e. no logical) information; such terms are mapped to $\cdot$ by ϵ.[†] ϵ maps each sort s of Rth^{A} either to a sort of *Rth* whose terms have the same logical content of the terms of sort s in Rth^{A}, or to $\cdot$ if the terms of sort s have no logical content. Variables are injectively mapped by ϵ consistently with the sort mapping: this establishes a bijective correspondence between variables in Rth^{A} "carrying" logical content (i.e. having sorts whose terms carry logical content) and variables in *Rth*. Such a correspondence allows instantiations of Rth^{A} to be uniquely mapped to instantiations in *Rth*, and obviously serves to map terms of Rth^{A} to terms of *Rth*. Rather than just mapping each operation of Rth^{A} to an operation of *Rth* (or to $\cdot$) consistently with the sort mapping, ϵ maps each term in Rth^{A} of the form $o(x_1, \dots, x_n)$, with $x_1, \dots, x_n$ all distinct variables, to a term in *Rth* whose variables are all among $\epsilon(x_1), \dots, \epsilon(x_n)$. This is more general than mapping operations to operations, which would correspond to map $o(x_1, \dots, x_n)$ to $\epsilon(o)(\epsilon(x_{i_1}), \dots, \epsilon(x_{i_m}))$ (where $i_1 < \cdots < i_m$ are all the indices between $1, \dots, n$ which are not mapped to $\cdot$ by ϵ). This generality is necessary in most practical cases to avoid introducing additional operations into *Rth* just to serve as images for ϵ.[‡] The action of ϵ over generic terms is determined by its action on variables and on the terms of the form $o(x_1, \dots, x_n)$. It is required that the ϵ-images of equations in Rth^{A} involving logical information, are consequences of the equations in *Rth*. This induces a well-defined erasing mapping from sequents of Rth^{A} (asserting logical and control information) to sequents of *Rth* (asserting logical information only); sequents of Rth^{A} asserting no logical information are just mapped to $\cdot$.

An important requirement is that each rule in Rth^{A} having as conclusion a sequent with logical content, is mapped by ϵ to a derivation structure in *Rth* whose target and open sequents correspond to the results of applying ϵ to conclusion and premises of the rule.

[†] An equivalent point of view is that ϵ is a partial mapping. We have chosen to define it as a total mapping, with an adjoined element $\cdot$, because it allows more convenient formulations. Indeed, ϵ can also be defined as a special case of a mapping from lists of terms to lists of terms, in the Lawvere theories associated with the sequent systems, that maps each singleton list in Rth^{A} to either another singleton list in *Rth* or to the empty list ($\cdot$ in our formulation).

[‡] A case that frequently arises in practice is having in Rth^{A} an operation $o{:}\, s_1, s_2 \to s$, where $\epsilon(s_1) = s_1$, $\epsilon(s_2) = \cdot$, and $\epsilon(s) = s_1$. $o(t_1, t_2)$ associates some logical information (encoded by t_1) with some control information (encoded by t_2). ϵ should discard t_2 leaving t_1 untouched, i.e. $\epsilon(o(t_1, t_2)) = t_1$. If o were to be mapped to some operation in *Rth*, *Rth* should explicitly contain an operation $id{:}\, s_1 \to s_1$ and an equation $id(x_1) = x_1$ (which achieve the desired effect). By mapping $o(x_1, x_2)$ to x_1 the same effect is achieved without any need to introduce operations and equations in *Rth*.

This requirement guarantees logical "soundness": the annotated inferences expressed by $\mathit{Rth}^{\mathrm{A}}$ "agree" with those in *Rth* w.r.t. the logical content. ϵ maps each rule of $\mathit{Rth}^{\mathrm{A}}$ to a derivation structure of *Rth*, rather than just to a rule (which is less general), to provide more flexibility: an annotated inference step may thus correspond to multiple non-annotated inference steps.[†] Annotated derivation structures of $\mathit{Rth}^{\mathrm{A}}$ are mapped by ϵ to non-annotated derivation structures of *Rth*. Viewing derivation structures as proof trees, the mapping works by replacing the connection between the root and the immediately connected nodes by the non-annotated proof tree corresponding to the annotated rule, and recursively carrying out the same kind of replacement on subtrees. Note that a subtree whose root carries no logical information is just discarded from the whole tree (because it does not contribute to the logical inferences).

Given an RTh for (first-order) clause resolution, an example of ARTh is one that annotates the literals of clauses by numeric indices. Such ARTh contains, among others, terms for indexed literals (i.e. literals paired with indices) and indexed clauses (i.e. disjunctions of indexed literals). ϵ maps them to terms for literals and clauses, respectively. Annotated rules express resolution constrained by indices in same way (e.g. resolve literals with the same index, or the literals with the greatest indices within their clauses), and possibly express how indices are updated (e.g. increment some indices after each resolution, take the maximum of various indices). A mundane instance of this example is lock resolution (Boyer, 1971).

2.2.2. GLUING AND PARAMETERIZATION

An ARTh ARth_0 over an RTh Rth_0 is *faithfully included* in an ARTh ARth_1 over an RTh Rth_1, also written $\mathit{ARth}_0 \hookrightarrow \mathit{ARth}_1$, iff $\mathit{Rth}_0 \hookrightarrow \mathit{Rth}_1$, $\mathit{Rth}_0^{\mathrm{A}} \hookrightarrow \mathit{Rth}_1^{\mathrm{A}}$, and $\epsilon_0(\alpha) = \epsilon_1(\alpha)$ for all $\alpha \in S_0^{\mathrm{A}} \uplus X^{\mathrm{A}} \uplus \mathcal{OT}_0^{\mathrm{A}} \uplus R_0^{\mathrm{A}}$. $\hookrightarrow$, as a binary relation over ARThs, is a partial order.

An ARTh ARth_1 over an RTh Rth_1 and an ARTh ARth_2 over an RTh Rth_2 are *glueable*, also 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 $\epsilon_1(\alpha) = \epsilon_2(\alpha)$ for all $\alpha \in (S_1^{\mathrm{A}} \cap S_2^{\mathrm{A}}) \uplus (X_1^{\mathrm{A}} \cap X_2^{\mathrm{A}}) \uplus (\mathcal{OT}_1^{\mathrm{A}} \cap \mathcal{OT}_2^{\mathrm{A}}) \uplus (R_1^{\mathrm{A}} \cap R_2^{\mathrm{A}})$. If $\mathit{ARth}_1 \bowtie \mathit{ARth}_2$ then $\mathit{ARth}_0 = \langle \mathit{Rth}_0^{\mathrm{A}}, \epsilon_0 \rangle$ is an ARTh over Rth_0, where $\mathit{Rth}_0^{\mathrm{A}} = \mathit{shared}(\mathit{Rth}_1^{\mathrm{A}}, \mathit{Rth}_2^{\mathrm{A}})$, $\mathit{Rth}_0 = \mathit{shared}(\mathit{Rth}_1, \mathit{Rth}_2)$, and $\epsilon_0(\alpha) = \epsilon_1(\alpha) = \epsilon_2(\alpha)$ for $\alpha \in S_0^{\mathrm{A}} \uplus \mathcal{T}_0^{\mathrm{A}} \uplus R_0^{\mathrm{A}}$. If $\mathit{ARth}_1 \bowtie \mathit{ARth}_2$, the result of *gluing* ARth_1 and ARth_2 is the ARTh $\mathit{ARth} = \mathit{ARth}_1 + \mathit{ARth}_2$ over $\mathit{Rth} = \mathit{Rth}_1 + \mathit{Rth}_2$ defined by:

$$\begin{array}{rl}
\text{(rth)} & \mathit{Rth}^{\mathrm{A}} = \mathit{Rth}_1^{\mathrm{A}} + \mathit{Rth}_2^{\mathrm{A}}; \\
\text{(eras0)} & \alpha \in S_0^{\mathrm{A}} \uplus X_0^{\mathrm{A}} \uplus \mathcal{OT}_0^{\mathrm{A}} \uplus R_0^{\mathrm{A}} \quad \Rightarrow \quad \epsilon(\alpha) = \epsilon_1(\alpha) = \epsilon_2(\alpha); \\
\text{(eras1)} & \alpha \in (S_1^{\mathrm{A}} \uplus X_1^{\mathrm{A}} \uplus \mathcal{OT}_1^{\mathrm{A}} \uplus R_1^{\mathrm{A}}) - (S_0^{\mathrm{A}} \uplus X_0^{\mathrm{A}} \uplus \mathcal{OT}_0^{\mathrm{A}} \uplus R_0^{\mathrm{A}}) \quad \Rightarrow \quad \epsilon(\alpha) = \epsilon_1(\alpha); \\
\text{(eras2)} & \alpha \in (S_2^{\mathrm{A}} \uplus X_2^{\mathrm{A}} \uplus \mathcal{OT}_2^{\mathrm{A}} \uplus R_2^{\mathrm{A}}) - (S_0^{\mathrm{A}} \uplus X_0^{\mathrm{A}} \uplus \mathcal{OT}_0^{\mathrm{A}} \uplus R_0^{\mathrm{A}}) \quad \Rightarrow \quad \epsilon(\alpha) = \epsilon_2(\alpha).
\end{array}$$

Gluing of ARThs is associative, commutative, and idempotent.

A *parameterized ARTh* (*pARTh*) over a pRTh *PRth* is a pair $\mathit{PARth} = \langle \mathit{ARth}_\pi, \mathit{ARth}_\beta \rangle$ where ARth_π and ARth_β are ARThs over Rth_π and Rth_β, respectively, and $\mathit{ARth}_\pi \hookrightarrow$

[†] Further flexibility is possible by relaxing the requirement that if $\vec{sq}$ are the premises of r then the open sequents of $\epsilon(r)$ must be exactly $\epsilon(\vec{sq})$, and just requiring instead that all the open sequents of $\epsilon(r)$ are among $\epsilon(\vec{sq})$. This makes it necessary to slightly extend the notion of derivation structure so to allow extension, duplication, and re-ordering of the open sequents; see Meseguer and Talcott (1998). Therefore, we do not consider it in this paper for brevity.

$ARth_\beta$. $ARth_\pi$ and $ARth_\beta$ are, respectively, the *parameter* and *body* of $PARth$. We have that $Rth^{\mathrm{A}}_\beta[Rth^{\mathrm{A}}_\pi]$ is a pRTh. We may write $ARth_\beta[ARth_\pi]$ instead of $\langle ARth_\pi, ARth_\beta\rangle$.

