Reasoning Theories

Towards an Architecture for
Open Mechanized Reasoning systems

Fausto Giunchiglia
IRST and Università di Trento
fausto@irst.it

Paolo Pecchiari
IRST and Università di Genova
peck@irst.it

Carolyn Talcott
Stanford University
clt@sail.stanford.edu

Abstract:

Our ultimate goal is to provide a framework and a methodology which will allow users, and not only system developers, to construct complex reasoning systems by composing existing modules, or to add new modules to existing systems, in a "plug and play" manner. These modules and systems might be based on different logics; have different domain models; use different vocabularies and data structures; use different reasoning strategies; and have different interaction capabilities. This paper makes two main contributions towards our goal. First, it proposes a general architecture for a class of reasoning modules and systems called *Open Mechanized Reasoning Systems* (*OMRSs*). An OMRS has three components: a *reasoning theory* component which is the counterpart of the logical notion of formal system, a *control* component which consists of a set of inference strategies, and an *interaction* component which provides an OMRS with the capability of interacting with other systems, including OMRSs and human users. Second, it develops the theory underlying the reasoning theory component. This development is motivated by an analysis of state of the art systems. The resulting theory is then validated by using it to describe the integration of the linear arithmetic module into the simplification process of the Boyer-Moore system, NQTHM.

1. Plug and Play Reasoning Devices – An Impossible Dream?

An important problem in the domain of automated reasoning is the development of mechanisms for the interoperation and integration of disparate provers. [1] The components of a prover may be tightly or loosely coupled, they may be based on different logics, they may have different domain models, they may use different vocabularies, representations of information, and reasoning strategies, and they may have different interaction capabilities. We want to be able to compose complex provers from existing modules

$$\text{prover} = \text{tautology-checker} + \text{rewriter} + \text{simplifier} + \text{unifier} + \ldots,$$

to add new modules to existing provers

$$\text{enhanced-prover} = \text{add-decider(prover,decider,use-specification)},$$

and to form provers using multiple logics

$$\begin{aligned}\text{multilogic-prover} = {} & \text{HO-proof-checker} \\ & + \text{FO-rewriter} \\ & + \text{Modal-inferencer} \\ & + \text{Simplifier} \\ & + \text{Model-checker},\end{aligned}$$

in a "plug and play" manner.

The need for composing modules, or adding new modules to existing provers, is motivated by the desire of not having to build from scratch a new prover for each new problem or variation of an old problem. Multi-logic provers are needed in many formalization problems, such as hardware and software verification, which typically rely on the use of a variety of decompositions and levels of abstraction. These in turn are most naturally formalized using a variety of logics: higher-order logic for general theory and methods; temporal and dynamic logics for behavior specification; first-order logics for reasoning about data structures; various decidable fragments for verification steps. Multi-logic provers are also needed in complex applications which require embedding of reasoning modules inside other systems. Some examples are: program transformation systems, including synthesis, partial evaluation and compiling; planning systems; intelligent agents; and natural language systems.

Currently, if you need a prover there are two choices: (1) implement your own; or (2) adapt an existing prover to your needs, or, more likely, adapt your needs to an existing prover. Given the state-of-the-art of technology for building provers,

---

[1] In this paper we use the word "prover" to mean any piece of software with reasoning capabilities, e.g. mechanized decision procedures, automated theorem provers, interactive theorem provers. When we want to distinguish stand alone provers from provers embedded inside other software we refer to the former as systems and the latter as modules.

neither option is satisfactory. Serious provers are difficult to build and there is little in the way of generic parts or tools to help. Existing systems are difficult to connect – they are packaged as stand alone software with inadequately described interfaces. Furthermore it is difficult to extract usable modules from existing provers, since they typically depend upon internal structures of the host prover.

Is the dream impossible? We can compare the current state of automated reasoning technology to early computer technology which featured stand alone computers with no capability to exchange data and no possibility of interchange of hardware modules. If we take this analogy seriously, the current situation with computer technology suggests there is hope.

2. Logical Services and OMRSs

To realize the dream we need to think of provers as *logical services* [62], and to develop a general framework for specifying and structuring provers as logical services. What is a logical service? That is, what is required of provers and their specifications in order to be able to interconnect or integrate them in useful, semantically meaningful ways? This is an open question and we do not expect there is a unique answer. Below we give a preliminary, informal analysis. One of the objectives of our work is to find helpful answers, as well as to develop tools for meeting the resulting requirements. We note that many of the issues discussed are not unique to the domain of automated reasoning. Solutions to problems found here should also be relevant in other application domains.

The specification of a logical service must include logical semantics, algorithmic and control information, and information regarding interaction requirements and capabilities. It must include information that allows for two way interactions with the environment. To qualify as a logical service, a prover needs wrappings that specify when, how, and for what purpose it can or should be used; and what services or information it requires, or can use. These wrappings must support a mix of openness and encapsulation that allows for the desiderata listed in the previous section, namely linking separate, independent provers; adding new modules to an existing prover; and extending and integrating modules. The specification must be such that integration activities should not require major redesign of the existing provers; and in the case of highly tuned provers, the recoding should be driven by local changes in a systematic, possibly partially automated manner.

Provers providing logical services must therefore be described at many levels: traditional consequence relations; data structures used for mechanizing deduction; inference algorithms; annotations and control information; interaction capabilities and protocols and their translation into the underlying representations; and the sharing and updating used for communication and for efficient implementation. An important gap that needs to be filled is an analysis of the structures and protocols that are needed in order to specify interactions, and to support mechanisms for incremental, restartable, reactive deduction.

We introduce the notion of *Open Mechanized Reasoning System (OMRS)* as an architecture for specifying and implementing logical services. This architecture has three layers.

Reasoning Theory = Sequents + Rules

Reasoning System = Reasoning Theory + Control

OMRS = Reasoning System + Interaction

A *reasoning theory* consists of a set of assertions, called *sequents*, and a set of inference *rules*. In an OMRS, the reasoning theory level is the counterpart of the logical notion of formal system. A reasoning theory determines a set of proof fragments called *reasoning structures*. Derivations and proofs are identified as reasoning structures satisfying certain conditions. A *reasoning system* consists of a reasoning theory and a set of strategies for searching the space of possible applications of inference rules. In an OMRS, the reasoning system level is the formal counterpart of the informal notion of a prover without interaction capabilities (e.g. a decision procedure module viewed as a black box). Finally, an OMRS is a reasoning system extended with a set of interaction capabilities. This level is the formal counterpart of the informal notion of a prover providing a logical service.

Notice that a logical system [43] also contains a notion of model, that is

Logical System = Reasoning Theory + Models

We realize that this is an important aspect of reasoning which must be eventually included in our framework. However, at this stage we have not dealt with this problem. It is a topic for future work.

Plan: The remainder of this paper develops the theory underlying the reasoning theory component of OMRSs. It is organized in three main parts: (I) analysis of the problem; (II) technical development; and (III) substantial example. Part I surveys existing systems (§3) and discusses the features that are needed to provide an adequate framework (§4). In part II, §5, §6 and §7 define and illustrate the notions of sequent system, rules, and reasoning theory, respectively. In §8 reasoning structures are defined as certain labelled graph structures and the notions of derivation and proof are defined. In §9 a set of primitive operations for constructing reasoning structures is defined and shown to be complete. Rule application operations corresponding to various directional modes are defined as sequences of primitive operations. Part III sketches an analysis of the integration of linear arithmetic into the Boyer-Moore prover, NQTHM, using the reasoning theory framework. §10 describes (briefly) NQTHM and §11 gives an outline of our analysis of the prover. In §12 the reasoning theory underlying the original prover is sketched. In §13 the modification of the reasoning theory to integrate linear arithmetic reasoning is discussed. In §14 some examples of reasoning structures of the NQTHM reasoning theory are given. Among other things these examples illustrate the use of reasoning structures to present proofs. Part IV contains end material. Related and future work are discussed respectively in §15 and §16. §17 contains the proofs of the main theorems about reasoning structures stated in the earlier sections.

3. Existing systems

To give an idea of features that an architecture for provers must account for, we briefly examine a few existing systems. First we look at the variety of techniques integrated into single logic provers. Then we look at some experiments with multi-logic provers. We emphasize that this is not intended as a comprehensive survey, but rather as a set of motivating examples. A database of automated reasoning systems can be found in [63] along with links to related surveys and other information.

3.1. Single Logic Systems

NQTHM [9, 10] combines techniques for propositional reasoning, equality reasoning, typeset inference, term rewriting, and linear arithmetic. Several of these techniques use data structures that encode special purpose representations of logical information. These data structures may also include control and heuristic information that must be maintained and propagated by processes even if they do not make use of it. The experience of integrating linear arithmetic reasoning into the prover as reported in [8] provides strong evidence that in general it is not adequate to simply integrate a decision procedure as a black box.

The LCF family of systems, that includes LCF itself [26, 52], HOL [25], NuPrl [14], and Isabelle [53], all provide capability for user defined proof procedures (tactics and tacticals) along with various builtin procedures. NuPrl provides for extraction of programs from proofs and a reflection principle for turning verified tactics into first class inference rules. Isabelle provides mechanisms for defining proof systems by specifying their syntax and rules of inference.

FOL [67, 68, 69] provides for construction of and reasoning in an intended model, rewriting, mixed syntactic and semantic simplification, tautology checking based on compiling formulas into a special representation, a decision procedure for monadic predicate logic, and user defined simplification sets (theory specific sets of rewrite rules).

EKL [36, 38] has a highly developed rewriting component. The rewriter 'compiles' contextual information and stores it as annotations to subexpressions in the form of rewrite rules. There is a language for expressing rewriting control strategies. EKL also incorporates a decision procedure for direct logic [37, 5] and some facility for representing terms as data and carrying out meta-level reasoning.

RRL [34, 35] represents all axioms as equations and views theories as congruence relations. The deductive machinery of RRL includes: a completion procedure;

rewriting; associative-commutative theories; linear arithmetic; Groebner basis over boolean ring; inductionless and explicit induction. It can be used for generating decision procedures for first-order theories, checking consistency and completeness of equational specifications, and solving equations modulo an equational theory.

The KADS system [61] uses resolution augmented with special purpose deciders. EHDM [66] and PVS [51] use a variety of ground decision procedures combined with rules for interactive proving. PVS has a rich type system and provides the ability to postpone type checking, by making presumptions, analogous to verification conditions. The Ontic rule compiler [42] compiles sets of rules of suitable form into efficient decision procedures. The IMPS system [16] uses theory interpretation maps to import results from one theory into another theory, and macetes (tricks) to express theory specific rules. Hyperproof [3]provides two representations of information, diagrams and first-order sentences, to reason about simple blocks worlds problems. Inference rules are provided to move information between the two forms as well as for reasoning within one paradigm.

3.2. Multi-logic Systems

The cooperating decision procedures (CDP) paradigm of Nelson and Oppen [49] is a method of combining solver/simplifier modules for disjoint theories to obtain solvers/simplifiers for combined theories. Shostak [59] gives an alternative algorithm for combining solvers, for theories meeting certain constraints, to obtain decision procedures for combined first order theories. As further evidence that it is not adequate to simply integrate a decision procedure as a black box, we note that most systems that incorporate a Nelson-Oppen type simplifier require that the simplifier be able to accept and use lemmas. While the basic composition algorithm is well-worked out and has clear semantics, the formal mechanism for introduction of lemmas has not been worked out.

The SDVS system [15, 55] is a proof system currently used for various levels of hardware verification. The system contains a number of modules reflecting the factoring of proof development into reasoning about dynamic and static aspects. Reasoning about static aspects uses a proof system for classical logic and domain specific simplifiers. Temporal logic and symbolic execution are used for the dynamic aspects. The simplifier module is based on the CDP paradigm. The EKL prover is embedded and used for proving higher-order statements. The SDVS simplifier has been integrated into the Ada verification system, Penelope, in order to provide support for simplifying of verification conditions [62].

NQTHM has been used as a component of various systems in which another component of the system (possibly an ad hoc program or a person) transforms the problem to suitable form. Verification Condition checking is described in [7] (for Fortran) and [48] (for Pascal). In [11] a general theory, MLP, for defining semantics of synchronous circuits, was developed (informally) using function parameters. A Lisp front-end was used to map circuit descriptions to a sequence of definition and prove-lemma events for NQTHM. This simulated the needed higher-order capability for expressing general definitions and quantification over functions.

Recent proceedings of HOL User meetings [13, 29] report a variety of experiments to incorporate additional inference capability, or to link HOL to other systems. FAUST [56] is a prover for full first-order logic developed for use with HOL. Archer [1] describes work in progress to define translations of certain HOL goals to input for a resolution prover. Joyce and Seger [32] describe a link to the Voss model checking tool to be used in hardware verification. Additional work encoding other formalisms in HOL includes: Boyer-Moore logic, Unity, a real number algebra decision procedure, and AC unification.

There is also a large amount of work in the area of hybrid systems. An overview of this work can be found in [30]. To mention some examples: Bundy [12] proposes using proof plans for combination of reasoning strategies; the blackboard model [31] is a mechanism for coordination of problem solving activities of planners and reasoning experts with a common language; Myers [47] proposes universal attachment with interfaces for calls to external procedures as a basis for integration of decision procedures for domain specific theories within a single logic. Sikka [60] proposes a generalization of this mechanism in which the external procedures are axiomatized within the logic.

4. Issues

In this section we discuss the issues arising in the analysis of existing provers, and the corresponding features which must be represented within reasoning theories. We first consider three aspects of existing systems corresponding to the three main concepts underlying reasoning theories: sequents; rules; and reasoning structures and deductions. We conclude the section with a discussion of the problem of integration of multiple heterogeneous reasoning theories.

4.1. Sequents

The sequents or assertions manipulated by actual provers typically have more structure and information (procedural and contextual) than assertions of traditional logical systems. The sequents of a simple natural deduction system are pairs consisting of a set or list of open assumption formulae, and a conclusion formula. A natural deduction system for a particular theory may also have a set of axioms usually left implicit, but still considered part of the sequent. Other traditional proof calculi have sets or lists of formulae in the conclusion part as well. Resolution based systems manipulate sets of clauses. Rewriters manipulate equalities between terms relative to a context of assumptions. During simplification using a Nelson-Oppen type simplifier, modules build up context as literals are assumed (and retracted). Abstractly, a simplifier context is a set of literals. Each participating module has its own view of the context, containing information expressible in its language. The internal logic of a simplifier module uses representations of information suited to its particular decision processes, for example: term graphs, simplex structures, bit vectors, and binary decision diagrams. Highly tuned provers such as NQTHM have a variety of special purpose inference modules, each with their own representation of context and assertions. Diagrams are another form of assertion: Venn diagrams in simple set theory; timing, state, and circuit diagrams used in hardware design and verification; and the notation systems of various software-design methodologies are a few examples. The Hyperproof system manipulates simple diagrams representing blocks world states.

Many provers make use of some form of schematic sequents. Languages that allow assertion of axiom schemas usually allow deduction of schematic sequents – theorem schemas. Axiom schemas are often used in first-order languages where quantification over predicate or function variables would be used in higher-order languages. Axioms schemas restricted to formulas of certain forms can be used to express classes of formulas that can not be expressed simply by object level quantification. Another use of schematic assertions is to provide the ability to factor out constraints and solve them lazily. This is useful in provers that use forms of higher-order unification. Schematic variables can also be used to give a first-order account of higher-order patterns in such as those of Combinatory Reduction Systems [40, 45, 50, 65]. Another interesting example of deduction using schematic variables is in the NQTHM linear arithmetic module, where a single derivation can serve multiple purposes by using place holders to represent auxiliary information that does not effect the polynomial derivation process.

To summarize, the notion of sequent should be abstract enough to express a wide variety of information and forms of representation including the features of the concrete assertions discussed above; it should allow for the use of local context; it should allow for the use of schematic variables.

4.2. Rules

Inference rules of standard logical systems, for instance natural deduction and resolution, are n-ary functions possibly with some side conditions which establish their applicability. Most provers have schematic rules, i.e. rules which manipulate sequents containing schematic variables; and many allow variable arity (variary) inference rules. Inside a prover, rules can be implemented backward as in NuPrl, forward as in FOL, both backward and forward as in HOL or GETFOL [20], or as operations which link the premisses (asserted as theorems) to the conclusion (asserted as an open goal) (in GETFOL this operation is called “matching”).

Rules used by provers are often elaborations of purely logical rules to include control and heuristic information used to constrain applicability. Side conditions for rule application can be arbitrarily complicated and rely on the use of complex reasoning modules, e.g. a tautology checker or some other decision procedure. One simple example in natural deduction is the check of the occurrence of the free variable in the application of universal quantifier introduction.

To summarize, the notion of rule should allow for the specification of concrete rules which manipulate complicated sequents; it should allow for schematic reasoning, i.e. for the manipulation of sequents containing schematic variables; it should allow for the possibility of a variable number of premisses. Rules should be defined to be adirectional, therefore uniformly capturing all the possible modes of application. Rules should allow for the specification of applicability constraints separate from the specification of the sequents linked by the rule. This provides for separate consideration of structural rule matching and more complex constraints, and for the postponement of constraint checking. The notion of constraint should be abstract enough to be applicable in the specification of any prover.

4.3. Reasoning Structures and Derivations

Existing provers support one or more forms of interaction and deduction: forward, backward, and mixed mode proof construction. NuPrl proofs are developed by refinement (goal directed construction). HOL allows both forward and goal directed deduction. FOL proofs are forward. GETFOL proofs can be in mixed mode.

Existing provers exhibit a variety of derivation structures ranging from derivations whose existence is implicit in the claim by a prover that an assertion has been proved, to data structures representing complete proofs within a specified formal system. NQTHM provides no formal representation of proof, it only succeeds or fails in determining whether or not a conjecture is a theorem (and reports the lemmas used in case of success). On the other extreme, NuPrl constructs proof terms, and provides for extraction of programs from proofs. In FOL, deductions are presented as sequences of structures called verification lines (VLs), representing a graph structure. Each VL has an identifier, a formula, a set of dependencies, and a justification. A dependency is an identifier of a VL whose justification is by assumption. Another kind of justification is the application of a deduction rule to a list of VL identifiers (the premiss VLs). A VL may be used as a premiss in several rule applications, providing a sharing of sub-deductions. In the IMPS system, derivations are represented as deduction graphs – labelled graphs with two sorts of nodes, sequent and rule nodes. A sequent may be linked as premiss to any number of rule nodes (possibly none) giving sharing of subderivations. A sequent may be the conclusion of zero or more rule nodes, allowing multiple proof (attempt)s. Cycles are allowed, expressing mutual derivability. Other forms of proof include Tableaux, Matings, Proof nets, and Truth tables.

Experience shows that there is need to organize large complex structures hierarchically, to be able to examine them at different levels of depth and detail, or to focus on meaningful substructures. This applies to programs, theories, logics, and especially to proof structures. Tactics and tacticals provide a hierarchical way of describing the search for or construction of a proof, but the resulting proofs (if actually constructed rather than just checked) are flat. There is some work on presentation of proofs that address these issues (cf. [1, 33]). The problem needs to be addressed at the proof construction level not just at the presentation level. It is important to structure both the proof construction process and the resulting proof structure. Hierarchical organization can be obtained by encapsulation of substructures as derived rule applications, procedure calls, lemmas, or simply boxes that can be opened up and examined in more detail if desired.

Some provers also support provisional reasoning (structures that are valid deductions when certain constraints are met). In these situations, checking for the applicability of an inference rule is sometimes postponed until after the rule is applied. In this case we say that the system performs provisional reasoning. Some examples are lazy occur check and lazy term unification, and postponement of establishing hypotheses in conditional rewriting.

To summarize, the notion of deduction should allow for any possible mode of construction and for provisional reasoning. It should provide for two dimensions of flexibility in the construction and structuring of derivations: horizontal flexibility to combine and refine fragments; and vertical flexibility to choose the level of detail exposed.

4.4. Integration of Reasoning Theories

As we noted in §3, many systems make use of multiple representations of information, possibly in the context of multiple logics, and may even represent heterogeneous proofs that combine reasoning in the different logics or different mechanizations of the same logic. In these cases, inference rules with premisses and conclusions corresponding to different kinds of assertions are needed to glue the pieces together. Examples of such rules in existing systems include: bridge rules in multi-language systems [19, 21]; reflection rules in FOL and NuPrl; meta-rules in the Boyer-Moore logic [10]; and inference rules for reading information off of a diagram, and for using linguistic assertions to extend diagrams in Hyperproof.NuPrl use these rules as formal mechanisms for integration of inferencers proved to be sound. The Ontic rule compiler provides a mechanism for integrating inferencers, when described by suitable rules. The Nelson-Oppen and Shostak algorithms for combining decision procedures suggest mechanisms for integration of additional inferencers, although no formal mechanism has been fully spelled out.

To summarize, the notions of reasoning theory, sequent, inference rule, reasoning structure and derivation should be general enough to allow for the integration of multiple reasoner, possibly implementing different logics and using different data structures.

II Technical Development

In this part we define the notions of sequent system, rule, reasoning theory, and reasoning structure. These abstract concepts are illustrated with examples from a proof system of classical logic (Natural Deduction) and existing provers, mainly NQTHM. Further details of the NQTHM example are given in Part III of this paper. A more complete analysis is presented in [22].

Before proceeding with the technical development, we introduce the mathematical notation we will use. We use the usual notation for set membership and function application. Let Y, Y_0, Y_1 be sets. We specify meta-variable conventions in the form: let y range over Y, which should be read as: the meta-variable y and decorated variants such as y', y_0, ..., range over the set Y. $Y_0 \times Y_1$ is the set of pairs with first component from Y_0 and second component from Y_1. Y^* is the set of finite sequences of elements of Y. We write $[y_1, \ldots, y_n]$ for the sequence of length $Len(\bar{y}) = n$ with ith element y_i. (Thus $[\,]$ is the empty sequence.) $u \diamond v$ denotes the concatenation of the sequences u and v. $\mathrm{P}_\omega(Y)$ is the set of finite subsets of Y. The empty set is denoted by $\emptyset$. We use the convention that if y ranges over Y, then $\bar{y}$ ranges over Y^* and $\widetilde{y}$ ranges over $\mathrm{P}_\omega(Y)$. $[Y_0 \xrightarrow{f} Y_1]$ is the set of finite maps from Y_0 to Y_1. We use $\vec{\emptyset}$ to denote the (unique) finite map with empty domain. $[Y_0 \to Y_1]$ is the set of total functions, f, with domain Y_0 and range contained in Y_1. We write $\mathrm{Dom}(f)$ for the domain of a function and $\mathrm{Rng}(f)$ for its range. If $f \in [Y_0 \to Y_1]$ and $g \in [Y_1 \to Y_2]$, then $g \circ f \in [Y_0 \to Y_2]$ is the composition of f and g: $(g \circ f) = \lambda y.g(f(y))$. For any function f, $f\{y \mapsto y'\}$ is the function f' such that $\mathrm{Dom}(f') = \mathrm{Dom}(f) \cup \{y\}$, $f'(y) = y'$, and $f'(z) = f(z)$ for $z \neq y, z \in \mathrm{Dom}(f)$; and $f \downarrow Y$ is the restriction of f to the set Y. If $\bar{y} = [y_i \mid i < n] \in Y_0^*$, $\widetilde{y} \in \mathrm{P}_\omega(Y_0)$ and $f \in [Y_0 \to Y_1]$, then we write $f(\bar{y})$ and $f(\widetilde{y})$ to denote respectively the sequence $[f(y_i) \mid i < n] \in Y_1^*$ and the set $\{f(y) \mid y \in \widetilde{y}\} \in \mathrm{P}_\omega(Y_1)$.

5. Sequent Systems

Only certain general features of sequents and rules are needed to describe the notions of reasoning structure and derivation associated to a reasoning theory, and the operations for constructing reasoning structures. These are abstracted in the notion of *sequent system*. This allows us to decouple the definitions of reasoning structure and derivation from the details of any specific sequent system.

5.1. Definition

A sequent system is a structure:

$$Ssys = \langle S, C, \models, I, \_\texttt{[}\_\texttt{]} \rangle$$

S is the set of sequents – assertions or judgements for consideration. C is the set of constraints, needed to allow for the construction of provisional derivations. $\models \subseteq (\mathrm{P}_\omega(C) \times C)$, is a consequence relation on constraints, which abstractly represents a constraint solving (satisfaction) mechanism. I is the set of instantiation maps (or instantiations), and $\_\texttt{[}\_\texttt{]}$ is the operation for application of instantiations to sequents and to constraints, that is $\_\texttt{[}\_\texttt{]} : [S \times I \to S]$ and $\_\texttt{[}\_\texttt{]} : [C \times I \to C]$. Thus, both sequents and constraints can be schematic and instantiation provides a means for filling in schemata. From now on, we let s range over S, c range over C, and ι range over I. In the remainder of this subsection we describe the requirements that such a structure must meet in order to qualify as a sequent system, and introduce some auxiliary definitions.

Satisfaction must obey the basic laws for a (classical) consequence relation (cf. [2, 43]):

(mon) if $\widetilde{c} \subseteq \widetilde{c}\,'$ and $\widetilde{c} \models c$, then $\widetilde{c}\,' \models c$;

(ax) if $c \in \widetilde{c}$, then $\widetilde{c} \models c$;

(cut) if $\widetilde{c} \models c$ and $\{c\} \cup \widetilde{c}\,' \models c'$ then $\widetilde{c} \cup \widetilde{c}\,' \models c'$.

We extend satisfaction to a relation between sets of constraints by defining

$$\widetilde{c} \models \widetilde{c}_1 \;\Leftrightarrow\; (\forall c \in \widetilde{c}_1)(\widetilde{c} \models c).$$

Let us call *schematic entities* the entities that I acts on producing entities of the same sort. The collection of schematic entities includes sequents and constraints, and it is closed under formation of finite sets or sequences, and finite maps whose range is a set of schematic entities with instantiation extended pointwise. For X any set of schematic entities:

$$\begin{aligned}
&\widetilde{x}\,\texttt{[}\iota\texttt{]} = \{x\texttt{[}\iota\texttt{]} \mid x \in \widetilde{x}\} \quad \text{for} \quad \widetilde{x} \in \mathrm{P}_\omega(X)\\
&\overline{x}\,\texttt{[}\iota\texttt{]} = [x_i\texttt{[}\iota\texttt{]} \mid i < n] \quad \text{for} \quad \overline{x} = [x_i \mid i < n] \in X^*\\
&f\texttt{[}\iota\texttt{]} = \lambda y.(f(y)\texttt{[}\iota\texttt{]}) = (\lambda x.x\texttt{[}\iota\texttt{]}) \circ f \quad \text{for} \quad f \in Y \to X
\end{aligned}$$

idi is the identity instantiation, $x\texttt{[idi]} = x$ for any schematic entity x. Two instantiations ι, ι' agree on a set X, written $\iota =_X \iota'$, if $x\texttt{[}\iota\texttt{]} = x\texttt{[}\iota'\texttt{]}$ for $x \in X$. Two finite sets X, Y of schematic entities are *schematically separated* if for any instantiation ι there is some ι' (restriction of ι to X) such that $\iota' =_X \iota$ and $\iota' =_Y \texttt{idi}$. Instantiations are closed under composition. Thus if $\iota_0, \iota_1 \in I$ then there is some $\iota = \iota_1 \circ \iota_0 \in I$ such that $x\texttt{[}\iota\texttt{]} = x\texttt{[}\iota_0\texttt{]}\texttt{[}\iota_1\texttt{]}$ for any schematic entity x.