Let $ARth_1$ and $ARth_1$ be ARThs over RThs Rth_1 and Rth_2, respectively. Let $\rho : Rth_1 \to Rth_2$. A *replacement mapping* ρ^{A} from $ARth_1$ to $ARth_2$ over ρ, also written $\rho^{\mathrm{A}} : ARth_1 \to^\rho ARth_2$, is a replacement mapping $\rho^{\mathrm{A}} : Rth^{\mathrm{A}}_1 \to Rth^{\mathrm{A}}_2$ such that

$$\alpha \in S^{\mathrm{A}}_1 \uplus X^{\mathrm{A}}_1 \uplus \mathcal{OT}^{\mathrm{A}}_1 \uplus R^{\mathrm{A}}_1 \quad \Rightarrow \quad \epsilon_2(\rho^{\mathrm{A}}(\alpha)) = \rho(\epsilon_1(\alpha))$$

where we consider $\rho(\cdot) = \cdot$.

Let $PARth$ be a pARTh over a pRTh $PRth$. Let $ARth_0$ be an ARTh over an RTh Rth_0. Let $\rho : Rth_\pi \to Rth_0$, and let $\rho^{\mathrm{A}} : ARth_\pi \to^\rho ARth_0$. The result of *replacing* the parameter of $PARth$ with $ARth_0$ by ρ^{A} is the ARTh $ARth$ over $Rth = Rth_\beta[Rth_0/Rth_\pi]$ defined as follows:

(rth) $Rth^{\mathrm{A}} = Rth^{\mathrm{A}}_\beta[Rth^{\mathrm{A}}_0/Rth^{\mathrm{A}}_\pi]$;
(erasp) $\alpha \in S^{\mathrm{A}}_0 \uplus X^{\mathrm{A}}_0 \uplus \mathcal{OT}^{\mathrm{A}}_0 \uplus R^{\mathrm{A}}_0 \quad \Rightarrow \quad \epsilon(\alpha) = \epsilon_0(\alpha)$;
(erasb) $\alpha \in (S^{\mathrm{A}}_\beta \uplus X^{\mathrm{A}}_\beta \uplus \mathcal{OT}^{\mathrm{A}}_\beta \uplus R^{\mathrm{A}}_\beta) - (S^{\mathrm{A}}_\pi \uplus X^{\mathrm{A}}_\pi \uplus \mathcal{OT}^{\mathrm{A}}_\pi \uplus R^{\mathrm{A}}_\pi) \quad \Rightarrow \quad \epsilon(\rho^{\mathrm{A}}(\alpha)) = \rho(\epsilon_\beta(\alpha))$.

When ρ^{A} is clear from its context, we may write $ARth_\beta[ARth_0/ARth_\pi]$ to denote $ARth$.

The fact that an ARTh is essentially an RTh (accompanied by an erasing mapping) allows the composition mechanisms for RThs to be lifted to ARThs in a relatively straightforward way. The mechanisms apply to the non-annotated and annotated levels (i.e. Rth and Rth^{A}), and the erasing mappings must satisfy some "consistency" conditions insuring that a unique erasing mapping can be computed for the result. The notion of faithful inclusion requires erasing mappings to agree on the common part, as does the notion of composability. The notion of replacement mapping requires replacement to commute with erasing, i.e. replacing and then erasing must yield the same result as erasing and then replacing (note that different erasing and replacement mappings are used in the two cases). If such conditions are fulfilled, a unique erasing mapping is determined for the result of gluing and replacing a parameter.

2.3. CONTROL LAYER: TACTICS

At the control level, we can view the computations performed by a system as constructions and manipulations of annotated derivation structures. Such computations in fact perform the logical inferences, at the same time using and changing the control information encoded by annotations. These constructions and manipulations of annotated derivation structures can be "projected" to constructions and manipulations of non-annotated derivation structures at the logical level. While annotations in general narrow the search space of proofs (by constraining the applicability of logical inferences), an ARTh does not describe the actual strategies used to build and modify (annotated) derivation structures.

An interesting class of proof strategies can be expressed by means of tactics (although they certainly do not cover all the possible interesting strategies). Tactics, first introduced in LCF (Gordon *et al.*, 1979) and later adopted in many popular theorem provers such as NuPrl (Constable *et al.*, 1986) and Isabelle (Paulson, 1989), are an effective means to specify backward proof strategies in a modular fashion. Roughly speaking, a tactic is a function that takes an assertion and returns a (possibly empty) list of assertions as output: the validity of the input assertion follows from that of the output assertions.

In other words, a tactic reduces the goal of proving an assertion to (hopefully simpler) subgoals (if the list of subgoals is empty, the input assertion is just proved). Typically, a tactic will not always return a list of subgoals: it will fail when given some particular assertions as input.[†] Failure means that the tactic is unable to reduce the goal into subgoals, but other tactics can instead succeed on the same goal. There are usually primitive tactics corresponding to the backward application of rules of inference. More complex tactics can be built out of simpler ones by means of tacticals, i.e. higher-order functions operating on tactics. Tacticals can express strategies of application of their argument tactics: for example, apply a tactic, if it fails apply another one, otherwise (if it succeeds) apply yet another one on the result(s). Therefore, we can view tactics as strategies for construction of proof trees.

In the remainder of this section we present formal notions to describe strategies of construction of derivation structures by means of tactics. Our notion of tactic constitutes a slight extension of the notion found in the literature. First, our tactics are relational rather than just functional. Second, our tactics can explicitly manipulate instantiations and thus express richer strategies where instantiations are applied to derivation structures under construction. Third, we allow tactics to return different failure values (rather than just one), which can convey more information about the cause of the failure and thus allow more sophisticated choices of alternative tactics to be applied. Since there is no formal difference between an RTh containing logical information only and an RTh also containing control information (that constitutes an ARTh, together with an erasing mapping), for simplicity we present our formal definitions w.r.t. RThs rather than ARThs.

2.3.1. TACTIC SYSTEMS

A *tactic system* is a quadruple $Tsys = \langle \Sigma^{\mathrm{T}}, E^{\mathrm{T}}, T, F\rangle$. Σ^{T} and E^{T} are such that the quadruple $\langle \Sigma^{\mathrm{T}}, \emptyset, E^{\mathrm{T}}, \emptyset\rangle$ is a sequent system. This defines (for the tactic system), the S^{T}-typed set $\mathcal{T}^{\mathrm{T}}$ of terms, the S^{T}-typed set $\mathcal{E}^{\mathrm{T}}$ of equations, the consequence relation $\vdash\, \subseteq \mathcal{P}_\omega(\mathcal{E}^{\mathrm{T}}) \times \mathcal{E}^{\mathrm{T}}$, and the S^{T}-typed equivalence relation $\equiv\, \subseteq \mathcal{T}^{\mathrm{T}} \otimes \mathcal{T}^{\mathrm{T}}$. T and F are sets of sorts in S^{T}, i.e. $T \subseteq S^{\mathrm{T}}$ and $F \subseteq S^{\mathrm{T}}$. The T-typed set of *tactics* is $Tac = \widetilde{\mathcal{T}}^{\mathrm{T}}|_T$, and the F-typed set of *failures* is $Fail = \widetilde{\mathcal{T}}^{\mathrm{T}}|_F$, where $\widetilde{\mathcal{T}}^{\mathrm{T}} = \mathcal{T}^{\mathrm{T}}/_{\equiv}$.

There are obvious similarities (exploited in the formal definition above) between the notion of sequent system and that of tactic system. The difference is that a tactic system has no variables ($X = \emptyset$), and instead of a set Q for sequents it has two sets (not necessarily disjoint, although they often are) T and F. The purpose of a tactic system is to introduce a vocabulary of tactics and failures. Terms of sort in T denote tactics, and terms of sort in F denote failures. Typically, a tactic system contains constants of sort in T (i.e. operations of arity $\rightarrow s$ with $s \in T$) that are understood as "names" for tactics. Operations with a result sort in T and some of the argument sorts also in T play the role of tacticals, because they build a new tactic (the result) from other tactics (the arguments). Equations are generally used to equate a constant (i.e. a named tactic) to a term involving operations applied to constants (i.e. a tactic built out of tacticals): this corresponds to tactic definitions found in the literature (in the setting of functional programming languages), because tactics are equivalence classes of terms,

[†] In the theorem provers mentioned above, failure concretely happens as exception raising. Mathematically, we can think of the codomain of a tactic as consisting of lists of assertions plus a distinguished value denoting failure.

and therefore the two terms denote the same tactic. Often, failures just consist of some disjoint constants, but the notion of a tactic system allows more sophisticated failures. The reason why a tactic system does not include variables and instantiations is that its goal is just to define a vocabulary of tactics and failures, for which ground terms suffice.

2.3.2. TACTIC THEORIES

A *tactic theory* (*TTh*) over an RTh *Rth* is a pair $Tth = \langle Tsys, TR\rangle$, where *Tsys* is a tactic system and $TR \subseteq Tevl^* \times Tevl$, where $Tevl = Tac \times Sq \times (Fail \uplus (\Delta \times I))$. The elements of *Tevl* are called *tactic evaluations*, and if $\langle \tau, sq, res\rangle \in Tevl$ we may write $\tau \lhd sq \rightsquigarrow res$ instead of $\langle \tau, sq, res\rangle$. The elements of *TR* are called *tactic rules*, and if $\langle [\tau_1 \lhd sq_1 \rightsquigarrow res_1, \ldots, \tau_n \lhd sq_n \rightsquigarrow res_n], \tau \lhd sq \rightsquigarrow res\rangle \in TR$ we may write

$$\frac{\tau_1 \lhd sq_1 \rightsquigarrow res_1 \quad \cdots \quad \tau_n \lhd sq_n \rightsquigarrow res_n}{\tau \lhd sq \rightsquigarrow res}$$

instead of $\langle [\tau_1 \lhd sq_1 \rightsquigarrow res_1, \ldots, \tau_n \lhd sq_n \rightsquigarrow res_n], \tau \lhd sq \rightsquigarrow res\rangle$, and we may write terms (of sorts in T and F) instead of tactics and failures (i.e. instead of the terms' equivalence classes). The *evaluation relation* $(\_ \lhd \_ \rightsquigarrow^* \_) \subseteq Tac \times Sq \times (Fail \uplus (\Delta \times I))$ is the smallest one such that

$$\begin{array}{l} \langle [\tau_1 \lhd sq_1 \rightsquigarrow res_1, \ldots, \tau_n \lhd sq_n \rightsquigarrow res_n], \tau \lhd sq \rightsquigarrow res\rangle \in TR \ \wedge \\ (1 \leq i \leq n \quad \Rightarrow \quad \tau_i \lhd sq_i \rightsquigarrow^* res_i) \quad \Rightarrow \tau \lhd sq \rightsquigarrow^* res. \end{array}$$

It is required that if $\tau \lhd sq \rightsquigarrow^* \langle \delta, \iota\rangle$ then $sq[\iota]$ is the conclusion of δ.

A TTh consists of a tactic system, which describes a vocabulary of tactics and failures, and of tactic rules, which describe how tactics "work". $\tau \lhd sq \rightsquigarrow^* res$ expresses that the application of tactic τ to sequent *sq* yields *res* as result. *res* can be either a failure $fail \in Fail$, or a pair $\langle \delta, \iota\rangle \in \Delta \times I$. The first case models that τ failed on *sq*, and *fail* gives information about the reason for that. The second case models that τ succeeded on *sq*, yielding a derivation structure δ and an instantiation ι. As required by definition, the conclusion of δ must be $sq[\iota]$. If $\iota = \mathtt{idi}$, this just means that δ has exactly *sq* as conclusion. In other words, the tactic has reduced the problem of proving *sq* (goal) to the problem of proving the (zero or more) open sequents of δ (subgoals), and δ constitutes a logical justification for that. This closely resembles the working of tactics as found in the literature. If $\iota \neq \mathtt{idi}$, in general $sq[\iota]$ may differ from *sq*. This allows us to conveniently specify some reasoning strategies (commonly employed by practical systems) where rather than proving a particular assertion (given by the user or generated during computation) the system proves a particular instance of a "schematic" assertion. For example, a first-order theorem prover may eliminate an existential quantifier by replacing the variable with a "meta-variable" (i.e. some data structure that denotes a place-holder for a term), and later replace such meta-variable with a term (because the term is computed as part of the subsequent reasoning). In our framework, the place-holder would be a variable in X, and the replacement would be an instantiation ι, returned by some tactic and propagated to the derivation structure under construction by suitable tactic rules.

Tactic rules describe which results may be returned by applying tactics to sequents; they specify the operational semantics of tactics. Note that non-determinism is allowed (i.e. relational tactics), with determinism (i.e. functional tactics) as a special case. Tactic rules of the form $\langle [\,], \tau \lhd sq \rightsquigarrow res\rangle$ directly express that applying τ to *sq* yields *res*. Tactic rules of the form $\langle [\tau_1 \lhd sq_1 \rightsquigarrow res_1, \ldots, \tau_n \lhd sq_n \rightsquigarrow res_n] \tau \lhd sq \rightsquigarrow res\rangle$ with $n \neq 0$

express that if applying τ_i to sq_i yields res_i for $1 \leq i \leq n$, then applying τ to sq yields res. This second form allows us to specify how tactics work together: they typically describe how complex tactic applications are decomposed into simpler ones, and how results are combined. The evaluation relation is basically the closure of all the tactic rules. Note that the notion of TTh includes no explicit notion of a primitive tactic (i.e. a tactic corresponding to the backward application of a single rule of inference) and nor of a tactical (i.e. an operator to combine tactics). These concepts can indeed be conveniently modeled in our framework. A primitive tactic is typically described by means of tactic rules of the form $\langle [\,], \tau \lhd sq \rightsquigarrow res\rangle$, and we can have different rules corresponding to different (backward) applications of the same inference rule. Tacticals, as already mentioned, can be described by suitable operations in the signature of a tactic system, and suitable tactic rules that express how the argument tactics of a tactical are combined together.

2.3.3. GLUING AND PARAMETERIZATION

A TTh Tth_0 over an RTh Rth_0 is *faithfully included* in a TTh Tth_1 over an RTh Rth_1, also written $\mathit{Tth}_0 \hookrightarrow \mathit{Tth}_1$, iff $\mathit{Rth}_0 \hookrightarrow \mathit{Rth}_1$, $S_0^{\mathrm{T}} \subseteq S_1^{\mathrm{T}}$, $O_1^{\mathrm{T}}|_{(S_1^{\mathrm{T}})^* \times S_0^{\mathrm{T}}} = O_0^{\mathrm{T}}$, $E_1^{\mathrm{T}}|_{S_0^{\mathrm{T}}} = E_0^{\mathrm{T}}$, $T_1 \cap S_0^{\mathrm{T}} = T_0$, $F_1 \cap S_0^{\mathrm{T}} = F_0$, and $\mathit{TR}_0 \subseteq \mathit{TR}_1$. If $\mathit{Tth}_0 \hookrightarrow \mathit{Tth}_1$ then $\mathcal{T}_1^{\mathrm{T}}|_{S_0^{\mathrm{T}}} = \mathcal{T}_0^{\mathrm{T}}$, $\widetilde{\mathcal{T}}_1^{\mathrm{T}}|_{S_0^{\mathrm{T}}} = \widetilde{\mathcal{T}}_0^{\mathrm{T}}$, $\mathit{Tac}_1|_{T_0} = \mathit{Tac}_0$, and $\mathit{Fail}_1|_{F_0} = \mathit{Fail}_0$. $\hookrightarrow$, as a binary relation over TThs, is a partial order.

Let Tth_1 and Tth_2 be TThs over RThs Rth_1 and Rth_2, respectively. If $\mathit{Rth}_1 \bowtie \mathit{Rth}_2$, let $\mathit{shared}(\mathit{Tth}_1, \mathit{Tth}_2) = \mathit{Tth}_0$, where $S_0^{\mathrm{T}} = S_1^{\mathrm{T}} \cap S_2^{\mathrm{T}}$, $O_0^{\mathrm{T}} = O_1^{\mathrm{T}} \cap O_2^{\mathrm{T}}$, $E_0^{\mathrm{T}} = E_1^{\mathrm{T}} \cap E_2^{\mathrm{T}}$, $T_0 = T_1 \cap T_2$, $F_0 = F_1 \cap F_2$, and $\mathit{TR}_0 = \mathit{TR}_1 \cap \mathit{TR}_2$. If Tth_0 is defined then it is a TTh over $\mathit{Rth}_0 = \mathit{shared}(\mathit{Rth}_1, \mathit{Rth}_2)$. Tth_1 and Tth_2 are *glueable*, also written $\mathit{Tth}_1 \bowtie \mathit{Tth}_2$, iff $\mathit{Rth}_1 \bowtie \mathit{Rth}_2$, Tth_0 is defined, $\mathit{Tth}_0 \hookrightarrow \mathit{Tth}_1$, $\mathit{Tth}_0 \hookrightarrow \mathit{Tth}_2$. If $\mathit{Tth}_1 \bowtie \mathit{Tth}_2$, the result of *gluing* Tth_1 and Tth_2 is the TTh $\mathit{Tth} = \mathit{Tth}_1 + \mathit{Tth}_2$ over RTh $\mathit{Rth} = \mathit{Rth}_1 + \mathit{Rth}_2$ defined by $S^{\mathrm{T}} = S_1^{\mathrm{T}} \cup S_2^{\mathrm{T}}$, $O^{\mathrm{T}} = O_1^{\mathrm{T}} \cup O_2^{\mathrm{T}}$, $E^{\mathrm{T}} = E_1^{\mathrm{T}} \cup E_2^{\mathrm{T}}$, $T = T_1 \cup T_2$, $F = F_1 \cup F_2$, and $\mathit{TR} = \mathit{TR}_1 \cup \mathit{TR}_2$. We have $\mathit{Tth}_1 \hookrightarrow \mathit{Tth}$, $\mathit{Tth}_2 \hookrightarrow \mathit{Tth}$, $\mathcal{T}_0^{\mathrm{T}} = \mathcal{T}_1^{\mathrm{T}} \cap \mathcal{T}_2^{\mathrm{T}}$, $\mathit{Tac}_0 = \mathit{Tac}_1 \cap \mathit{Tac}_2$, $\mathit{Fail}_0 = \mathit{Fail}_1 \cap \mathit{Fail}_2$, $\mathcal{T}^{\mathrm{T}} = \mathcal{T}_1^{\mathrm{T}} \cup \mathcal{T}_2^{\mathrm{T}}$, $\mathit{Tac} = \mathit{Tac}_1 \cup \mathit{Tac}_2$, and $\mathit{Fail} = \mathit{Fail}_1 \cup \mathit{Fail}_2$. Gluing of TThs is associative, commutative, and idempotent.

A *parameterized TTh* (*pTTh*) over a pRTh PRth is a pair $\mathit{PTth} = \langle \mathit{Tth}_\pi, \mathit{Tth}_\beta \rangle$, where Tth_π and Tth_β are TThs over Rth_π and Rth_π, respectively, and $\mathit{Tth}_\pi \hookrightarrow \mathit{Tth}_\beta$. Tth_π and Tth_β are, respectively, the *parameter* and *body* of PTth. We may write $\mathit{Tth}_\beta[\mathit{Tth}_\pi]$ instead of $\langle \mathit{Tth}_\pi, \mathit{Tth}_\beta \rangle$.