We need to be able to produce "fresh" copies of schematic entities relative to any other finite set of entities. For this purpose we assume that there is a subset, $u \in II \subseteq I$, of invertible instantiations called renamings. We further assume that there is a sufficient supply of renamings so that any two finite sets X, Y of schematic entities can be renamed apart. That is, there is some renaming u such that $X[u]$ and Y are schematically separated.

Instantiation preserves satisfaction: if $\widetilde{c} \models c$, then $\widetilde{c}[\iota] \models c[\iota]$. An instantiation ι satisfies or solves a constraint c (written $\iota \models c$) if c instantiated by ι holds, i.e. $\models c[\iota]$, and similarly for sets of constraints. One of the most basic kinds of constraints in a system allowing schematic entities is matching or unification, i.e. equations between schematic entities. We require that this form of constraint be a part of all sequent systems. Thus, equations $s \sim s'$ between sequents are among the constraints, and instantiation propagates to the sequent terms – $(s \sim s')[\iota] = s[\iota] \sim s'[\iota]$. Also the usual laws for equality hold:

(reflexive) $\emptyset \models s \sim s$;

(transitive) $\{s \sim s', s' \sim s''\} \models s' \sim s''$;

(symmetric) $\{s \sim s'\} \models s' \sim s$.

Typically schematic entities are obtained by including schematic variables of various syntactic sorts among the basic syntactic entities from which others are generated. Then instantiations are just (finite) maps from schematic variables to syntactic entities. The axioms for instantiations are intended to capture this intuition without forcing this particular model. We say that s is *fully schematic* (i.e. a schematic variable) if for any s' there is some ι such that $s' = s[\iota]$. A sequent is said to be *ground* (non-schematic) if $s[\iota] = s$, for $\iota \in I$. Many sequent systems will have both fully schematic and ground sequents, but this is not required.

5.2. Examples

We give two examples of sequent systems – the first from natural deduction, and the second from NQTHM.

Example (ND sequent system): ND is a natural deduction system for classical first-order predicate logic [54]. $Ssys_{\mathrm{ND}}$ is the sequent system underlying our representation of natural deduction as a reasoning theory. Its sequents are pairs $\Gamma \vdash A$ consisting of a set of formulas, Γ, called assumptions, and a conclusion formula, A. Term and formula meta-variables of traditional presentations are reified as schematic variables. Terms and formulas are built in the usual way, starting from individual constants, schematic variables for terms and formulas, individual variables, function and predicate symbols, propositional connectives and quantifiers. $\Gamma \vdash A$ states that A is a consequence of Γ. Constraints include equations between expressions of the same syntactic sort. Additional constraints include predicates such as the binary predicate *Nofree*, which holds of a set of formulas Γ and a variable a just if a does not occur free in any of the formulas in Γ. Instantiations are finite maps from schematic variables to syntactic entities of appropriate sort, and instantiation application is the homomorphic lifting of these maps to terms, formulas, and other syntactic entities.

Example (NQTHM **sequent system):** In contrast to the ND sequent system, the sequent system underlying NQTHM, called $Ssys_{\mathrm{NQTHM}}$ from now on (see part III for more detail), includes a wide variety of data structures, needed because NQTHM contains many special purpose reasoning modules. For example, the typeset reasoning module uses structures called typeset alists, which associate sets of types (shells) to terms. The linear arithmetic module manipulates a polynomial data base. A polynomial contains a linear inequation, a set of literals that must hold for the polynomial to be valid, the set of literals used in obtaining the polynomial, and other heuristic information. In part III we present $Ssys_{\mathrm{NQTHM}}$ using six sorts of sequent, each corresponding to the form of assertions manipulated by one or more reasoning modules.

Here as an example we briefly introduce four of these sequent sorts.

All the reasoning in NQTHM is carried out within the context of an NQTHM theory, which is a sequence of events. Events include function and shell definitions, axiom declarations, prove-lemma requests, and instructions for using lemmas. Conjectures are given by the user to the prover in form of terms construed as booleans. Correspondingly in $Ssys_{\mathrm{NQTHM}}$ there is a sequent sort whose sequents have the form $h \vdash_{\mathrm{U}} t$, where h represents an NQTHM theory, t is a term construed as a boolean and the sign U has been added to distinguish this sort of sequent from other sorts of sequent used inside $Ssys_{\mathrm{NQTHM}}$. In NQTHM the top level reasoning processes manipulate clauses and sets of clauses. In $Ssys_{\mathrm{NQTHM}}$ this is represented using assertions of the form $h \vdash_{\mathrm{W}} \widetilde{cl}$ or $h \vdash_{\mathrm{P}} cl \rightarrow \widetilde{cl}$, where cl is a clause (disjunction of literals) and $\widetilde{cl}$ is a finite set of clauses. Reasoning performed by the NQTHM rewriting module, called rewriter, is carried out within a local context obtained by assuming false all literals of a clause except the one currently being rewritten. This information is stored both in typeset form and in polynomial form. Rewriting reasoning is represented in $Ssys_{\mathrm{NQTHM}}$ using assertions of the form $h \vdash_{\mathrm{R}} ti, pi; t \rightarrow_m \langle t', \ldots \rangle$, where ti, pi is the local context, ti is the typeset representation of the local context, pi is the polynomial representation of the local context, m records the mode of rewriting (it can be `I` or `B` depending on whether the corresponding rewriting step in NQTHM preserves equality or propositional equivalence) and "..." represents information propagated but not used by the rewriter.

These sequents are built from various structures representing theories, local context, rewriting formulas, and so on. Schematic entities and instantiations are obtained as usual by including schematic variables for each sort of entity. In addition to the syntactic equation constraints, there are a variety of predicates defined on syntactic entities. For example, $imp(p)$ holds if the polynomial p contains an impossible inequation (e.g. $2 \leq 0$) and $\mathrm{Ts}(ti, t_1) \cap \mathrm{Ts}(ti, t_2) \sim \emptyset$ holds if the typesets of t_1 and t_2 in the local context ti are disjoint.

6. Rules

As for sequent systems, only certain general features of rules are needed to describe the notions of reasoning structure and deduction, and the operations for constructing reasoning structures. In order to allow for provisional reasoning, we treat applicability constraints as constituents of rules. Thus a rule is a relation on tuples consisting of a non-empty sequence of sequents and a finite set of constraints. We require that rules be closed under instantiation. Typically such relations consist of tuples of sequents of the same general structure, but this is not required. A rule set is a finite set of rules each associated with a unique identifier. Such sets are conveniently thought of as finite maps from identifiers to rules. Mathematically, the use of rule sets is just a way of partitioning one rule into several parts and giving each part a name.

6.1. Definition

Let $Ssys = \langle S, C, \models, I, \_\texttt{[}\_\texttt{]}\rangle$ be a sequent system, and let Id be a set of identifiers. Then the set of rules $R \in \mathbf{Rule}[Ssys]$ over $Ssys$, and the set of rule sets, $\widetilde{r} \in \mathbf{Rset}[Ssys, Id]$ over $(Ssys, Id)$ are defined by

$$\begin{aligned}\mathbf{Rule}[Ssys] &= \{R \subseteq (S^* \times S \times \mathrm{P}_\omega(C)) \\ &\qquad \mid (\forall \langle \overline{s}, s, \widetilde{c}\rangle \in R)(\forall \iota \in I)(\langle \overline{s}, s, \widetilde{c}\rangle\texttt{[}\iota\texttt{]} \in R)\} \\ \mathbf{Rset}[Ssys, Id] &= [Id \xrightarrow{\mathrm{f}} \mathbf{Rule}[Ssys]]\end{aligned}$$

If $\widetilde{r} \in \mathbf{Rset}[Ssys, Id]$ and $id \in Id$ we say that $\langle \overline{s}, s, \widetilde{c}\rangle \in \widetilde{r}(id)$ is an *instance* of id with premisses, $\overline{s}$, conclusion, s, and applicability conditions, $\widetilde{c}$. We may write $\langle id, \overline{s}, s, \widetilde{c}\rangle \in \widetilde{r}$ for $\langle \overline{s}, s, \widetilde{c}\rangle \in \widetilde{r}(id)$, and say that $\langle id, \overline{s}, s, \widetilde{c}\rangle$ is an instance of $\widetilde{r}$. A *rule generator* is any subset rg of $S^* \times S \times \mathrm{P}_\omega(C)$. The rule generated by rg is the set $rg\texttt{[}I\texttt{]}$. An *n-ary rule* is a rule contained in $S^n \times S \times \mathrm{P}_\omega(C)$, i.e. a rule such that its instances have all the form $\langle \overline{s}, s, \widetilde{c}\rangle$ where $\overline{s}$ is a list of n sequents. Classical rules, e.g. the rules of ND, have a fixed number n of premisses and are presented with a schema. In our framework these rules correspond to n-ary rules whose generator is a singleton set (see example below). Our framework allows also the definition of more complex rules like rules with a variable numbers of premisses and rules with more variance in the structure of the premisses and conclusion.

6.2. Examples

We start with some simple examples of inference rules from ND, and then move to examples of increasing complexity.

Example (ND – propositional rules): We use the ND sequent system $Ssys_{\mathrm{ND}}$, defined in §5.2. The set Id_{ND} includes identifiers for the inference rules for assumption introduction, $\vee$ introduction, $\neg$ introduction and elimination, and for $\bot$.

$$Id_{\mathrm{ND}} \supset \{\mathrm{ASS}, \vee\mathrm{I_r}, \vee\mathrm{I_l}, \neg\mathrm{E}, \neg\mathrm{I}, \bot_\mathrm{c}, \bot_\mathrm{i}\}$$

The informal notation used for the corresponding inference rules is

$$\text{ASS} \quad \frac{}{\{A\} \vdash A}$$

$$\vee \mathrm{I_r} \quad \frac{\Gamma \vdash A_1}{\Gamma \vdash A_1 \vee A_2} \qquad\qquad \vee \mathrm{I_l} \quad \frac{\Gamma \vdash A_1}{\Gamma \vdash A_2 \vee A_1}$$

$$\neg\mathrm{E} \quad \frac{\Gamma_1 \vdash A \qquad \Gamma_2 \vdash \neg A}{\Gamma_1 \cup \Gamma_2 \vdash \bot} \qquad\qquad \neg\mathrm{I} \quad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}$$

$$\bot_{\mathrm{c}} \quad \frac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A} \qquad\qquad \bot_{\mathrm{i}} \quad \frac{\Gamma \vdash \bot}{\Gamma \vdash A}$$

where Γ_1, Γ_2 are schematic variables standing for (finite) sets of formulas (the assumptions), and A_1, A_2 are schematic variables standing for formulas. We will use this informal notation in the rest of the paper. The schemas like those above should be thought of as presenting the rule generators for the rules considered. For instance the schemas for ASS and $\neg$E present respectively the rule generators

$$\{\langle [], \{A\} \vdash A, \emptyset \rangle\}$$

$$\{\langle [\Gamma_1 \vdash A, \Gamma_2 \vdash \neg A], \Gamma_1 \cup \Gamma_2 \vdash \bot, \emptyset \rangle\}$$

Note that instances of these rules may contain schematic variables. (For example the generating element is an instance.) In our framework this corresponds to the possibility of representing schematic reasoning.

Example (ND – $\forall$ **introduction):** The propositional rules for ND can all be expressed without explicit constraints, since all of the restrictions on their applicability can be expressed schematically in the form of their premisses and conclusion (e.g., in the above example, $\neg$E can be applied only if the conclusion of the first premiss is the negation of the conclusion of the second). However this is not always the case. The rule for $\forall$ introduction requires an additional constraint on the occurrences of free variables. We add $\forall$I to the set of rule identifiers. The informal notation used to present the rule associated to $\forall$I is the following:

$$\forall\mathrm{I} \quad \frac{\Gamma \vdash A}{\Gamma \vdash (\forall x) A} \quad \text{if} \quad \mathit{NoFree}(\Gamma, x)$$

Recall that $\mathit{NoFree}(\Gamma, x)$ holds just if the variable x does not occur free in any of the assumptions in Γ. In this case the notation used above presents the rule generator

$$\{\langle [\Gamma \vdash A], \Gamma \vdash (\forall x) A, \{\mathit{NoFree}(\Gamma, x)\} \rangle\}$$

The presence in this rule generator of a non-empty constraint corresponds in our framework to the possibility of representing provisional reasoning.

In the definitions of sequent and inference rule we have made explicit the notions of constraint and constraint satisfaction but have not specified any particular constraint solving or checking mechanisms. This allows flexibility in the presentation of rules and (as we will see later) in the construction of derivations. It also allows flexibility in the structure of derivations in that we can hide as much or as little information in constraints as we choose. This is illustrated by the two examples below.

Example (A simple tautology rule): Many systems have tautological deciders implemented as primitive inference rules (for example FOL, GETFOL). We represent this situation by inference rules with constraints that can be checked by invoking the decider. We extend ND by adding a simple form of tautology rule, TAUT,

$$\text{TAUT} \quad \frac{}{\Gamma \vdash A} \quad \text{if} \quad \mathit{TautConseq}(\Gamma, A)$$

where $\mathit{TautConseq}(\Gamma, A)$ holds just if the formula A is a tautological consequence of Γ (cf. [58], p. 26).

An alternative to the black-box view of tautology checking is to include a tautology checker, TC, as a part of the reasoning theory. Thus we would add a new sort of sequent

$$\widehat{\Gamma} \vdash_{\mathrm{TC}} \widehat{A}$$

corresponding to the representation of formulas used by TC along with the inference rules upon which TC is based. We could then replace the above rule by:

$$\text{TAUT} \quad \frac{\widehat{\Gamma} \vdash_{\mathrm{TC}} \widehat{A}}{\Gamma \vdash A}$$

Example (The NQTHM typeset reasoning module): The typeset reasoning module of NQTHM is directly represented via typeset sequents and rules for typeset deduction in [22]. In part III an alternative approach is taken in which typeset reasoning is represented by adding typeset constraints to the inference rules introduced to describe rewriting and linear arithmetic reasoning. An example is the inference rule for equality reasoning used by the rewriter.

$$\mathtt{NE} \quad \frac{}{h \vdash_{\mathrm{R}} ti, pi;\ (\mathtt{EQUAL}\ t_1\ \ t_2) \rightarrow_{\mathtt{I}} \langle \mathtt{F}, \emptyset, \emptyset \rangle} \quad \text{if} \quad \mathrm{Ts}(ti, t_1) \cap \mathrm{Ts}(ti, t_2) \sim \emptyset$$

According to this rule, an equality is rewritten to `F` (meaning false) if the typesets of the two terms t_1 and t_2, are disjoint in the context ti. The mode `I`, subscripting the arrow in the conclusion of the rule, indicates that this rewriting step preserves term identity.

Our definitions do not make any assumption about the arity of inference rules. This allows us, among other things, to define variary inference rules with corresponding variary rule generators. Consider the two examples below.

Example (A variary tautology rule): The variary tautology rule nTAUT has as premisses a finite list of sequents $\mathit{\Gamma}_1 \vdash A_1, \ldots, \mathit{\Gamma}_n \vdash A_n$ and conclusion the sequent $\mathit{\Gamma}_1 \cup \ldots \cup \mathit{\Gamma}_n \vdash A$ under the constraint that A tautologically follows from $A_1 \wedge \ldots \wedge A_n$.

$$\text{nTAUT} \quad \frac{\begin{array}{c}\mathit{\Gamma}_1 \vdash A_1 \\ \vdots \\ \mathit{\Gamma}_n \vdash A_n\end{array}}{\mathit{\Gamma}_1 \cup \ldots \cup \mathit{\Gamma}_n \vdash A} \quad \text{if} \quad \mathit{TautConseq}(A_1 \wedge \ldots \wedge A_n, A)$$

The rule generator presented by the notation introduced above is the following:

$$\bigcup_{n \in \mathbf{Nat}} \{\langle [\mathit{\Gamma}_1 \vdash A_1, \ldots, \mathit{\Gamma}_n \vdash A_n], \mathit{\Gamma}_1 \cup \ldots \cup \mathit{\Gamma}_n \vdash A, \{\mathit{TautConseq}(A_1 \wedge \ldots \wedge A_n, A)\}\rangle\}$$

Example (An NQTHM **variary rule):** An example of a variary NQTHM rule is the rule introduced to represent the application of replacement rules during rewriting reasoning.

$$\texttt{rewR} \quad \frac{\begin{array}{l}h \vdash_{\mathrm{R}} ti, pi;\ l_1 \rightarrow_{\mathtt{B}} \langle \mathtt{T}, \ldots\rangle \\ \vdots \\ h \vdash_{\mathrm{R}} ti, pi;\ l_n \rightarrow_{\mathtt{B}} \langle \mathtt{T}, \ldots\rangle\end{array}}{h \vdash_{\mathrm{R}} ti, pi;\ t \rightarrow_m \langle t', \ldots\rangle} \quad \text{if} \quad \mathit{rrMatch}(h, [l_i]_{1\ldots n}, t, t', m)$$

where $\mathtt{T}$ is a constant meaning true and $\mathit{rrMatch}(h, [l_i]_{1\ldots n}, t, t', m)$ is the constraint that there is a replacement rule in h with hypotheses $l'_1, \ldots, l'_n$ and conclusion $t_l \rightarrow_m t_r$, and a substitution (instantiation) s such that $t = t_l\mathtt{[}s\mathtt{]}$, $t' = t_r\mathtt{[}s\mathtt{]}$, and $l_i = l'_i\mathtt{[}s\mathtt{]}$, for $1 \leq i \leq n$. This rule is discussed in more detail in §12.2 and §13.3.

7. Reasoning Theories

We think of a reasoning theory as presenting a formal system or theory. It specifies a set of sequents, and a set of rules.

7.1. Definition

A reasoning theory, Rth, is a structure

$$Rth = \langle Ssys, Id, \widetilde{r} \rangle$$

such that $Ssys$ is a sequent system, Id is a set of identifiers, and $\widetilde{r} \in \mathbf{Rset}[Ssys, Id]$ is a rule set.

7.2. Examples

The ND reasoning theory is an example of a (simple) reasoning theory.

Example (ND)**:**

$$Rth_{\mathrm{ND}} = \langle Ssys_{\mathrm{ND}}, Id_{\mathrm{ND}}, \widetilde{r}_{\mathrm{ND}} \rangle$$

where $Ssys_{\mathrm{ND}}$ is the sequent system described in §5. Id_{ND} includes the rule names given in the ND example in §6.2 and additional names for the remaining quantifier rules. $\widetilde{r}_{\mathrm{ND}}$ associates to each ND rule name the corresponding rule, as described in the ND example in §6.2.

7.3. Composing Reasoning Theories

As discussed in §3 and in §4.4, provers often integrate special purpose reasoning modules which in turn use their own data structures and inference strategies. The natural way to structure such provers using our framework is as the gluing together of separate reasoning theories using additional inference rules. To illustrate this idea we define a simple operation for gluing together a family of disjoint reasoning theories.

Let

$$Rth_1 = \langle Ssys_1, Id_1, \widetilde{r}_1 \rangle$$

$$\ldots$$

$$Rth_n = \langle Ssys_n, Id_n, \widetilde{r}_n \rangle$$

be disjoint reasoning theories, with $Ssys_i = \langle S_i, C_i, \models_i, I_i, \_[\_]_i \rangle$ and $\widetilde{r}_i \in \mathbf{Rset}[Ssys_i, Id_i]$ for $1 \leq i \leq n$. By disjointness we mean that the families of sets S_i, C_i, I_i, and Id_i for $1 \leq i \leq n$ are each pairwise disjoint. Thus $S_i \cap S_j = \emptyset$, for $1 \leq i \neq j \leq n$, etc.

The (disjoint) union, $Ssys$, of the sequent systems $Ssys_i$ for $1 \leq i \leq n$ is defined by

$$Ssys = \bigcup_{1 \leq i \leq n} Ssys_i = \langle S, C, \models, I, \_[\_] \rangle$$

$$
\begin{aligned}
S &= \bigcup_{1 \le i \le n} S_i \\
C &= \bigcup_{1 \le i \le n} C_i \cup \bigcup_{1 \le i \ne j \le n} \{ s \sim s' \mid s \in S_i, s' \in S_j \} \\
I &= I_1 \times \ldots \times I_n
\end{aligned}
$$

$\models$ and $\_[\_]$, the identity instantiation, and composition are defined as follows.

$$
\begin{aligned}
&\widetilde{c} \models c \quad \text{iff} \quad \widetilde{c} \cap C_i \models_i c \quad \text{if} \quad c \in C_i \\
&\widetilde{c} \not\models s \sim s' \quad \text{if} \quad s \in S_i \wedge s' \in S_j \wedge i \ne j \\
&x[\iota] = x[\iota \downarrow i]_i \quad \text{if} \quad x \in S_i \cup C_i \\
&\mathtt{idi} = \langle \mathtt{idi}_1, \ldots, \mathtt{idi}_n \rangle \\
&\iota \circ \iota' = \langle \iota \downarrow 1 \circ \iota' \downarrow 1, \ldots, \iota \downarrow n \circ \iota' \downarrow n \rangle
\end{aligned}
$$

where $\iota \downarrow j$ is the j-th element of the tuple ι. It is easy to check that $C, \models$ satisfy (ax), (mon), (cut), and that the other requirements for a sequent system are satisfied.

Let $Id = \bigcup_{1 \le i \le n} Id_i$, and let Id_B be a set of identifiers disjoint from Id. Let $\widetilde{r} = \bigcup_{1 \le i \le n} \widetilde{r}_i$ ($\widetilde{r}(id) = \widetilde{r}_i(id)$ if $id \in Id_i$), and let $\widetilde{r}_B \in \mathbf{Rset}[Ssys, Id_B]$ be a set of inference rules over the joined sequent system. The gluing of the Rth_i via $\widetilde{r}_B$ is defined by

$$
Rth = glueRth([Rth_1, \ldots, Rth_n], Id_B, \widetilde{r}_B) = \langle Ssys, Id \cup Id_B, \widetilde{r} \cup \widetilde{r}_B \rangle
$$

We say that Rth is a *composite reasoning theory*, with *components* Rth_i, and *glue* $Id_B, \widetilde{r}_B$. The elements of the rule sets $\widetilde{r}_i$ are called the *internal rules* (briefly i-rules) of Rth_i. The elements of $\widetilde{r}_B$ are called *bridge rules*.

In addition to the isolated components of a composite reasoning theory, it is also useful to consider these components combined with the bridge rules that link them to other components, i.e. (instances of) rules whose conclusion sequent belongs to that component. We call these open structures *reasoning theory fragments*, or simply fragments. Given a family of reasoning theories and a set of bridge rules as above, we define the i-th fragment $Frag_i$ as follows.

$$
Frag_i = \langle Ssys_i, Id_i \cup Id_B, \widetilde{r}_i \cup (\widetilde{r}_B \downarrow i) \rangle
$$

is a *fragment* (of Rth), where $\widetilde{r}_B \downarrow i$ is defined by

$$
(\widetilde{r}_B \downarrow i)(id) = \{ \langle \overline{s}, s, \widetilde{c} \rangle \mid \langle \overline{s}, s, \widetilde{c} \rangle \in \widetilde{r}_B(id) \wedge s \in S_i \}
$$

for $id \in Id_B$.

In general a fragment $Frag_i$ will not be a proper reasoning theory. This is because bridge rules may mention sequents (and constraints) not in $Ssys_i$. In fact in the examples of which we are aware bridge rules always have premisses and conclusions in different component sequent systems, although this is not a requirement. If we relax the disjointness requirement for the component theories additional structure will be required of constraints and instantiations in order to produce a composite reasoning theory, and there may be 'bridge' constraints in addition to sequent equations. Part III describes in detail how to structure NQTHM as a composite reasoning theory.

8. Reasoning Structures and Derivations

A reasoning theory determines a set of proof structure fragments that we call *reasoning structures*. Reasoning structures represent stages in the construction of proofs. The proof fragments represented by reasoning structures can be schematic and/or provisional. Certain reasoning structures are singled out that represent derivations and proofs in the traditional sense. This allows us to decouple the specification of derivability from the control strategies for constructing derivations, and gives greater flexibility for algorithm design and for definition of high-level control abstractions.

Reasoning structures provide two independent forms of flexibility: horizontal and vertical. Horizontal flexibility is flexibility in mode of proof construction, abstraction and reuse of derivations, and schematic reasoning. It comes from being able to stitch together fragments rather like a patchwork quilt and to incrementally refine schematic information. Vertical flexibility provides control over the level of immediately visible detail. It comes from nesting of reasoning structures and the ability to encapsulate a substructure into a nesting link, or open up a nesting link.

In this section, we let $Rth = \langle Ssys, Id, \widetilde{r}\,\rangle$ be an arbitrary but fixed reasoning theory. We let SN (sequent nodes) and LN (rule nodes) be two disjoint countable sets, used to construct reasoning structures.

We give the definition of reasoning structures in two steps. First we define basic reasoning structures. They provide the horizontal dimension of flexibility. Next we add the vertical dimension of flexibility. Finally we define derivations and proofs as reasoning structures satisfying certain additional conditions, and show that these restricted classes of structures can be easily mapped to standard tree-like proof structures.

8.1. Reasoning Structures

A reasoning structure, rs, is a labelled graph. The nodes of rs are partitioned into two sets: sequent nodes and link nodes. The edges of rs go from link nodes to sequent nodes or from sequent nodes to link nodes. For each link node there is a unique outgoing edge. The target sequent node is called the conclusion. The remaining (incoming) edges are ordered and the target sequent nodes are called the premisses.[2] Sequent nodes are labelled by sequents and link nodes are labelled by *justifications.* One kind of justification is a rule application – represented by a rule identifier and a set of constraints. We call link nodes with such justifications, *rule application links.* Another kind of justification is a 4-tuple consisting of a set of constraints, an instantiation map, a sequence of sequent nodes, and a reasoning structure. The instantiation map relates schematic variables of the nested structure to those of its containing structure. The sequent nodes are the nodes in the nested reasoning structure which correspond to the premiss and conclusion nodes of the labelled link node. We call link nodes with such justifications, *nesting links.* These are the only kinds of justification we consider for the present.

Basic reasoning structures over a reasoning theory Rth and nodes SN, LN are those with no nesting links.

Definition (Basic Reasoning Structures, $\mathbf{Rs}_0[Rth, SN, LN]$):

$\mathbf{Rs}_0[Rth, SN, LN]$ is the set of structures

$$rs = \langle Sn, Ln, g, sg, sL, lL\rangle$$

such that

(1) $Sn \in \mathrm{P}_\omega(SN)$ is the set of sequent nodes of rs, and $Ln \in \mathrm{P}_\omega(LN)$ is the set of link nodes of rs;

(2) $g : [Ln \to Sn]$ maps each link node to its associated goal sequent node;

(3) $sg : [Ln \to Sn^*]$ maps each link node to its (possibly empty) associated sequence of subgoal sequent nodes;

(4) $sL : [Sn \to S]$ is the sequent node labelling map;

(5) $lL : [Ln \to [Id \times \mathrm{P}_\omega(C)]]$ is the link node labelling map. This map must be such that for $ln \in Ln$ if $lL(ln) = \langle id, \widetilde{c}\rangle$, $\overline{s} = sL(sg(ln))$, and $s = sL(g(ln))$, then $\langle \overline{s}, s, \widetilde{c}'\rangle \in \widetilde{r}(id)$ for some $\widetilde{c}'$ such that $\widetilde{c} \models \widetilde{c}'$.