Let Tth_1 and Tth_2 be TThs over RThs Rth_1 and Rth_2, respectively. Let $\rho : \mathit{Rth}_1 \to \mathit{Rth}_2$. A *replacement mapping* ρ^{T} from Tth_1 to Tth_2 over ρ, also written $\rho^{\mathrm{T}} : \mathit{Tth}_1 \to^{\rho} \mathit{Tth}_2$, is a pair $\rho^{\mathrm{T}} = \langle \rho_{\mathrm{S}}^{\mathrm{T}}, \rho_{\mathrm{O}}^{\mathrm{T}} \rangle$, where:

(srt) $\rho_{\mathrm{S}}^{\mathrm{T}} : S_1^{\mathrm{T}} \to S_2^{\mathrm{T}}$;
(op) $\rho_{\mathrm{O}}^{\mathrm{T}} : O_1^{\mathrm{T}} \to_{\rho_{\mathrm{S}}^{\mathrm{T}}} O_2^{\mathrm{T}}$;
(eq) if $(t_1 = t_2) \in E_1^{\mathrm{T}}$ then $E_2^{\mathrm{T}} \vdash (\rho_{\mathrm{T}}^{\mathrm{T}}(t_1) = \rho_{\mathrm{T}}^{\mathrm{T}}(t_2))$, where $\rho_{\mathrm{T}}^{\mathrm{T}} : \mathcal{T}_1^{\mathrm{T}} \to_{\rho_{\mathrm{S}}^{\mathrm{T}}} \mathcal{T}_2^{\mathrm{T}}$ is defined by $\rho_{\mathrm{T}}^{\mathrm{T}}(o(t_1, \ldots, t_n)) = \rho_{\mathrm{O}}^{\mathrm{T}}(o)(\rho_{\mathrm{T}}^{\mathrm{T}}(t_1), \ldots, \rho_{\mathrm{T}}^{\mathrm{T}}(t_n))$;
(tcsrt) $s \in T_1 \quad \Rightarrow \quad \rho_{\mathrm{S}}^{\mathrm{T}}(s) \in T_2$;
(flsrt) $s \in F_1 \quad \Rightarrow \quad \rho_{\mathrm{S}}^{\mathrm{T}}(s) \in F_2$;
(trul) if $\langle [\tau_1 \lhd sq_1 \rightsquigarrow res_1, \ldots, \tau_n \lhd sq_n \rightsquigarrow res_n], \tau \lhd sq \rightsquigarrow res \rangle \in \mathit{TR}_1$ then $\langle [\rho_{\mathrm{E}}^{\mathrm{T}}(\tau_1 \lhd sq_1 \rightsquigarrow res_1), \ldots, \rho_{\mathrm{E}}^{\mathrm{T}}(\tau_n \lhd sq_n \rightsquigarrow res_n)], \rho_{\mathrm{E}}^{\mathrm{T}}(\tau \lhd sq \rightsquigarrow res) \rangle \in \mathit{TR}_2$, where

$\rho^{\mathrm{T}}_{\mathrm{E}} : \mathit{Tevl}_1 \to \mathit{Tevl}_2$ is defined by $\rho^{\mathrm{T}}_{\mathrm{E}}(\tau \triangleleft \mathit{sq} \leadsto \mathit{res}) = \rho^{\mathrm{T}}_{\mathrm{TF}}(\tau) \triangleleft \rho(\mathit{sq}) \leadsto \rho^{\mathrm{T}}_{\mathrm{R}}(\mathit{res})$, where $\rho^{\mathrm{T}}_{\mathrm{TF}} : \mathit{Tac}_1 \cup \mathit{Fail}_1 \to_{\rho^{\mathrm{T}}_{\mathrm{S}}} \mathit{Tac}_2 \cup \mathit{Fail}_2$ is defined by $\rho^{\mathrm{T}}_{\mathrm{TF}}([\![t]\!]) = [\![\rho^{\mathrm{T}}_{\mathrm{T}}(t)]\!]$ for $t \in \mathcal{T}_1^{\mathrm{T}}|_{T_1 \cup F_1}$, and $\rho^{\mathrm{T}}_{\mathrm{R}} : \mathit{Fail}_1 \uplus (\Delta_1 \times I_1) \to_{(\rho^{\mathrm{T}}_{\mathrm{TF}} \uplus \rho)} \mathit{Fail}_2 \uplus (\Delta_2 \times I_2)$ is defined by $\rho^{\mathrm{T}}_{\mathrm{R}}(\mathit{fail}) = \rho^{\mathrm{T}}_{\mathrm{TF}}(\mathit{fail})$ for $\mathit{fail} \in \mathit{Fail}_1$ and $\rho^{\mathrm{T}}_{\mathrm{R}}(\langle \delta, \iota \rangle) = \langle \rho(\delta), \rho(\iota) \rangle$ for $\langle \delta, \iota \rangle \in \Delta_1 \times I_1$.

We may drop the indices and just write ρ^{T} instead of $\rho^{\mathrm{T}}_{\mathrm{S}}$, $\rho^{\mathrm{T}}_{\mathrm{O}}$, etc.

Let PTth be a pTTh over a pRTh PRth. Let Tth_0 be a TTh over an RTh Rth_0. Let $\rho : \mathit{Rth}_\pi \to \mathit{Rth}_0$, and let $\rho^{\mathrm{T}} : \mathit{Tth}_\pi \to^\rho \mathit{Tth}_0$. The result of *replacing* the parameter of PTth with Tth_0 by ρ^{T} is the TTh Tth over $\mathit{Rth}_\beta[\mathit{Rth}_0/\mathit{Rth}_\pi]$ defined as follows, where we also lift ρ^{T} to Tth_β, $\rho^{\mathrm{T}} : \mathit{Tth}_\beta \to^\rho \mathit{Tth}_0$:

$$
\begin{array}{rl}
\text{(srt)} & S^{\mathrm{T}} = S^{\mathrm{T}}_0 \uplus (S^{\mathrm{T}}_\beta - S^{\mathrm{T}}_\pi); \\
\text{(srplc)} & s \in S^{\mathrm{T}}_\beta - S^{\mathrm{T}}_\pi \quad \Rightarrow \quad \rho^{\mathrm{T}}(s) = s; \\
\text{(op)} & O^{\mathrm{T}} = O^{\mathrm{T}}_0 \uplus \{o{:}\rho^{\mathrm{T}}(\vec{s} \to s) \mid o{:}\vec{s} \to s \in O^{\mathrm{T}}_\beta - O^{\mathrm{T}}_\pi\}; \\
\text{(orplc)} & o \in O^{\mathrm{T}}_\beta - O^{\mathrm{T}}_\pi \quad \Rightarrow \quad \rho^{\mathrm{T}}(o) = o; \\
\text{(eq)} & E^{\mathrm{T}} = E^{\mathrm{T}}_0 \uplus \{(\rho^{\mathrm{T}}(t_1) = \rho^{\mathrm{T}}(t_2)){:}s \mid (t_1 = t_2){:}s \in E^{\mathrm{T}}_\beta - E^{\mathrm{T}}_\pi\}; \\
\text{(tcsrt)} & T = T_0 \uplus (T_\beta - T_\pi); \\
\text{(flsrt)} & F = F_0 \uplus (F_\beta - F_\pi); \\
\text{(trul)} & \mathit{TR} = \mathit{TR}_0 \uplus \{\rho^{\mathrm{T}}(\mathit{tr}) \mid \mathit{tr} \in \mathit{TR}_\beta - \mathit{TR}_\pi\}.
\end{array}
$$

We have $\mathit{Tth}_0 \hookrightarrow \mathit{Tth}$. When ρ^{T} is clear from its context, we may write $\mathit{Tth}_\beta[\mathit{Tth}_0/\mathit{Tth}_\pi]$ to denote Tth.

Because of the structural similarity between TThs and RThs, the composition mechanisms for TThs are very analogous to those for RThs. Note that since every TTh is associated with an RTh, the composition mechanisms for TThs involve the underlying RThs. As already mentioned, for simplicity we have presented the formal notions for TThs with reference to RThs, rather than ARThs. However, an OMRS specification contains (1) an RTh Rth, (2) an ARTh ARth over Rth, and (3) a TTh Tth over $\mathit{Rth}^{\mathrm{A}}$. Rth specifies the logic, while ARth and Tth constitute the control. These three formal objects are organized in layers: Rth is at the bottom, ARth is over Rth, and Tth is over ARth. When composition mechanisms (gluing and parameterization) are used to compose OMRS specifications together, composition takes place at each layer.

3. Constraint Contextual Rewriting as a Case Study

Let us consider the problem of simplifying the clause

$$(\log(x \cdot y) > 0) \ \vee \ (2 \cdot \log y < 0) \ \vee \ (\log x \leq 3) \ \vee \ (x \leq 0) \ \vee \ (y < x) \tag{2}$$

using the following conditional rewrite rule:

$$U > 0 \wedge V > 0 \to \log(U \cdot V) = \log U + \log V \tag{3}$$

($+$, $<$ and $\leq$ have their usual arithmetic interpretation). The key idea of Constraint Contextual Rewriting is that while rewriting a literal (the *focus literal*) the negation of the remaining literals in the clause (the *context*) can be assumed true. For instance, let $\log(x \cdot y) > 0$ be the focus literal in (refeq:ex), then the context is $\{2 \cdot \log y \not< 0, \log x \not\leq 3, x \not\leq 0, y \not< x\}$. Application of (3) to the focus literal yields $\log x + \log y > 0$, under the

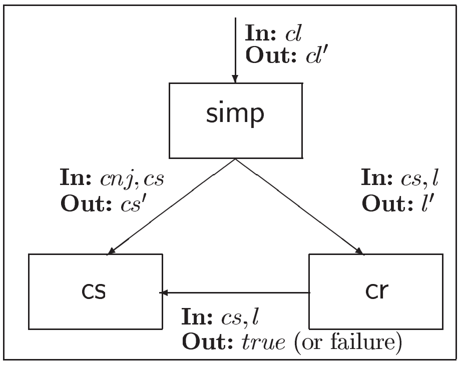

Figure 1. CCR(X): the integration schema.

proviso that the instantiated conditions, namely $x > 0$ and $y > 0$, can be established. Indeed, both conditions follow from the context by simple arithmetic reasoning. Moreover, the rewritten version of the focus literal, i.e. $\log x + \log y > 0$, follows from the context by simple arithmetic reasoning and this allows us to rewrite the focus literal (and hence the whole clause) to *true*. In general, in order to establish whether a given literal l follows from a context c it is possible to invoke a decision procedure for some decidable fragment of the background theory to check the satisfiability of the set of literals obtained by adding the negation of l to c. In our example a decision procedure for linear arithmetic would suffice. The key observation here is that the reasoning involved can be mechanized by combining traditional conditional rewriting with a decision procedure. Furthermore the pattern of interaction between rewriting and the decision procedure does not depend on the theory decided by the decision procedure. The notation CCR(X) (by analogy with CLP(X) used to denote the Constraint Logic Programming paradigm (Jaffar and Maher, 1994)) emphasizes the independence of Constraint Contextual Rewriting from the theory (X) decided by the decision procedure. The traditional notion of contextual rewriting (Zhang, 1995) is an instance of CCR(X) whereby X is instantiated to a decision procedure for ground equalities and new forms of contextual rewriting can be obtained by instantiating X to decision procedures for different decidable theories.[†]

The interplay between the simplifier (simp), the rewrite engine (cr), and the decision procedure (cs) in CCR(X) is depicted in Figure 1. simp takes a clause (*cl*) and returns a simplified clause (*cl′*). cr performs conditional rewriting on the input literal by using *cs* as rewriting context and returns a rewritten literal. cs can be invoked by both simp and cr. In the first case, cs takes a conjunction of literals *cnj* and a context *cs* as input and returns a new context *cs′* obtained by extending *cs* with the literals in *cnj*. In the second case, cs takes a context *cs* and a literal l and returns *true* whenever it is able to determine that l is entailed by *cs*; otherwise, it reports failure.