Definition ($Graph(rs)$): The directed graph, $Graph(rs)$, underlying a basic reasoning structure $rs = \langle Sn, Ln, g, sg, sL, lL\rangle$ is the graph with nodes $Sn \cup Ln$ and edges

$$\{(ln, g(ln)) \mid ln \in Ln\} \cup \{(sn, ln) \mid ln \in Ln \wedge sn \in sg(ln))\}.$$

Notice that a sequent node may be the conclusion or a premiss of more than one link node. This allows representation of multiple proof attempts for a given goal, and sharing of substructures.

Example (ND basic reasoning structure): Figure 1 gives a graphical representation of a reasoning structure. Circles represent sequent nodes, squares represent link nodes. Arrows go from premiss nodes to rule nodes, and from rule nodes to

[2] The directionality of the edges is simply a device to distinguish the premisses from the conclusion of a link node. We sometimes reverse the directions and correspondingly refer to the conclusion as the goal and the premisses as subgoals.

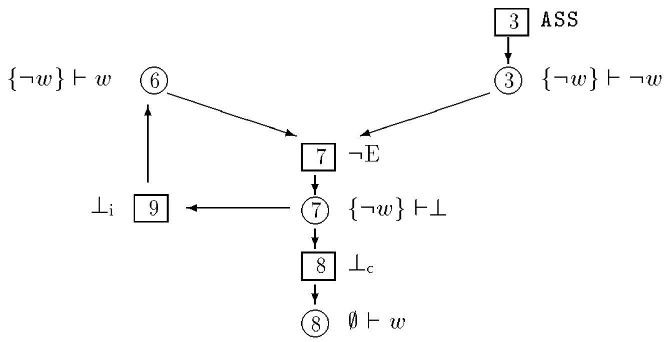

Figure 1. ND reasoning structure rs_1

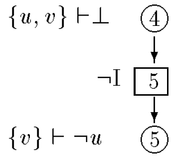

Figure 2. ND reasoning structure rs_2

conclusion nodes. The numbers inside are used to refer to node occurrences. Labels are put on the side of nodes. Rule application links whose justification contains an empty set of constraints are labelled only with a rule name. This reasoning structure has a cycle which corresponds in one direction to the application of $\perp_{\rm i}$ and, in the other direction, to the application of $\neg$E. Figure 2 gives another reasoning structure. These two reasoning structures can be stitched together to obtain the reasoning structure in Figure 3 (u, v and w are schematic variables for formulas). Notice that to obtain this result we have constructed a third reasoning structure deriving $\{A\} \vdash A \vee \neg A$, which we have then linked to the first two using link node 4. Notice also that sequent node 3 is the premiss of two different link nodes. This form of sharing corresponds to the introduction in two different places of the assumption $\neg(A \vee \neg A)$ in a traditional proof figure.

Part III gives examples of NQTHM reasoning structures spanning multiple heterogeneous reasoning theories.

We define general reasoning structures by allowing successively deeper levels of nesting, starting with basic reasoning structures at level 0.

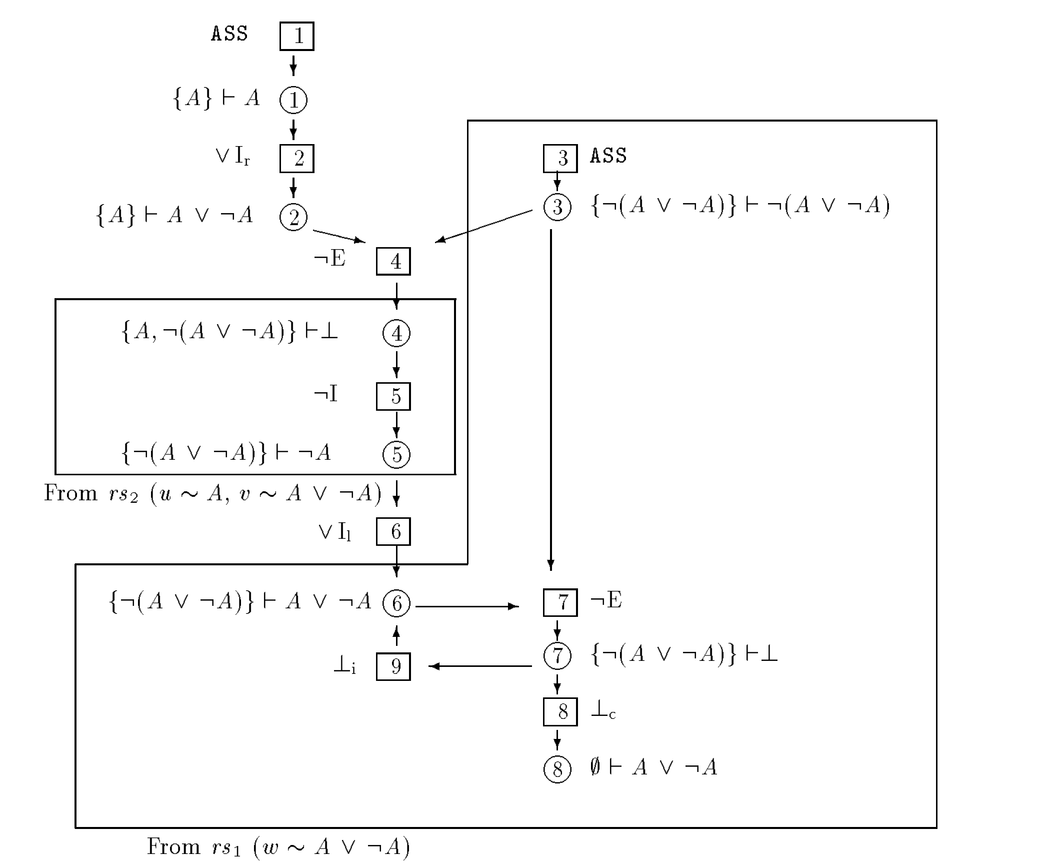

Figure 3. Stitching together rs_1 and rs_2.

Definition (Reasoning Structures, $\mathbf{Rs}[Rth, SN, LN]$): The set, $\mathbf{Rs}[Rth, SN, LN]$, of reasoning structures is defined as follows.

$$\mathbf{Rs}[Rth, SN, LN] = \bigcup_{n \in \mathbf{Nat}} \mathbf{Rs}_n[Rth, SN, LN]$$

where $\mathbf{Rs}_n[Rth, SN, LN]$ is the set of reasoning structures of level n. The reasoning structures of level 0 are the basic reasoning structures defined above. The reasoning structures of level $n+1$ are the structures

$$rs = \langle Sn, Ln, g, sg, sL, lL \rangle$$

such that conditions (1-4) in the definition of basic reasoning structures holds, and

(5_{n+1}) $lL : [Ln \to [Id \times \mathrm{P}_\omega(C)] + [\mathrm{P}_\omega(C) \times I \times [Sn^*, Sn] \times \mathbf{Rs}_n[Rth, SN, LN]]]$ such that if ln is a rule application link then condition (5) for basic reasoning structures holds and if ln is a nesting link with $lL(ln) = \langle \widetilde{c}, \iota, [\overline{sn}, sn], rs' \rangle$ and

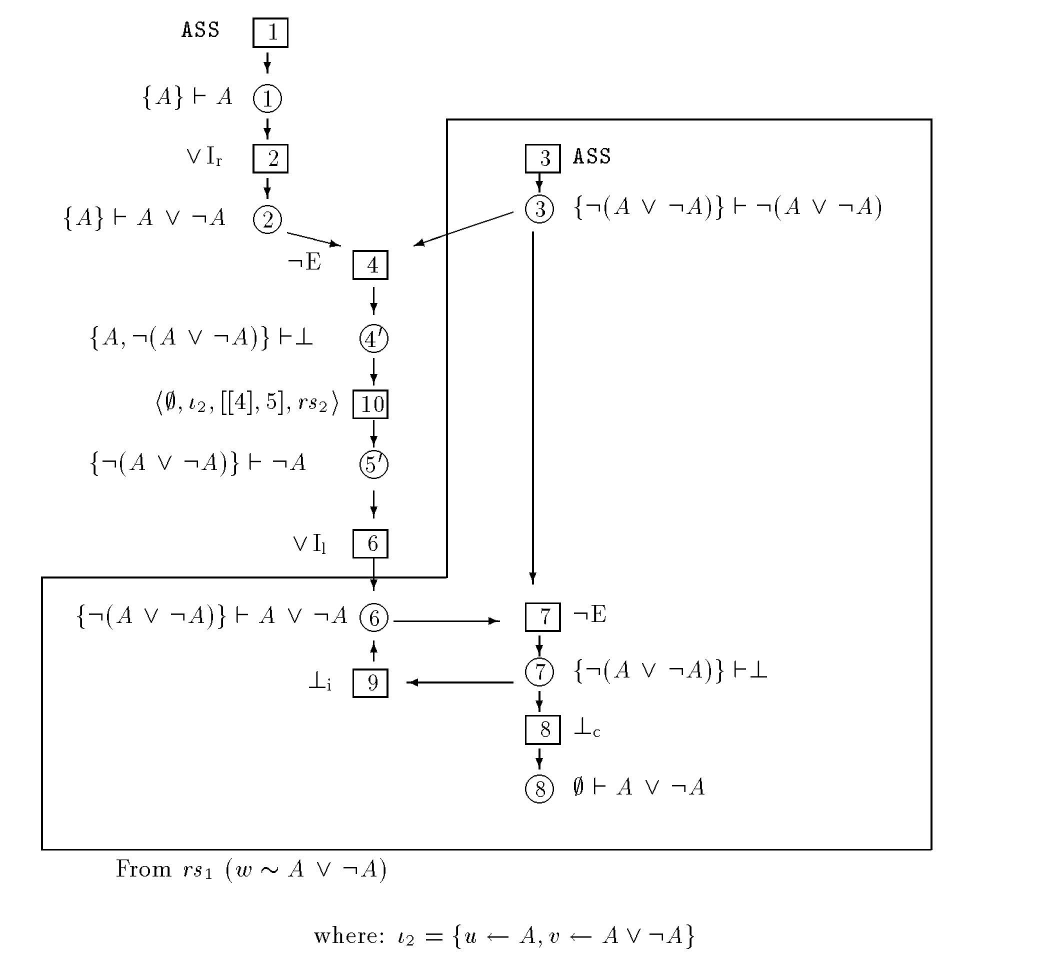

Figure 4. Nesting rs_2.

$rs' = \langle Sn', Ln', g', sg', sL', lL' \rangle$ then $[\overline{sn}, sn] \in (Sn')^*$, and $sL'([\overline{sn}, sn])[\iota] = [sL(sg(ln)), sL(g(ln))]$.

Example (ND nested reasoning structure): The reasoning structure in Figure 2 can be connected to the reasoning structure in Figure 1 by using a vertical nesting node rather than horizontal stitching. The result is the reasoning structure in Figure 4. The label of rule node 10 $\langle \emptyset, \iota_2, [[4], 5], rs_2 \rangle$ contains the empty set of constraints, the instantiation map ι_2, the premiss list [4] and conclusion 5, and the nested reasoning structure $rstr_2$. Thus the premiss sequent 4 of rs_2 is associated with the sequent node $4'$ of the outer structure, and the conclusion sequent 5 of rs_2 is associated with the sequent node $5'$ of the outer structure.

8.2. Derivations

In order to define the notion of derivation we need to define how instantiations are applied to reasoning structures.

Definition (instantiation, $rs\,[\iota]$): If $rs = \langle Sn, Ln, g, sg, sL, lL\rangle \in \mathbf{Rs}[Rth, SN, LN]$ and $\iota \in I$, then

$$rs\,[\iota] = \langle Sn, Ln, g, sg, sL[\iota]\,, lL'\rangle$$

where, for $ln \in Ln$

(ra) if $lL(ln) = \langle id, \widetilde{c}\rangle$, then $lL'(ln) = \langle id, \widetilde{c}\,[\iota]\,\rangle$,

(nest) if $lL(ln) = \langle \widetilde{c}\,, \iota_1, [\overline{sn}\,, sn], rs_1\rangle$, then $lL'(ln) = \langle \widetilde{c}\,[\iota]\,, \iota \circ \iota_1, [\overline{sn}\,, sn], rs_1\rangle$.

Intuitively a reasoning structure is a *derivation* of a conclusion sequent from a set of assumption sequents, if it represents a traditional proof figure. That is, if it satisfies conditions 1-5 below.

(1) Each rule application link has no unsolved constraints.

(2) Each sequent node is the conclusion of at most one inference (link node).

(3) There is a unique sequent node that does not occur as the premiss of any inference. The sequent labelling this node is the *conclusion* of the derivation. The sequents labelling occurrences which are not the conclusion of any inference are the *open assumptions.*

(4) The underlying graph is acyclic.

(5) For each nesting link, the associated tuple $\langle \widetilde{c}\,, \iota, [\overline{sn}\,, sn], rs\rangle$ is such that $\widetilde{c}$ is the empty set and the reasoning structure $rs\,[\iota]$ is a derivation with conclusion node sn and open assumption nodes $\overline{sn}$.

A reasoning structure is a *proof* if it is a derivation with no open assumptions. A sequent s is *Rth*-derivable from a set of sequents $\widetilde{s}$ if there exists a derivation $rs \in \mathbf{Rs}[Rth, SN, LN]$ with conclusion s and open assumptions contained in $\widetilde{s}$. A sequent s is *Rth*-provable if it is *Rth*-derivable from the empty set of sequents.