3.1. AN OMRS SPECIFICATION OF CONSTRAINT CONTEXTUAL REWRITING

CCR(X) can be conveniently specified in the OMRS framework by exploiting the modularity as well as the distinction between the logic and the control layers. The logic

[†] Armando and Ranise (1998a) show that the integration schemas employed in the simplifiers of NQTHM (Boyer and Moore, 1988) and Tecton (Kapur and Nie, 1994) are both instances of CCR(X).

layer is specified by a pRTh Rth_{ccr} (Section 3.1.1) obtained by (i) specifying the logic of the simplifier, of the rewrite engine, and of the decision procedure by means of suitably defined RThs (Rth_{simp}, Rth_{cr}, and Rth_{cs}, respectively), (ii) by gluing together Rth_{simp}, Rth_{cr}, and Rth_{cs}, and then (iii) by making the resulting RTh parametric in Rth_{cs}. The control layer is specified analogously by defining a pARTh $ARth_{\mathsf{ccr}}$ (Section 3.1.2) and a pTTh Tth_{ccr} (Section 3.1.3) along the same lines.

3.1.1. A REASONING THEORY FOR CCR(X)

SIMPLIFICATION

S_{simp} consists of the sorts TERM (terms), LIT (literals), CL (clauses), CNJ (conjunctions), and SEQ (sequents). $Q_{\mathsf{simp}} = \{\mathrm{SEQ}\}$. O_{simp} contains the symbols $\mathit{false} : [\,] \to \mathrm{LIT}$, $\mathit{true} : [\,] \to \mathrm{LIT}$, and $\approx : [\mathrm{TERM},\ \mathrm{TERM}] \to \mathrm{LIT}$ for truth, falsity, and equality, respectively. $\neg$ is an operation of arities $[\ \mathrm{LIT}] \to \mathrm{LIT}$ and $[\mathrm{CL}] \to\ \mathrm{CNJ}$.[†] $\vee$ ($\wedge$) is an associative and commutative operation of arities $[s_1, s_2] \to \mathrm{CL}$ ($[s_1, s_2] \to \mathrm{CNJ}$) for all $s_1, s_2 \in \{\mathrm{LIT}, \mathrm{CL}\}$ ($s_1, s_2 \in \{\mathrm{LIT}, \mathrm{CNJ}\}$, resp.).[‡] E_{simp} is such that $\neg \mathit{true} \equiv \mathit{false}$, $\neg \mathit{false} \equiv \mathit{true}$, $\neg\neg l \equiv l$ for all $l \in \mathcal{T}|_{\{\mathrm{LIT}\}}$, $(c \vee \mathit{false}) \equiv c$, $(c \vee \mathit{true}) \equiv \mathit{true}$ for all $c \in \mathcal{T}|_{\{\mathrm{LIT},\mathrm{CL}\}}$, $(c \wedge \mathit{false}) \equiv \mathit{false}$, $(c \wedge \mathit{true}) \equiv c$ for all $c \in \mathcal{T}|_{\{\mathrm{LIT},\mathrm{CNJ}\}}$, and $\neg(l_1 \vee l_2) \equiv (\neg l_1 \wedge \neg l_2)$ for all $l_1, l_2 \in \mathcal{T}|_{\{\mathrm{LIT}\}}$. O_{simp} contains also the operations $\_ \xrightarrow[\mathsf{simp}]{} \_ : [\mathrm{CL}, \mathrm{CL}] \to \mathrm{SEQ}$, $\_ :: \_ \xrightarrow[\mathsf{cr}]{} \_ : [\mathrm{CNJ},\ \mathrm{LIT}, \mathrm{LIT}] \to \mathrm{SEQ}$, $\_ :: \_ \xrightarrow[\mathsf{cs}]{} \_ : [\mathrm{CNJ}, \mathrm{CNJ}, \mathrm{CNJ}] \to \mathrm{SEQ}$, and $\mathsf{cs\text{-}init} : [\mathrm{CNJ}] \to \mathrm{SEQ}$. Intuitively, $cl \xrightarrow[\mathsf{simp}]{} cl'$ asserts that cl' is the result of simplifying cl; $cs :: l \xrightarrow[\mathsf{cr}]{} l'$ asserts that l' is the result of rewriting l' using cs (also called *constraint store*) as context; $cnj :: cs \xrightarrow[\mathsf{cs}]{} cs'$ asserts that cs' is the result of extending cs with the literals in cnj; finally, $\mathsf{cs\text{-}init}(cs)$ asserts that cs is the "empty" constraint store.

A sequent of the form $c :: e \xrightarrow[\lambda]{} e'$ represents a (contextual) reduction relation, i.e. it asserts that e' is the result of reducing e to e' in context c. In what follows, symbols beginning with a question mark denote variables of the sequent system under consideration. The reflexivity and transitivity properties of such relations are formalized by the following rules in R_{simp}:

$$\frac{}{?c :: ?e \xrightarrow[\lambda]{} ?e}\ \texttt{refl} \qquad \frac{?c :: ?e \xrightarrow[\lambda]{} ?e' \qquad ?c :: ?e' \xrightarrow[\lambda]{} ?e''}{?c :: ?e \xrightarrow[\lambda]{} ?e''}\ \texttt{trans}$$

where $?c$, $?e$, and $?e'$ are variables of appropriate sort. R_{simp} contains analogous rules for the sequents $e \xrightarrow[\mathsf{simp}]{} e'$ and the rule:

$$\frac{\mathsf{cs\text{-}init}(?cs_0) \qquad \neg ?cl :: ?cs_0 \xrightarrow[\mathsf{cs}]{} ?cs \qquad ?cs :: ?l \xrightarrow[\mathsf{cr}]{} ?l'}{?cl \vee ?l \xrightarrow[\mathsf{simp}]{} ?cl \vee ?l'}\ \texttt{cl-simp}$$

which states that a literal $?l$ in a clause $?cl \vee ?l$ can be rewritten to $?l'$ provided that $?l'$

[†] To simplify the presentation we assign multiple arities to operations, with the convention that when we say that *an operation o has arities $aty_1, aty_2, \ldots, aty_m$* (for $m > 1$) we mean that there exist operations $o_1 : aty_1$, $o_2 : aty_2, \ldots,$ and $o_m : aty_m$.

[‡] We say that a binary operation $\star : [s, s] \to s$ is *associative* iff $e_1 \star (e_2 \star e_3) \equiv (e_1 \star e_2) \star e_3$ and it is *commutative* iff $e_1 \star e_2 \equiv e_2 \star e_1$, for all $e_1, e_2, e_3 \in \mathcal{T}|_{\{s\}}$.

is the result of rewriting $?l$ in the context obtained by extending the empty constraint store $?cs_0$ with the negation of the literals in $?cl$.

Finally, let $\alpha \in \mathcal{T}|_{\{\textsc{cl},\textsc{cnj}\}}$ and $\Gamma \subseteq \mathcal{T}_{\{\textsc{cl},\textsc{cnj}\}}$, then α is a *logical consequence* of Γ iff $\Gamma \models \alpha$, where $\models$ denotes the entailment relation in classical logic. A *theory* is a subset of $\mathcal{T}|_{\{\textsc{cl},\textsc{cnj},\ \textsc{lit}\}}$ closed under logical consequence. If T is a theory, then $\Gamma \models_T \alpha$ abbreviates $T \cup \Gamma \models \alpha$ and $\Gamma \models_T \alpha \leftrightarrow \beta$ abbreviates the conjunction of $\Gamma \cup \{\alpha\} \models_T \beta$ and $\Gamma \cup \{\beta\} \models_T \alpha$. α is T-*consistent* iff there exists a model of $T \cup \{\alpha\}$, and T-*inconsistent* otherwise. α is T-*valid* iff α is a logical consequence of T or, equivalently, iff $\alpha \in T$. In what follows, T_c and T_j are theories such that $T_c \subseteq T_j$ and $\mathcal{R}$ is a finite set of T_j-valid clauses.

REWRITING

The sequent system of Rth_{cr} can be obtained from that of Rth_{simp} by removing the sort CL and the operations $\vee$, $\xrightarrow[\mathsf{simp}]{}$, and $\mathsf{cs\text{-}init}$ and adding the sorts POS (positions) and SUBST (substitutions), the operations $\_{}_{|\_}$ of arities $[\textsc{term}, \textsc{pos}] \to \textsc{term}$ and $[\textsc{lit}, \textsc{pos}] \to \textsc{term}$, $\_[\_]\_$ of arities $[\textsc{term}, \textsc{term},\ \textsc{pos}] \to \textsc{term}$ and $[\textsc{lit}, \textsc{term}, \textsc{pos}] \to \textsc{term}$, $\_ \lhd \_$ of arities $[\textsc{term}, \textsc{subst}] \to \textsc{term}$, $[\textsc{lit}, \textsc{subst}] \to \textsc{lit}$, and $\mathsf{cs\text{-}unsat}$ of arity $[\textsc{cnj}] \to \textsc{seq}$. Intuitively, $s_{|u}$ denotes the sub-term at position u in s, $s[t]_u$ denotes the results of replacing the sub-term at position u in s with the term t, $s \lhd \sigma$ denotes the result of applying the substitution σ to s, and $\mathsf{cs\text{-}unsat}(cs)$ asserts the unsatisfiability of cs. E_{cr} is obtained from E_{simp} by removing the axioms for $\vee$ and adding a suitable axiomatization for $\_{}_{|\_}$, $\_[\_]\_$, and $\_ \lhd \_$.

R_{cr} contains the rules `refl` and `trans` for all sequents of the form $c :: e \xrightarrow[\lambda]{} e'$ in Rth_{cr}. The following rule formalizes the interface between the rewrite engine and the decision procedure:

$$\frac{\neg ?l :: ?cs \xrightarrow[\mathsf{cs}]{} ?cs' \qquad \mathsf{cs\text{-}unsat}(?cs')}{?cs :: ?l \xrightarrow[\mathsf{cr}]{} true}\ \mathtt{cxt\text{-}entails}$$

by stating that a literal $?l$ can be rewritten to *true* in context $?cs$ provided that the constraint store obtained by extending $?cs$ with the negation of $?l$ is unsatisfiable. Finally the rules

$$\frac{?cs :: cnj \lhd ?\sigma \xrightarrow[\mathsf{cr}]{} true}{?cs :: ?l[s \lhd ?\sigma]_{?u} \xrightarrow[\mathsf{cr}]{} ?l[t \lhd ?\sigma]_{?u}}\ \mathtt{crew}$$

for all $(cnj \to s \approx t) \in \mathcal{R}$,[†] formalize conditional rewriting by asserting that a sub-term $s \lhd ?\sigma$ at position $?u$ in a literal $?l$ can be replaced with $t \lhd ?\sigma$ in context $?cs$ provided that there exists a clause $cnj \to s \approx t$ in $\mathcal{R}$ and the instantiated conditions can be recursively rewritten to *true*.

CONSTRAINT SOLVING

The sequent system for Rth_{cs} can be obtained from that for Rth_{simp} by removing the sort CL, the operations $\vee$, $\xrightarrow[\mathsf{simp}]{}$, $::\ \xrightarrow[\mathsf{cr}]{}$ and then by adding the operation $\mathsf{cs\text{-}unsat}$

[†] $cnj \to (l \approx r)$ abbreviates the clause $\neg cnj \vee (l \approx r)$, where cnj is a conjunction, and l and r are terms. We call *conditions* the literals in cnj.

of arity $[\textsc{cnj}] \to \textsc{seq}$. R_{cs} contains the rules `refl` and `trans` for all sequents of the form $c :: e \underset{\lambda}{\longrightarrow} e'$ as well as rules of the form `cs-init` : $[\,] \to \mathsf{cs\text{-}init}(cs)$, `cs-unsat` : $[\,] \to \mathsf{cs\text{-}unsat}(cs)$, and `cs-simp` : $[\,] \to cnj :: cs \underset{\mathsf{cs}}{\longrightarrow} cs'$. The rules are such that if `cs-init` : $[\,] \to \mathsf{cs\text{-}init}(cs) \in R_{\mathsf{cs}}$, then cs is T_c-valid; if `cs-unsat` : $[\,] \to \mathsf{cs\text{-}unsat}(cs) \in R_{\mathsf{cs}}$, then cs is T_c-inconsistent; and finally if `cs-simp` : $[\,] \to cnj :: cs \underset{\mathsf{cs}}{\longrightarrow} cs' \in R_{\mathsf{cs}}$, then $\{cnj, cs\} \models_{T_c} cs'$.

GLUING AND PARAMETERIZATION

A pRTh for CCR(X) in given by:

$$Rth_{\mathsf{ccr}} := (Rth_{\mathsf{simp}} + Rth_{\mathsf{cr}} + Rth_{\mathsf{cs}})[Rth_{\mathsf{cs}}].$$

It can be readily verified that Rth_{simp}, Rth_{cr}, and Rth_{cs} are glueable and that Rth_{cs} is faithfully included in $Rth_{\mathsf{simp}} + Rth_{\mathsf{cr}} + Rth_{\mathsf{cs}}$.

We are now in the position to state and prove the soundness of the simplification schema obtained by putting together the various reasoning theories as specified above. The soundness of CCR(X) within the OMRS framework is formally stated as follows.

Proposition 3.1. (Soundness of Rth_{ccr})
If there exists a derivation $\delta : \langle [\,], cl \underset{\mathsf{simp}}{\longrightarrow} cl' \rangle$ in Rth_{ccr}, then $\models_{T_j} cl \leftrightarrow cl'$.

The proof of this proposition follows from the soundness of Rth_{cr} and Rth_{cs}. The soundness of these two *Rth*s can be proved by induction on the structure of the derivations. The interested reader is referred to Armando and Ranise (1998a, 2000) for more details. Here, it is important to notice how the OMRS framework provides the necessary concepts which allow for a formal specification and proof of important properties such as the soundness of CCR(X).

3.1.2. AN ANNOTATED REASONING THEORY FOR CCR(X)

We define the ARThs $ARth_{\mathsf{simp}}$, $ARth_{\mathsf{cr}}$, and $ARth_{\mathsf{cs}}$ over the RThs Rth_{simp}, Rth_{cr}, and Rth_{cs} (resp.) defined in Section 3.1.1.

SIMPLIFICATION

The ARTh for simplification is $ARth_{\mathsf{simp}} = \langle Rth^{A}_{\mathsf{simp}}, \epsilon_{\mathsf{simp}} \rangle$ where Rth^{A}_{simp} is equal to Rth_{simp} with the only difference being that the operation $\vee$ is no longer commutative (it is still associative though) and ϵ_{simp} is the identity mapping. The fact that $\vee$ is no longer commutative means that the relative order of literals in clauses matters at the control level. This affects the derivability relation presented by $ARth_{\mathsf{simp}}$ (when compared to that of Rth_{simp}). For instance, Rth_{simp} has derivation structures of the type $\langle [\,], a \vee b \underset{\mathsf{simp}}{\longrightarrow} b \vee a \rangle$ whereas no derivation structures of such a type exist in $ARth_{\mathsf{simp}}$.

REWRITING

The ARTh for rewriting is $ARth_{\mathsf{cr}} = \langle Rth^{A}_{\mathsf{cr}}, \epsilon_{\mathsf{cr}} \rangle$ where Rth^{A}_{cr} is obtained from Rth_{cr} by adding the operation $\lessdot$: $[\textsc{cnj}, \textsc{cnj}] \to \textsc{seq}$ and replacing `crew` by:

$$\frac{(cnj\lhd ?\sigma \wedge ?l[t\lhd ?\sigma]_{?u}) \prec\!\!\prec ?l[s\lhd ?\sigma]_{?u} \qquad ?cs :: cnj\lhd ?\sigma \xrightarrow[\mathsf{cr}]{} true}{?cs :: ?l[s\lhd ?\sigma]_{?u} \xrightarrow[\mathsf{cr}]{} ?l[t\lhd ?\sigma]_{?u}}\ \texttt{crew}$$

for all $(cnj \rightarrow s \approx t) \in \mathcal{R}$. Intuitively, $cnj \prec\!\!\prec cnj'$ asserts that the set of literals in cnj is smaller than the set of literals in cnj' w.r.t. the multiset extension of a simplification ordering.[†] This is reflected by the existence in R^A_{cr} of a set of axioms of the form $\texttt{msetord} : [\,] \rightarrow cnj \prec\!\!\prec cnj'$ such that $\{\langle |cnj|, |cnj'| \rangle \mid \texttt{msetord} : [\,] \rightarrow cnj \prec\!\!\prec cnj' \in R^A_{\mathsf{cr}}\}$ is the multiset extension of a simplification ordering. ($|cnj|$ denotes the set of literals occurring in cnj.) ϵ_{cr} is such that $\epsilon_{\mathsf{cr}}(cnj \prec\!\!\prec cnj') = \cdot$ (i.e. the sequents $cnj \prec\!\!\prec cnj'$ have no logical counterpart) and it is the identity mapping elsewhere.

CONSTRAINT SOLVING

The ARth for modeling the decision procedure is $ARth_{\mathsf{cs}} = \langle Rth^A_{\mathsf{cs}}, \epsilon_{\mathsf{cs}} \rangle$ where ϵ_{cs} is the identity mapping. Rth^A_{cs} closely resembles Rth_{cs}, the only difference being that if $\texttt{cs-simp} : [\,] \rightarrow cnj :: cs \xrightarrow[\mathsf{cs}]{} cs' \in R^A_{\mathsf{cs}}$, then $\{cnj, cs\} \models_{T_c} cs'$ and $\langle cnj, cs' \rangle \prec^{cs} \langle cnj, cs \rangle$, where $\prec^{cs}$ is a well-founded relation.

GLUING AND PARAMETERIZATION

A pARTh for CCR(X) is given by:

$$ARth_{\mathsf{ccr}} := (ARth_{\mathsf{simp}} + ARth_{\mathsf{cr}} + ARth_{\mathsf{cs}})[ARth_{\mathsf{cs}}].$$

It can be readily verified that $ARth_{\mathsf{simp}}$, $ARth_{\mathsf{cr}}$, and $ARth_{\mathsf{cs}}$ are glueable and that $ARth_{\mathsf{cs}}$ is faithfully included in $ARth_{\mathsf{simp}} + ARth_{\mathsf{cr}} + ARth_{\mathsf{cs}}$.

PROPOSITION 3.2. (SOUNDNESS OF $ARth_{\mathsf{ccr}}$) *If there exists a derivation* $\delta : \langle [\,], cl \xrightarrow[\mathsf{simp}]{} cl' \rangle$ *in* $ARth_{\mathsf{ccr}}$, *then* $\models_{T_j} \epsilon_{\mathsf{ccr}}(cl') \leftrightarrow \epsilon_{\mathsf{ccr}}(cl)$.

Besides the soundness of $ARth_{\mathsf{ccr}}$, we are now in the position to formally state and prove the termination of the integration schema. The recasting of the termination argument given in Armando and Ranise (2000) into the OMRS framework is a routine exercise and therefore is not discussed here.

3.1.3. A TACTIC THEORY FOR CCR(X)

We define the TThs Tth_{simp}, Tth_{cr}, and Tth_{cs} over the ARThs $ARth_{\mathsf{simp}}$, $ARth_{\mathsf{cr}}$, and $ARth_{\mathsf{cs}}$ (resp.) defined in Section 3.1.2.

[†] A simplification ordering is a well-founded relation closed under substitution and replacement that contains the sub-term relation. See Dershowitz and Jouannaud (1990) for the details.