In general a derivation is a DAG (directed acyclic graph) rather than a tree, since we allow a sequent node to be a premiss of more than one inference. This allows not only the traditional sharing of lemmas, but even the sharing of subderivations. The next few lemmas state basic properties of derivations, and relate them to traditional proof figures. The first says that the underlying graph of a derivation is rooted in the conclusion. The second says that instantiation preserves the property of being a derivation. The third and fourth show that, for consideration only of derivability, nesting links and sharing of subderivations can be eliminated (vertical and horizontal unfolding). (Proofs can be found in §17.)

Lemma (reach): Let $rs = \langle Sn, Ln, g, sg, sL, lL\rangle$ be a derivation with conclusion node sn_0, then every sequent node is reachable from sn_0 by a chain of subgoal links, i.e. for each $sn \in Sn$ there is a sequence $[ln_j, sn_{j+1} \mid 0 \leq j < n]$ such that $g(ln_j) = sn_j$, $sn_{j+1} \in sg(ln_j)$ for $0 \leq j < n$, and $sn_n = sn$.

Lemma (derivation instantiation): If rs is a derivation of s from $\widetilde{s}$ then $rs[\iota]$ is a derivation of $s[\iota]$ from $\widetilde{s}\,[\iota]$ for any instantiation ι.

Lemma (elimination of nesting): If rs is a $\mathbf{Rs}[Rth, SN, LN]$ derivation of s from $\widetilde{s}$ then we can find a level 0 derivation $rs_0 \in \mathbf{Rs}_0[Rth, SN, LN]$ of s from $\widetilde{s}$.

Lemma (derivation trees): Let rs be a level 0 derivation of s from $\widetilde{s}$ then there exists a level 0 derivation rs' of s from $\widetilde{s}$ such that $Graph(rs')$ is a tree.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[A]^1}{A \vee \neg A} \qquad [\neg(A \vee \neg A)]^2}{\bot}}{\neg A}{}^{1}}{A \vee \neg A} \qquad [\neg(A \vee \neg A)]^2}{\bot}}{A \vee \neg A}{}^{2}$$

Figure 5. ND classical proof of $A \vee \neg A$

Example (ND proof): The reasoning structures in Figure 3 and Figure 4 both contain proofs of $A \vee \neg A$. These proofs are representations of the standard natural deduction style proof shown in Figure 5 (here assumptions discharged are put in square parentheses and discharging points are labelled by numbers).

9. Operations on Reasoning Structures

We start by giving a set of primitive operations for construction of basic reasoning structures and top level manipulation of nested structures. These are then extended uniformly to build nested reasoning structures. Finally we show how inference rule applications can be defined in terms of the primitive operations.

9.1. Primitive Operations

To construct basic reasoning structures we define one constant, `mtrs`, the empty reasoning structure, and four operations: add a sequent, `addS`; add a rule application link, `linkR`; solve constraints, `solveC`. To construct nested reasoning structures we add an operation, `linkN`, that adds a nesting link. We show that this set of operations is complete in the sense that all reasoning structures can be generated from the empty structure using only these operations.

Definition (empty reasoning structure, `mtrs`): The empty reasoning structure `mtrs` is the structure $\langle Sn, Ln, g, sg, sL, lL\rangle$ where Sn and Ln are the empty set and g, sg, sL, lL are the function with empty domain.

$$\mathtt{mtrs} = \langle \emptyset, \emptyset, \vec{\emptyset}, \vec{\emptyset}, \vec{\emptyset}, \vec{\emptyset}\rangle$$

Each operation, O, has two arguments: a reasoning structure occurrence and a tuple of additional parameters. The reasoning structure occurrence is either a reasoning structure (top level occurrence) or a reasoning structure together with a path consisting of a sequence of nesting links that selects a nested structure. In the following we let rs stand for a reasoning structure $\langle Sn, Ln, g, sg, sL, lL\rangle$. We begin by describing the tuple of arguments, A, appropriate for each operation, O, and the top level action $O(rs, A)$. We then show how this is uniformly lifted to nested substructures. [3]

Definition (adding a sequent, $\mathtt{addS}(rs, s)$): If $s \in S$, then $\mathtt{addS}(rs, s)$ is

$$\langle Sn \cup \{sn\}, Ln, g, sg, sL\{sn \mapsto s\}, lL\rangle$$

where $sn \in SN - Sn$.

Note that this and other operations are only functions modulo choice of new nodes. This can be fixed in various ways. We will leave it informal for now and in the following we will assume that two reasoning structures are equal if they are equal modulo node renaming.

Definition (rule linking, $\mathtt{linkR}(rs, \overline{sn}, sn, r)$): If $sn \in Sn$, $\overline{sn} \in Sn^*$, $r = \langle id, \overline{s}, s, \widetilde{c}\rangle \in \widetilde{r}$, $\models \{\overline{s} \sim sL(\overline{sn}), s \sim sL(sn)\}$, then $\mathtt{linkR}(rs, \overline{sn}, sn, r)$ is

$$\langle Sn, Ln \cup \{ln\}, g\{ln \mapsto sn\}, sg\{ln \mapsto \overline{sn}\}, sL, lL\{ln \mapsto \langle id, \widetilde{c}\rangle\}\rangle$$

where $ln \in LN - Ln$.[4]

---

[3] Note that we write $O(rs, a_1, \ldots, a_n)$ rather than $O(rs, \langle a_1, \ldots, a_n\rangle)$, and we use the same name O for top level and nested application.

[4] In this paper, we require that rule node labels in reasoning structures satisfy condition (5) given in §8, that is:

(5) $lL : [Ln \to [Id \times \mathrm{P}_\omega(C)]]$ such that for $ln \in Ln$ if $lL(ln) = \langle id, \widetilde{c}\rangle$, $\overline{s} = sL(sg(ln))$, and $s = sL(g(ln))$, then $\langle \overline{s}, s, \widetilde{c}\,'\rangle \in \widetilde{r}(id)$ for some $\widetilde{c}\,'$ such that $\widetilde{c} \models \widetilde{c}\,'$.

In an early version of this paper, we considered an apparently more flexible alternative (5'):

(5') $lL : [Ln \to [Id \times \mathrm{P}_\omega(C)]]$ such that for $ln \in Ln$ if $lL(ln) = \langle id, \widetilde{c}\rangle$, $\overline{s} = sL(sg(ln))$, and $s = sL(g(ln))$, then for any $\iota \in I$ such that $\models \widetilde{c}\,[\iota]$ there is some $\widetilde{c}_0$ such that $\models \widetilde{c}_0$ and $\langle \overline{s}\,[\iota], s\,[\iota], \widetilde{c}_0\rangle \in \widetilde{r}(id)$.

and the following definition of primitive rule linking

Definition (rule linking, $\mathtt{linkR'}(rs, \overline{sn}, sn, r)$): If $sn \in Sn$, $\overline{sn} \in Sn^*$, $\overline{s}, s, \widetilde{c}$ is schematically separated from rs (i.e. from the sequents and constraints occurring in node labels), and $r =$

Note that $\overline{sn}$ can be a sequence with repetitions. An example in which this possibility can be useful, is application of conjunction introduction to obtain $A \wedge A$ from A. Depending on circumstances, one may or may not want to identify the derivations of two occurrences of A in the premiss. If sharing is desired then the linking uses two occurrences of the same sequent node in the premiss list.

Definition (constraint solving, $\mathtt{solveC}(rs, ln, \widetilde{c}\,')$): If $ln \in Ln$, $lL(ln) = \langle id, \widetilde{c}\,\rangle$, and $\widetilde{c}\,' \models \widetilde{c}$, then

$$\mathtt{solveC}(rs, ln, \widetilde{c}\,') = \langle Sn, Ln, g, sg, sL, lL\{ln \mapsto \langle id, \widetilde{c}\,'\rangle\}\rangle$$

and if $ln \in Ln$, $lL(ln) = \langle \widetilde{c}, \iota, [\overline{sn}, sn], rs_1\rangle$, and $\widetilde{c}\,' \models \widetilde{c}$, then

$$\mathtt{solveC}(rs, ln, \widetilde{c}\,') = \langle Sn, Ln, g, sg, sL, lL\{ln \mapsto \langle \widetilde{c}\,', \iota, [\overline{sn}, sn], rs_1\rangle\}\rangle$$

Note that we can use `solveC` to add constraints to a link node as well as to eliminate solved constraints.

Nesting links are introduced with minimal structure justifications, just the required sequent nodes, their corresponding sequent labels and the instantiation to put in the link. These can then be extended by applying operations to nested substructures.

Definition (nesting link, $\mathtt{linkN}(rs, \overline{sn}, sn_0, \iota, \overline{s}, s_0)$): If $sn_0 \in Sn$, $\overline{sn} = [sn_1, \ldots sn_n] \in Sn^*$, $s_0 \in S$, $\overline{s} = [s_1, \ldots s_n] \in S^*$, and $s_j[\iota] = sL(sn_j)$ for $0 \leq j \leq n$, then $\mathtt{linkN}(rs, \overline{sn}, sn_0, \iota, \overline{s}, s_0)$ is

$$\langle Sn, Ln \cup \{ln\}, g\{ln \mapsto sn_0\}, sg\{ln \mapsto \overline{sn}\}, sL, lL\{ln \mapsto \langle \emptyset, \iota, [\overline{sn}\,', sn'_0], rs_0\rangle\}\rangle$$

where $ln \in LN - Ln$, $Sn_0 = \{sn'_0, sn'_1, \ldots sn'_n\} \subset SN - Sn$ (distinct fresh sequent nodes), $\overline{sn}\,' = [sn'_1, \ldots sn'_n]$, $sL_0(sn'_j) = s_j$ for $0 \leq j \leq n$, and $rs_0 = \langle Sn_0, \emptyset, \emptyset, \emptyset, sL_0, \emptyset\rangle$.

To define the general application of operations on reasoning structures we first define the set of paths of a reasoning structure. Each path selects a nested substructure. We represent paths as sequences of nesting link nodes.

Definition (path in reasoning structure): $\overline{ln} \in LN^*$ is a path in a reasoning structure $rs = \langle Sn, Ln, g, sg, sL, lL\rangle$ selecting rs' iff one the following conditions is satisfied:

$\langle id, \overline{s}, s, \widetilde{c}\,\rangle \in \widetilde{r}$, then $\mathtt{linkR}'(rs, \overline{sn}, sn, r)$ is

$$\langle Sn, Ln \cup \{ln\}, g\{ln \mapsto sn\}, sg\{ln \mapsto \overline{sn}\}, sL, lL\{ln \mapsto \langle id, \widetilde{c}^{\,\dagger}\rangle\}\rangle$$

where $ln \in LN - Ln$, and $\widetilde{c}^{\,\dagger} = \widetilde{c} \cup \{\overline{s} \sim sL(\overline{sn}), s \sim sL(sn)\}$.

This option was rejected because it is not possible in general to construct all reasoning structures using the primitive operations – since for a given link node there may be no one rule instance that covers all satisfiable instances of the constraints.

(1) $\overline{ln} = [\,]$, and $rs' = rs$; or

(2) $\overline{ln} = [ln] \diamond \overline{ln}\,'$, where $ln \in Ln$, $lL(ln) = \langle \tilde{c}, \iota, [\overline{sn}, sn], rs_1 \rangle$, and $\overline{ln}\,'$ is a path in rs_1 selecting rs'.

Note that [ ] is the only path in a basic reasoning structure.

Definition (General application of operations): If O is one of the primitive operations, $rs = \langle Sn, Ln, g, sg, sL, lL \rangle$ is a reasoning structure, $\overline{ln}$ is a path in rs, and A is a tuple of arguments appropriate for O and the nested structure selected by $\overline{ln}$ in rs, then we define $O(\langle rs, \overline{ln} \rangle, A)$ by induction on $\overline{ln}$ as follows:

(mt) if $\overline{ln} = [\,]$, then $O(\langle rs, \overline{ln} \rangle, A) = O(rs, A)$

(nmt) if $\overline{ln} = [ln] \diamond \overline{ln}_1$, $lL(ln) = \langle \tilde{c}, \iota, [\overline{sn}, sn], rs_1 \rangle$, then

$$O(\langle rs, \overline{ln} \rangle, A) = \langle Sn, Ln, g, sg, sL, lL\{ln \mapsto \langle \tilde{c}, \iota, [\overline{sn}, sn], O(\langle rs_1, \overline{ln}_1 \rangle, A) \rangle\} \rangle$$

The reasoning structure operations presented above are sound and complete in the sense made precise by the following two theorems. (Proofs can be found in §17.)

Theorem (soundness):

(1) `mtrs` is a basic reasoning structure.

(2) The operations `addS`, `linkR`, `solveC`, and `linkN` all map reasoning structures to reasoning structures (when applied to appropriate arguments).

(3) The operations `addS`, `linkR`, and `solveC` all map basic reasoning structures to basic reasoning structures (when applied to appropriate arguments).

Theorem (completeness): If $rs \in \mathbf{Rs}[Rth, SN, LN]$, then rs can be constructed from the empty reasoning structure using only the operations `addS`, `linkR`, `solveC`, and `linkN`. The basic reasoning structures are generated by excluding `linkN`.

Theorem (independence): (**completeness**) fails if any of the operations in the list are omitted.

9.2. Inference Rules as Operations on Reasoning Structures

The soundness and completeness results given in §9.1 guarantee that the primitive operations defined there allow us to construct any given reasoning structure from the empty structure. More work is necessary to guarantee that other maps on reasoning structures can be defined by simple compositions of these primitives.