SIMPLIFICATION

The simplification strategy is specified by the TTh $Tth_{\mathsf{simp}} = \langle Tsys_{\mathsf{simp}}, TR_{\mathsf{simp}}\rangle$ over the ARTh $ARth_{\mathsf{simp}}$. $T^T_{\mathsf{simp}} = \{\textsc{tac}, \ \textsc{tacs}\}$, $F^T_{\mathsf{simp}} = \{\textsc{fail}\}$, and $S^T_{\mathsf{simp}} = T^T_{\mathsf{simp}} \cup F^T_{\mathsf{simp}}$. O^T_{simp} contains a failure symbol $fail : [\,] \to \textsc{fail}$, a tactic $\mathtt{simplify} : [\,] \to \textsc{tac}$, a tactic $r : [\,] \to \textsc{tac}$ for each rule r in $ARth_{\mathsf{simp}}$ (namely the tactic $\mathtt{refl}$, $\mathtt{trans}$, and $\mathtt{cl\text{-}simp}$ of arity $[\,] \to \textsc{tac}$), and the tacticals $\mathtt{THENL} : [\textsc{tac}, \textsc{tacs}] \to \textsc{tac}$, $\mathtt{ORELSE} : [\textsc{tac}, \textsc{tac}] \to \textsc{tac}$, and $\mathtt{NF} : [\textsc{tac}] \to \textsc{tac}$. O^T_{simp} also contains the constructors for lists of tactics $[\,] : [\,] \to \textsc{tacs}$ and $[\_|\_] : [\textsc{tac}, \textsc{tacs}] \to \textsc{tacs}$.[‡]

For each rule in R_{simp} of the form $r : \langle [sq_1, \ldots, sq_n], (c :: e \xrightarrow[\lambda]{} e')\rangle$, the following tactic rule is in TR_{simp}:

$$\frac{}{r \lhd (c_0 :: e_0 \xrightarrow[\lambda]{} ?e) \rightsquigarrow res}$$

with $res = \langle [sq_1[\iota_p], \ldots, sq_n[\iota_p]], r, \iota_p\rangle, \iota_c\rangle$ if there exists an instantiation ι such that $(c :: e \xrightarrow[\lambda]{} e')[\iota] \equiv (c_0 :: e_0 \xrightarrow[\lambda]{} ?e)[\iota]$ (and in such a case ι_p is the restriction of ι to the variables in $\{sq_1, \ldots, sq_n\}$ and ι_c is the restriction of ι to $?e$) and $res = fail$ otherwise. For instance, the tactic rule associated to $\mathtt{cl\text{-}simp}$ is:

$$\frac{}{\mathtt{cl\text{-}simp} \lhd (cl \xrightarrow[\mathsf{simp}]{} ?cl') \rightsquigarrow res}$$

with $res = \langle\langle [\mathsf{cs\text{-}init}(?cs_0), \neg cl :: ?cs_0 \xrightarrow[\mathsf{cs-simp}]{} ?cs, ?cs :: ?l \xrightarrow[\mathsf{cr}]{} ?l'], \mathtt{cl\text{-}simp}, \iota_p\rangle, \iota_c\rangle$ if there exists an instantiation ι s.t. $(?cl \vee ?l)[\iota] \equiv cl$ (in such a case $\iota_p = \iota$ and ι_c is such that $?cl'[\iota_c] \equiv (?cl \vee ?l')[\iota]$) and $res = fail$ otherwise.

$(t_0 \ \mathtt{ORELSE} \ t_1)$ denotes the following proof strategy: try t_0 and in the case of failure try t_1. This is formalized by the following tactic rules:

$$\frac{t_0 \lhd sq \rightsquigarrow \langle \delta, \iota\rangle}{(t_0 \ \mathtt{ORELSE} \ t_1) \lhd sq \rightsquigarrow \langle \delta, \iota\rangle} \qquad \frac{t_0 \lhd sq \rightsquigarrow fail \qquad t_1 \lhd sq \rightsquigarrow res}{(t_0 \ \mathtt{ORELSE} \ t_1) \lhd sq \rightsquigarrow res}$$

$(t_0 \ \mathtt{THENL}[t_1 \ldots t_n])$ tries t_0 on the input sequent. If this yields a derivation of the form $[sq_1, \ldots, sq_n]; \delta$ then t_i is applied to sq_i for $i = 1, \ldots, n$ (in this order). The resulting derivations and instantiations are then combined in the obvious way. $(t_0 \ \mathtt{THENL} \ [t_1 \ldots t_n])$ fails if one of its argument tactics does. The tactic rules for $\mathtt{THENL}$ are not given here due to the lack of space.

When applied to a sequent of the form $c :: e \xrightarrow[\lambda]{} ?e$, $\mathtt{NF}$ computes the derivation structure of a sequent $c :: e \xrightarrow[\lambda]{} e'$ and binds $?e$ to e', where e' is the maximally reduced version of e in context c w.r.t. the tactic t. This is formalized by means of the following equations in E^T_i (for all $t \in \mathcal{T}^T_i$):

$$\mathtt{NF}(t) = (\mathtt{trans} \ \mathtt{THENL} \ [t \ , \ \mathtt{NF}(t)]) \ \mathtt{ORELSE} \ \mathtt{refl}.$$

The overall simplification strategy is modeled by the following equation in E^T_{simp}:

```
simplify = NF(cl-simp THENL [ cs-init, NF(cs-simp), NF(ccr) ])
```

[‡] For notational convenience we write $(t_0 \ \mathtt{ORELSE} \ t_1)$ and $(t_0 \ \mathtt{THENL} \ [t_1, \ldots, t_n])$, in place of $\mathtt{ORELSE}(t_0, t_1)$ and $\mathtt{THENL}(t_1, [t_1, \ldots, t_n])$, respectively. We also assume that $\mathtt{THENL}$ has precedence over $\mathtt{ORELSE}$ and that $\mathtt{ORELSE}$ associates to the right.

When applied to a sequent of the form $cl \underset{\mathsf{simp}}{\longrightarrow} ?cl$, `simplify` computes the derivation structure of a sequent $cl \underset{\mathsf{simp}}{\longrightarrow} cl'$, where cl' is the maximally reduced version of cl w.r.t. the application of `cl-simp` followed by the application of suitable tactics to the resulting sub-goals.

REWRITING

The simplification strategy is specified by the TTh $Tth_{\mathsf{cr}} = \langle Tsys_{\mathsf{cr}}, TR_{\mathsf{cr}}\rangle$ over the ARTh $ARth_{\mathsf{cr}}$. $T^T_{\mathsf{cr}} = \{\textsc{tac}, \textsc{tacs}\}$, $F^T_{\mathsf{cr}} = \{\textsc{fail}\}$, and $S^T_{\mathsf{cr}} = T^T_{\mathsf{cr}} \cup F^T_{\mathsf{cr}}$. O^T_{cr} contains the operations `refl`, `trans`, `crew`, `cxt-entails`, `msetord`, and `ccr` of arity $\langle [\,], \textsc{tac}\rangle$ and the tacticals `THENL` : $[\textsc{tac}, \textsc{tacs}] \to \textsc{tac}$, `ORELSE` : $[\textsc{tac}, \textsc{tac}] \to \textsc{tac}$, and `NF` : $[\textsc{tac}] \to \textsc{tac}$. O^T_{simp} also contains the constructors for lists of tactics $[\,] : [\,] \to \textsc{tacs}$ and $[\_|\_] : [\textsc{tac}, \textsc{tacs}] \to \textsc{tacs}$. The tactic rules for the tacticals are those of Tth_{simp}. The tactic rules associated to the rules of $ARth_{\mathsf{cr}}$ are defined similarly to those in Tth_{simp}. For instance, the tactic rule associated to `cxt-entails` is as follows:

$$\frac{}{\texttt{cxt-entails} \lhd (cs :: l \underset{\mathsf{cr}}{\longrightarrow} ?l') \rightsquigarrow \langle\langle[\delta_1, \delta_2], \texttt{cxt-entails}, \iota_p\rangle, \iota_c\rangle}$$

where $\delta_1 = \neg l :: cs \underset{\mathsf{cs}}{\longrightarrow} ?cs'$, $\delta_2 = \mathsf{cs\text{-}unsat}(?cs')$, ι_p is such that $?l[\iota_p] = l$, $?cs[\iota_p] = cs$ and ι_c is such that $?l'[\iota_c] = true$.

The overall rewriting strategy is expressed by the following equation in E^T_{ccr}:

```
ccr = (cxt-entails THENL [NF(cs-simp), cs-unsat])
      ORELSE (crew THENL [msetord, NF(ccr)])
```

which expresses the strategy of first applying the rule `cxt-entails` and to resort to `crew` only in the case of failure.

CONSTRAINT SOLVING

The decision procedure is specified by the TTh $Tth_{\mathsf{cs}} = \langle Tsys_{\mathsf{cs}}, TR_{\mathsf{cs}}\rangle$ over the ARTh $ARth_{\mathsf{cs}}$. $T^T_{\mathsf{cs}} = \{\textsc{tac}\}$, $F^T_{\mathsf{cs}} = \{\textsc{fail}\}$, and $S^T_{\mathsf{cs}} = T^T_{\mathsf{cs}} \cup F^T_{\mathsf{cs}}$. O^T_{cr} contains the operations `cs-init`, `cs-unsat`, and `cs-simp` of arity $\langle [\,], \textsc{tac}\rangle$. The tactic rules associated with the rules of $ARth_{\mathsf{cr}}$ are defined along the same lines as those in Tth_{simp}. For instance, the tactic rules associated with `cs-init` are:

$$\frac{}{\texttt{cs-init} \lhd \mathsf{cs\text{-}init}(?cs) \rightsquigarrow res}$$

with $res = \langle\langle[\,], \texttt{cs-init}, \texttt{idi}\rangle, \iota_c\rangle$ if there exists $\texttt{cs-init} : [\,] \to \mathsf{cs\text{-}init}(cs) \in R_{\mathsf{cs}}$ (and in such a case ι_c is such that $?cs[\iota_c] = cs$) and $res = fail$ otherwise.

GLUING AND PARAMETERIZATION

A pTTh for CCR(X) is given by:

$$Tth_{\mathsf{ccr}} := (Tth_{\mathsf{simp}} + Tth_{\mathsf{cr}} + Tth_{\mathsf{cs}})[Tth_{\mathsf{cs}}]$$

It can be readily verified that Tth_{simp}, Tth_{cr}, and Tth_{cs} are glueable and that Tth_{cs} is faithfully included in $Tth_{\mathsf{simp}} + Tth_{\mathsf{cr}} + Tth_{\mathsf{cs}}$.

Proposition 3.3. (Soundness of Tth_{ccr}) *If* $\mathtt{simplify} \triangleleft (cl \underset{\mathsf{simp}}{\longrightarrow} ?cl) \rightsquigarrow^{*} res$ *then* $res = fail$ *or* $res = \langle \delta, \iota \rangle$ *where* $\delta : [\,] \rightarrow (cl \underset{\mathsf{simp}}{\longrightarrow} ?cl[\iota])$ *is a derivation structure of* $ARth_{\mathsf{ccr}}$ *and* $\models_{T_j} \epsilon_{\mathsf{ccr}}(cl) \leftrightarrow \epsilon_{\mathsf{ccr}}(?cl[\iota])$.

4. Related Work

This paper is not a complete account of the OMRS specification framework, but focuses on the most recent work on the formalization of annotations and tactics. As a consequence, we have presented simplified versions of some formal concepts, whose more general formulations can be found in other papers (Giunchiglia *et al.*, 1994; Coglio, 1996; Coglio *et al.*, 2000). As shown in Bertoli *et al.* (1998) 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 generally, an OMRS specification can be used to support a variety of fundamental activities, ranging from the design and implementation phases up to the formal analysis of the properties of reasoning systems and the synthesis of provably correct reasoning components.

Among the features of CCR(X) neglected in this paper it is worth mentioning the *augmentation heuristic*. Such heuristics allow for the extension of the rewriting context with information about symbols which are uninterpreted for the decision procedure. As shown in Boyer and Moore (1988), augmentation can improve dramatically the effectiveness of the decision procedure at the cost of increasing the complexity of the resulting integration schema. However, the extension of the specification of CCR(X) given in this paper to incorporate augmentation should be straightforward.

OpenMath (Abbott *et al.*, 1998) is a language for *representing* and *communicating mathematics* initially intended as an interlingua for CASs. Recently OpenMath has been extended to support the encoding of generic mathematical entities (e.g. formulae and proofs) and therefore it can be conveniently used to support communication among a variety of reasoning systems: CASs, TPs, and MCs. (Caprotti and Cohen (1999) reports on the use of OpenMath to interconnect the computer algebra system Maple with interactive proof development systems such as Coq or Lego.) OMRS is a framework for the specification of the services provided by reasoning systems. If fleshed out with a concrete syntax (possibly based on OpenMath's) OMRS can provide a language and a theory for *representing*, *communicating*, and *reasoning about mathematical services*. The task of specifying mathematical services (intentionally left out of the scope of OpenMath) is crucial if the combination/integration issue is at stake. From these considerations it turns out that OMRS is largely complementary to OpenMath.

MathWeb (Franke *et al.*, 1999) is a distributed network architecture for automated and interactive theorem proving based on the *Agent-Oriented Programming* (Shoham, 1993) aiming at supporting modularization, interoperability, robustness, and scalability of mathematical software systems. In MathWeb each reasoning system is implemented as an agent which maintains information about the capabilities of other agents in a lookup table. However the agents' capabilities are modeled by a predefined set of "performatives" (e.g. `evaluate`, `ask-one`) whose semantics are left informal. It would be an interesting case study to see whether the semantics of such performatives could be specified in the OMRS framework.

The relationships between OMRS and Rewriting Logic (M.-Oliet and Meseguer, 1996) are investigated in Meseguer and Talcott (1998), where a category theoretic approach allows the obvious analogies to be made more precisely. Roughly speaking, the sequent system of a reasoning theory corresponds to the equational part of a rewriting logic theory, reasoning theory rules correspond to the rule part of a rewriting logic theory, and the derivations in a reasoning theory correspond to proof terms of a rewriting logic theory. Formally, these connections are stated by introducing the notions of abstract reasoning theory (ARTh), algebra of derivation structures for ARThs, and functor between an ARTh and the associated rewriting logic theory. As for the OMRS concepts presented in this paper, the reflective capabilities of Rewriting Logic (Clavel and Meseguer, 1996) allow for the specification of annotations at the meta-level, and the specification of strategies for the application of annotated rules at the meta-meta-level. The OMRS specification framework, therefore, offers a suitable way of structuring specifications in Rewriting Logic. On the other hand, the reflection mechanism provides a formal and rigorous foundation to specify the semantical relationships between the logic and control layers. One of the main advantages in establishing a correspondence between Rewriting Logic and the OMRS specification framework is the capability of exploiting the available efficient implementations of Rewriting Logic (e.g. Maude and ELAN (Borovansky *et al.*, 1998)) to fast-prototype reasoning systems directly from OMRS specifications.

5. Conclusions

In this paper we have presented the control layer of the OMRS specification framework. The layer allows for the specification of the control component of mechanized reasoning systems via annotations and tactics. We have shown that the additional layering offered by the OMRS framework complements and smoothly extends the standard approach to structure specifications based on modularity. As a case study we have outlined an OMRS specification of CCR(X) as a set of cooperating specialized reasoning modules. The case study has shown that the additional structure provided by the OMRS specification framework is fundamental to cope with the complexity of functionalities provided by state-of-the-art implementations.

Acknowledgements

We especially thank Paolo Pecchiari for his initial contributions to this work, and Carolyn Talcott for contributing to the technical development of the theory. We also would like to thank the anonymous referees for their valuable comments about the paper.

References

Abbott, J., van Leuwen, A., Strotmann, A. (1998). OpenMath: communicating mathematical information between co-operating agents in a knowledge network. *J. Intell. Syst.*, **8**, 401–426.

Armando, A., Ranise, S. (1998a). Constraint contextual rewriting. In Caferra, R., Salzer, G. eds, *Proceedings of the 2nd International Workshop on First Order Theorem Proving (FTP'98), Vienna, Austria, November 23–25, 1998*, pp. 65–75.

Armando, A., Ranise, S. (1998b). From integrated reasoning specialists to "Plug-and-Play" reasoning components. In *Proceedings of the Fourth International Conference Artificial Intelligence and Symbolic Computation (AISC98), Plattsburgh (USA)*, LNCS **1476**, pp. 42–54.

Armando, A., Ranise, S. (2000). Termination of constraint contextual rewriting. In *Proceedings of 3rd International Workshop on Frontiers of Combining Systems (FroCoS'2000), France, Nancy*, LNAI **1794**, pp. 47–61.

Ballarin, C., Homann, K., Calmet, J. (1995). Theorems and algorithms: an interface between Isabelle and Maple. In Levelt, A. H. M. ed., *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'95), Montreal, Canada*, pp. 150–157. New York, ACM Press.

Bertoli, P. G., Calmet, J., Giunchiglia, F., Homann, K. (1998). Specification and integration of theorem provers and computer algebra systems. In Calmet, J., Plaza, J. eds, *Proceedings of the International Conference on Artificial Intelligence and Symbolic Computation (AISC-98)*, LNAI **1476**, pp. 94–106. Berlin, Springer.

Borovansky, P., Kirchner, C., Kirchner, H., Moreau, P., Ringeissen, C. (1998). An overview of ELAN. In Kirchner, C., Kirchner, H. eds, *Proceedings of the 2nd International Workshop on Rewriting Logic and its Applications*, *Pont-A-Mousson, France*.

Boyer, R. S. (1971). Locking: A Restriction of Resolution. Ph.D. Thesis, University of Texas, Austin, Texas.

Boyer, R., Moore, J. S. (1979). *A Computational Logic*. New York, NY, Academic Press.

Boyer, R., Moore, J. S. (1988). Integrating decision procedures into heuristic theorem provers: a case study of linear arithmetic. *Mach. Intell.*, **11**, 83–124.

Buchberger, B., Jebelean, T., Kriftner, F., Marin, M., Tomuţa, E., Vāsaru, D. (1997). A survey of the theorema project. In Küchlin, W. W. ed., *ISSAC '97. Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, July 21–23, 1997, Maui, Hawaii*, pp. 384–391. New York, NY, USA, ACM Press.

Caprotti, O., Cohen, A. M. (1999). Integrating computational and deduction systems using openmath. *Electron. Notes Theor. Comput. Sci.*, **23**.

Clarke, E., Zhao, X. (1992). Analytica—a theorem prover in mathematica, *Lecture Notes in Computer Science*, **607**, 761–765.

Clavel, M., Meseguer, J. (1996). Axiomatizing reflective logics and languages. In Kiczales, G. ed., *Proceedings of the Reflection'96, Xerox PARC*, pp. 263–288.

Coglio, A. (1996). 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. Thesis, DIST—University of Genoa (Italy).

Coglio, A., Giunchiglia, F., Meseguer, J., Talcott, C. (2000). Composing and controlling deduction in reasoning theories using mappings. In *Third International Workshop on Frontiers of Combining Systems, FroCoS 2000*, LNAI **1794**, pp. 200–216.

Constable, R., Allen, S., Bromley, H. *et al.* (1986). *Implementing Mathematics with the NuPRL Proof Development System*. Englewood Cliffs, NJ, Prentice Hall.

Dershowitz, N., Jouannaud, J. (1990). Rewriting systems. In *Handbook of Theoretical Computer Science*, pp. 243–320. Amsterdam, Elsevier.

Ehrig, H., Mahr, B. (1985). *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. New York, NY, Springer.

Franke, A., Hess, S., Jung, C., Sorge, V. (1999). Agent-oriented integration of distributed mathematical services. *J. Universal Comput. Sci.*, **5**, 156–187.

Gentzen, G. (1934). Untersuchungen über das logische schließ. *Math. Z.*, **39**, 176–210, 405–433. English translation in Gentzen (1969).

Gentzen, G. (1969). Investigations into logical deduction. In Szabo, M. ed., *The Collected Papers of Gerhard Gentzen*, pp. 68–128. Amsterdam, North-Holland.

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

Gordon, M., Milner, A., Wadsworth, C. (1979). *Edinburgh LCF—A Mechanized Logic of Computation*, LNCS **78**. Berlin, Springer.

Harrison, J. (1998). *Theorem Proving with the Real Numbers*. Berlin, Springer.

Harrison, J., Théry, L. (1998). A skeptic's approach to combining Hol and Maple. *J. Autom. Reasoning*, **21**, 279–294.

Jackson, P. (1994). Exploring abstract algebra in constructive type theory, *Lecture Notes in Computer Science*, **814**, 590–604.

Jaffar, J., Maher, M. (1994). Constraint logic programming: a survey. *J. Log. Program.*, **19/20**, 503–581.

Kapur, D., Nie, X. (March 1994). Reasoning about Numbers in Tecton. Technical Report, Department of Computer Science, State University of New York, Albany, 12222 NY.

M.-Oliet, N., Meseguer, J. (1996). Rewriting logic as a logical and semantic framework. In Meseguer, J. ed., *First International Workshop on Rewriting Logic and its Applications, RWLW96*, volume 4 of *Electronic Notes in Theoretical Computer Science*. `http://www.elsevier.nl/locate/entcs/volume4.html`.

Meseguer, J., Talcott, C. (1998). Mapping OMRS to rewriting logic. In Kirchner, C., Kirchner, H. eds, *2nd International Workshop on Rewriting Logic and its Applications, WRLA'98*, volume 15 of *Electronic Notes in Theoretical Computer Science*. `http://www.elsevier.nl/locate/entcs/volume15.html`.

Paulson, L. (1989). The foundation of a generic theorem prover. *J. Autom. Reasoning*, **5**, 363–396.

Shoham, Y. (1993). Agent-oriented programming. *Artif. Intell.*, **60**, 51–92.

Zhang, H. (1995). Contextual rewriting in automated reasoning. *Fundam. Inform.*, **24**, 107–123.

Originally Received 7 November 1999
Revised 29 September 2000
Accepted 1 November 2000