Rules of a reasoning theory are relations that don't impose any directionality of application. Thus there is complete flexibility in their use to construct reasoning structures. They can be used for forward or backward chaining, or in various mixed modes to hook together derivation fragments. In this subsection we show how operations on reasoning structures corresponding to different modes of application of inference rules can be defined as simple compositions of the primitive operations.

In particular we define forward and backward application of an inference rule. We define these operations for basic reasoning structures. These can be lifted uniformly to nested reasoning structures in the same manner as primitive operations are lifted. Let $r = \langle id, \overline{s}, s_0, \widetilde{c}\,\rangle$ be a rule instance, with $\overline{s} = [s_1, \ldots, s_n]$ such that $\models \widetilde{c}$.

Definition (Forward application of r): Let $rs = \langle Sn, Ln, g, sg, sL, lL\rangle$ be a reasoning structure with $\overline{sn} = [sn_1, \ldots, sn_n] \in Sn^*$, and $sL(sn_j) = s_j$ for $1 \leq j \leq n$. Assume $sn_0 \notin Sn$ and $ln_0 \notin Ln$. Then $\mathtt{fwdR}(rs, r, \overline{sn}) = rs'$, where rs' is obtained by the following sequence of primitive operations:

(add sequent node) $rs_0 = \mathtt{addS}(rs, s_0)$ introducing sn_0;

(add link node) $rs_1 = \mathtt{linkR}(rs_0, \overline{sn}, sn_0, r)$ introducing ln_0; and

(solve constraints) $rs' = \mathtt{solveC}(rs_1, ln_0, \emptyset)$.

Definition (Backward application of r): Let $rs_0 = \langle Sn, Ln, g, sg, sL, lL\rangle$ be a reasoning structure, $sn_0 \in Sn$, $sL(sn_0) = s_0$. Assume $\overline{sn} = [sn_1, \ldots, sn_n]$ with $sn_j \notin Sn$ for $1 \leq j \leq n$, and $ln_0 \notin Ln$. Then $\mathtt{bkwdR}(rs_0, r, s_0) = rs'$, where rs' is obtained by the following sequence of primitive operations:

(add sequent nodes) rs_n is defined by $rs_{i+1} = \mathtt{addS}(rs_i, s_{i+1})$, introducing sn_{i+1} for $0 \leq i < n$;

(add link node) $rs_{n+1} = \mathtt{linkR}(rs_n, \overline{sn}, sn_0, r)$ introducing ln_0; and

(solve constraints) $rs' = \mathtt{solveC}(rs_{n+1}, ln_0, \emptyset)$.

We have defined forward and backward rule applications as partial operations whose domain consists of rule instances in which the constraints are satisfied. They correspond to traditional operations for constructing derivations. In our framework it is also possible to define lazy forms of rule application that allow us to postpone solving constraints, and to define mixed mode applications of inference rules, i.e. applications where nodes labelled by some subset of the premisses and conclusion sequents are present, and nodes labelled by the remainder are created. An interesting particular case is when no new node is added and the application of an inference rule consists only of adding a link and the constraints. In this case an inference rule application amounts to an application of `linkR`.

Example (ND rule linking): One example of a linking application is the stitching together of the reasoning structures in Figure 1 and Figure 2 to obtain the reasoning structure in Figure 3. The link node ln_6 is added to "link" together the sequent nodes sn_5 (the premiss of the rule application) and sn_6 (the conclusion). The rule instance r used in this case is

$$\langle \vee\mathrm{I}_1, [\{\neg(A \vee \neg A)\} \vdash \neg A], \{\neg(A \vee \neg A)\} \vdash A \vee \neg A, \emptyset\rangle.$$

Notice that, even if an inference rule application is always directional, a proof or derivation can be "read" out of a reasoning structure according to the desired mode of application. Reasoning structures do not include information about how they have been constructed. This information could be introduced as rule node annotations.

10. The NQTHM theorem prover

NQTHM is a sophisticated prover with complex reasoning heuristics, data structures and algorithms designed for efficient representation and processing of various sorts of information, and coded for optimal performance. The logic of NQTHM is a quantifier-free first-order logic of tree structured data and functions defined by recursion on well-founded orderings. The search for a proof in NQTHM is driven by heuristics (called processes) that integrate several proof techniques: simplification; destructor elimination; cross-fertilization; generalization; elimination of irrelevance; and induction. Each process maps a formula in clausal form to a set of clauses (considered conjunctively) and can be considered as a derived rule of inference that runs backwards (cf. p. 129 [10]).

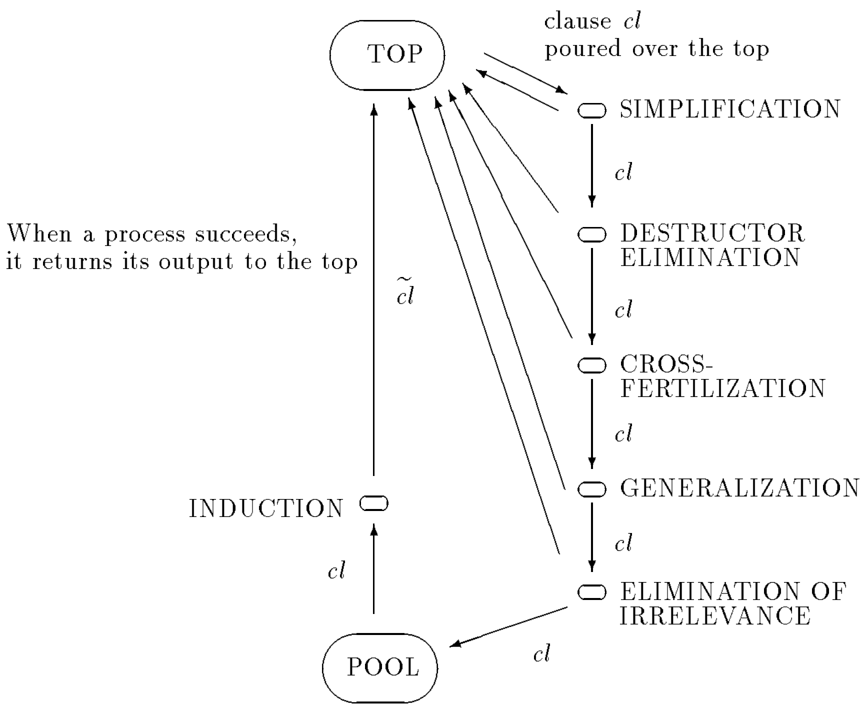

Figure 6. The NQTHM waterfall

The top level control can be described in analogy to a waterfall. (See Figure 6.) Conjectures to be proved are converted to clausal form and poured in at the top. Repeatedly, a clause is removed from the top and poured over the waterfall, until none remain. A clause is poured over the waterfall by trying on it, in order, simplification, destructor elimination, cross fertilization, generalization and elimination of irrelevance. If some process succeeds, it returns a set of clauses. These clauses are put into the top part (shades of Escher). If none succeeds, the clause is put into the pool. When top is empty the pool is cleaned up, i.e. the subsumed clauses are deleted. Then a clause is selected from the pool for induction, an induction schema is applied and the result is poured in at the top. When there is no clause left both in the top and in the pool, the initial conjecture has been proved.

A theory is built up in NQTHM by processing a sequence of events (called a history). Events include function and shell definitions, axioms, and proofs of lemmas. Shell definition is the NQTHM mechanism for defining finite recursive data types. A history also includes information describing the use of lemmas by various heuristics. For example some lemmas are tagged as rewrite rules for use by the rewriter.

11. Outline of our analysis of NQTHM

The original prover, called from now on pNQTHM, did not include any special arithmetic reasoning capability. In [8] the integration of a linear arithmetic module into pNQTHM and the interactions of the rewriter and linear arithmetic modules within the simplification process are described in detail. In the following sections we represent this integration at the reasoning theory level, describing the corresponding modifications to the pNQTHM reasoning theory. We have chosen this example because of the significance of the system (NQTHM is state-of-the-art, and a better understanding of how it works is by itself of considerable interest), because it is thoroughly documented, and it constitutes one of the most challenging case studies we could think of.

One of the main difficulties in the integration of a new module into a tightly coded system like NQTHM is that the existing procedures must be modified to generate, manipulate and propagate the information needed or generated by the new module. For example, in the case of integration of linear arithmetic, the local context information is represented in two ways: as typeset information and as polynomial information. In addition, the linear arithmetic module generates additional assumptions and dependency information that the rewriter must propagate. One of our main goals here is to show how the extra information and modifications can be isolated inside the definition of the sequent system and rules of the modified system. The methodology we use is the following:

(1) Specification of the original system;

(2) Specification of the module to be added;

(3) Refinement of the specification of the original system to incorporate the additional information passed to and from the new module.

(4) "Gluing together" of the new module and the modified system, which might require the addition of new bridge rules.

We focus on the simplification process and only sketch our analysis. In particular we describe sequents informally and only present some (of the many) internal and bridge rules. A more complete specification, based on the description given in [8] and examination of the NQTHM code, is given in [22].

We conclude our analysis of NQTHM by giving some example reasoning structures representing NQTHM deductions. We do this in order to give more realistic examples than the natural deduction examples in part II, and also to suggest how this methodology can be applied to provide NQTHM with the (presently missing) capability of producing proof structures.

We stress that of course we do not mean that the reasoning theory level of integration is all (or even most of) the work which must be done to complete the integration of a new reasoning module. The code of a highly tuned implementation must be modified according to the transformation described at the reasoning theory level. This means at least specifying the conditions under which a new module should be invoked, the rule that justifies the invocation, and the mode of rule application. For instance, in the case of integration of linear arithmetic into NQTHM this means adding clauses to the rewriter and other modules, allowing them to invoke the linear arithmetic module using appropriate bridge rules. Much work needs to be done in order for this to be truly systematic, much less automatic. However, even before that is possible, it seems likely that there is something to be gained from this level of specification in documenting decisions and guiding hand recoding. One further caveat is that some experience is needed to determine the kinds of specifications that are amenable to such refinements and extensions.

12. The Reasoning Theory Rth_{pNQTHM}

The inference modules of the simplification process of pNQTHM are the typeset specialist, the rewriter and the sweeping process (sweeper). The typeset specialist computes the type information associated to a term, e.g. that a term is a number or a list, under some type assumptions on terms (hereinafter we refer to such assumptions as typeset information). The rewriter rewrites an input term by applying rewrite rules obtained from axioms and lemmas contained in the current theory and unfolding function symbols. The rewriter performs these tasks in a context containing many kinds of information, the most important for our analysis being typeset information. The sweeper is the module which interacts with the rewriter in order

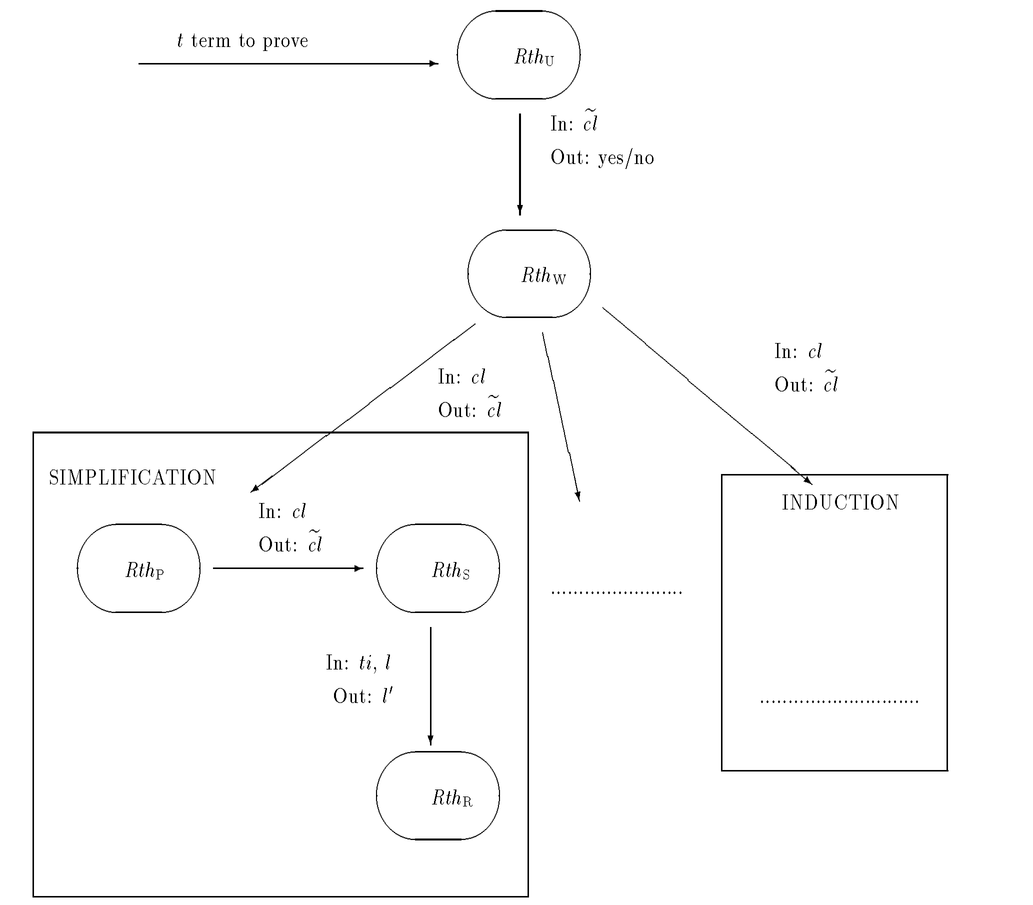

Figure 7. The Reasoning Theory Rth_{pNQTHM}

to rewrite (sweep) all the literals of the clause given in input to the simplification process.

In order to make the module structure explicit, we present the pNQTHM reasoning theory

$$Rth_{\text{pNQTHM}} = \langle Ssys_{\text{pNQTHM}}, Id_{\text{pNQTHM}}, \widetilde{r}_{\text{pNQTHM}} \rangle$$

as a composite reasoning theory constructed out of five component reasoning theories

$$Rth_{\text{pNQTHM}} = glueRth([Rth_U, Rth_W, Rth_P, Rth_S, Rth_R], Id_B, \widetilde{r}_B)$$

where each component reasoning theory corresponds to a reasoning module of pNQTHM, and $\widetilde{r}_B$ is a set of bridge rules (partially) described below. Figure 7 provides an overview of the organization of Rth_{pNQTHM}. Rth_U corresponds to (the reasoning theory view of) the module in charge of interaction with the user. Rth_W corresponds to the master module of the waterfall, the module which controls the interactions with the modules implementing the various waterfall processes. Rth_{P} corresponds to the master module of the simplification process. Finally Rth_{S} and Rth_{R} correspond respectively to the sweeper and the rewriter modules. The arrows in Figure 7 between component reasoning theories correspond in our framework to bridge rules of Rth_{pNQTHM}. Bridge rules describe interactions between inference modules of pNQTHM. Arrows of this kind are labelled by the data structures passed (In) and returned (Out) in the interaction (see below).

The reasoning theory level description of pNQTHM can be done in many ways, and at many levels of detail. The description given here is aimed at understanding the interactions among the modules of the simplification process. We do not consider therefore the other waterfall processes. Moreover, since typeset reasoning does not interact with other modules, we treat the typeset reasoning module as a collection of primitive operations on typeset information structures and do not formulate a reasoning theory for this module. The context and data structures manipulated by NQTHM contain additional (non-logical) information used to control the heuristic proof strategies. We have omitted this information for the present, as we are not treating issues of control in this paper.

12.1. The Rth_{pNQTHM} Sequent System

The Rth_{pNQTHM} sequent system

$$Ssys_{\mathrm{pNQTHM}} = \langle S_{\mathrm{pNQTHM}}, C_{\mathrm{pNQTHM}}, \models_{\mathrm{pNQTHM}}, I_{\mathrm{pNQTHM}}, \_[\_]_{\mathrm{pNQTHM}} \rangle$$

is defined as

$$Ssys_{\mathrm{pNQTHM}} = \bigcup_{i \in \{\mathrm{U,W,P,R,S}\}} Ssys_i$$

where each component sequent system has its own sequents, and constraints. There are schematic variables for the various sequent sorts and other syntactic sorts, such as terms and clauses. Instantiations are finite maps (substitutions) from schematic variables to syntactic entities of the appropriate sort, and application is as usual. In pNQTHM all reasoning takes place in the context of some pNQTHM theory. At the user level, assertions are conjectures to be proved within this context. At the waterfall level processes manipulate clauses and clause sets. Internal to the simplification process, attention is focused on a single term occurring somewhere in the clause being simplified. This term might be one of the literals of the clause, and hence considered to denote a boolean, or it might be a proper subterm of some literal and hence considered to denote an individual. Often additional assumptions can be made when reasoning about a term, based on its location in a clause. Such assumptions are kept in a local context. Thus we structure pNQTHM sequents as pairs consisting of an pNQTHM theory h and a current conjecture. The conjecture may be further decomposed into local context information and focus information. The notation for these sequents is summarized below.

(U)	$h \vdash_{\mathrm{U}} t$	(W)	$h \vdash_{\mathrm{W}} \widetilde{cl}$
(P)	$h \vdash_{\mathrm{P}} cl \rightarrow \widetilde{cl}$	(S)	$h \vdash_{\mathrm{S}} \widetilde{cl} \rightarrow \widetilde{cl}\,'$
(R)	$h \vdash_{\mathrm{R}} ti;\ t \rightarrow_m t'$		

Turnstyles are labelled to keep track of the component sequent system. For the sort U, the conjecture is a term, t, representing the formula $t \neq \mathtt{F}$ (i.e. viewed as a boolean). A sequent of sort U, asserts that the formula represented by t is a consequence of the theory h. For the sort W, the conjecture is a set of clauses $\widetilde{cl}$ (considered conjunctively). A clause is a set of literals (considered disjunctively), and a literal is a term viewed as a boolean. The interpretation of sequents of sort W is analogous to those of sort U. For the sort P, the conjecture is $cl \rightarrow \widetilde{cl}$ where cl is a clause (corresponding to the input to the process) and $\widetilde{cl}$ is a set of clauses (corresponding to the output of the process). A P-sequent asserts that cl is a consequence of the theory h, under the additional assumptions $\widetilde{cl}$. For the sort S, the conjecture is $\widetilde{cl} \rightarrow \widetilde{cl}\,'$. An S-sequent asserts that $\widetilde{cl}$ is equivalent to $\widetilde{cl}\,'$ in h. In pNQTHM when a clause is simplified, each literal is rewritten in a local context containing the typeset information that the remaining literals are false. We represent this by using a typeset information structure, ti, as the local context of an R-sequent. The focus information of an R-sequent is a triple $t \rightarrow_m t'$ where t and t' are terms and m is a mode (`B` if t is to be viewed as a boolean, `I` if t is to be viewed as an individual). An R-sequent asserts that t is equivalent (relative to the mode) to t' in the theory h (under the additional assumptions contained in ti).

12.2. The Rth_{pNQTHM} Rules

To provide some intuition for each rule we explain the reasoning step of pN-QTHM that the rule is intended to describe. Notice that we present these rules using the classical forward form, i.e. from the premisses to the conclusion, even though the rule is adirectional and their intended use in the prover is backward.

The W-rule `qed` expresses the fact that there are no clauses (subgoals) left both in the pool and in the top of the waterfall. It means that the initial conjecture has been proved.

$$\texttt{qed}\quad \frac{}{h \vdash_{\mathrm{W}} \emptyset}$$

The R-rule `rewR` represents the application of a replacement rule, i.e. a rewriting rule obtained from an axiom or a lemma in the current theory, to a term. The replacement rules of a theory are pairs of the form $\langle \wedge_{1\leq i\leq n} l_i', t_l \rightarrow_m t_r \rangle$, where

$\wedge_{1 \le i \le n} l'_i$ are the hypotheses of the rule and $t_l \rightarrow_m t_r$ is the conclusion. Logically speaking a replacement rule represents the formula

$$\wedge_{1 \le i \le n} l'_i \Rightarrow (t_l \; eq_m \; t_r)$$

where eq_m is $=$, if m is $\mathtt{I}$, and $\Leftrightarrow$, if m is $\mathtt{B}$.

$$\texttt{rewR} \quad \frac{[h \vdash_{\mathrm{R}} ti;\; l_i \rightarrow_{\mathtt{B}} \mathtt{T} \mid 1 \le i \le n]}{h \vdash_{\mathrm{R}} ti;\; t \rightarrow_m t'} \quad \text{if} \quad rrMatch(h, [l_i]_{1\ldots n}, t, t', m)$$

where $\mathtt{T}$ is a constant used for denoting truth and $rrMatch(h, [l_i]_{1\ldots n}, t, t', m)$ is the replacement rule matching constraint. This holds if there is a replacement rule in h with hypotheses $\wedge_{1 \le i \le n} l'_i$ and conclusion $t_l \rightarrow_m t_r$; and a substitution, s, that matches t_l to t, t_r to t', and l'_i to l_i, for $1 \le i \le n$.

The bridge rule **clausify** represents a call of the master module of the waterfall. This invocation is done with the goal of putting the clausification of the initial conjecture at the top set of clauses of the waterfall (arrow from Rth_{U} to Rth_{W} in Figure 7).

$$\texttt{clausify} \quad \frac{h \vdash_{\mathrm{W}} preprocess(h, t)}{h \vdash_{\mathrm{U}} t}$$

where $preprocess(h, t)$ expands abbreviations (recorded in h) and converts the result to clausal form.

The bridge rule **select** represents the fact that, in the waterfall process, clauses in the top-pool are selected and then fed into the special purpose deduction modules (arrow from Rth_{W} to Rth_{P} in Figure 7). The sequents of the reasoning theories associated to the high-level processes have all the same form of P-sequents, as all of them take as input a clause and return a set of clauses.

$$\texttt{select} \quad \frac{h \vdash_{\mathrm{P}} cl \rightarrow \widetilde{cl}_1 \qquad h \vdash_{\mathrm{W}} \widetilde{cl} \cup \widetilde{cl}_1}{h \vdash_{\mathrm{W}} \widetilde{cl} \cup \{cl\}}$$

The bridge rule **swInit** represents the call of the sweeper by the master module of the simplification process. The goal of this call is the rewriting of each literal of a clause (arrow from Rth_{P} to Rth_{S} in Figure 7).

$$\texttt{swInit} \quad \frac{h \vdash_{\mathrm{S}} \{cl\} \rightarrow \widetilde{cl}}{h \vdash_{\mathrm{P}} cl \rightarrow \widetilde{cl}}$$

The sweeper invokes the rewriter with the goal of simplifying the literals contained in a clause. The rewriter takes as input a literal of the clause and the typeset information that the remaining literals of the clause are false. If the answer of the rewriter is T, this means that the clause is true. The bridge rule swT describes this situation in our framework (arrow from Rth_{S} to Rth_{R} in Figure 7).

$$\texttt{swT}\quad \frac{h \vdash_{\mathrm{R}} initTI(cl);\ l \rightarrow_{\mathtt{B}} \mathtt{T}}{h \vdash_{\mathrm{S}} \widetilde{cl} \cup \{cl \cup \{l\}\} \rightarrow \widetilde{cl}}$$

where $initTI(cl)$ is the typeset information structure corresponding to assuming the negation of each literal in cl. From a logical point of view the premiss of this rule corresponds to the formula

$$(\bigwedge_{l' \in cl} \neg l') \;\Rightarrow\; (l \Leftrightarrow \mathtt{T}).$$

13. The Rth_{NQTHM} Reasoning Theory

The modules of the NQTHM simplification process are: the master simplifier; the sweeper; the rewriter; and the linear arithmetic specialist. The linear arithmetic specialist reasons about linear inequalities over the natural numbers. The other modules correspond to modules of pNQTHM modified in a suitable way in order to integrate the linear arithmetic specialist into NQTHM.

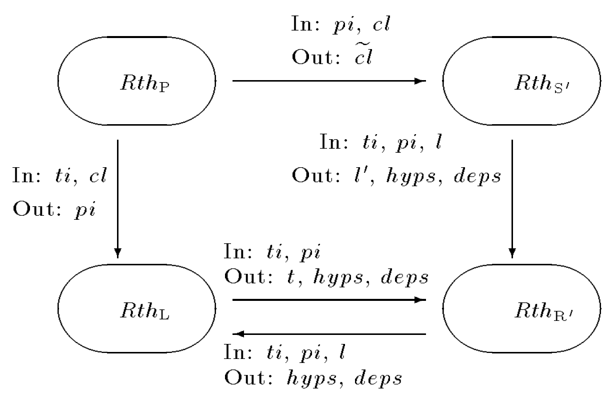

Figure 8. The Reasoning Theory of the NQTHM simplification process

As for pNQTHM, we present the NQTHM reasoning theory as a composite reasoning theory

$$Rth_{\mathrm{NQTHM}} = \langle Ssys_{\mathrm{NQTHM}}, Id_{\mathrm{NQTHM}}, \widetilde{r}_{\mathrm{NQTHM}} \rangle$$

constructed by putting together the reasoning theory Rth_{L} corresponding to the linear arithmetic module and a modified version of Rth_{pNQTHM} as follows

$$Rth_{\mathrm{NQTHM}} = glueRth([Rth_{\mathrm{U}}, Rth_{\mathrm{W}}, Rth_{\mathrm{P}}, Rth_{\mathrm{S'}}, Rth_{\mathrm{R'}}, Rth_{\mathrm{L}}], Id_{B'}, \widetilde{r}_{B'})$$

where $Rth_{\mathrm{S'}}$ and $Rth_{\mathrm{R'}}$ are suitable modifications of Rth_{S} and Rth_{R}. In Figure 8 we show the part of the structure of Rth_{NQTHM} relevant to the simplification process.

Below we first describe Rth_{L}. Then we discuss how $Rth_{\mathrm{S'}}$ and $Rth_{\mathrm{R'}}$ are obtained from Rth_{S} and Rth_{R}, by refining the S and R sequent sorts and suitably revising the rules involving these sorts to include the additional information passed to and from the linear arithmetic specialist. Finally, we describe some of the bridge rules which must be added to link Rth_{L} to $Rth_{\mathrm{R'}}$ and $Rth_{\mathrm{S'}}$.

13.1. The Rth_{L} Reasoning Theory

The linear arithmetic specialist gets as input a data structure, called polynomial database, typeset information to be used by the typeset specialist and a set of literals to be translated into the linear arithmetic world, and returns a new polynomial database. A polynomial database contains a set of data structures, called polynomials, which are used for representing linear arithmetic inequalities. A polynomial contains a linear inequation of the form $z_0 + z_1 \cdot t_1 \ldots z_n \cdot t_n \leq 0$ (where the z_i are integers and the t_i are terms), a set of literals, *lh*, called linearization hypotheses, and a set of literals, *ld*, called linearization dependencies. The formula represented by a polynomial is $lh \Rightarrow p$. The linearization dependencies are the literals from which a polynomial is derived, and are used to project different views of the polynomial database. More explicitly, in NQTHM each literal of a clause is rewritten in the local context where the remaining literals are assumed false. In the presence of linear arithmetic, these literals assumed to be false are represented in two ways: as typeset information to be used by the typeset module and as a polynomial database used by the linear arithmetic module. Rather than build a polynomial database for each complement to a literal in a given clause, the polynomial database corresponding to assuming false each literal in the clause is built. The polynomial database corresponding to the complement of a given literal is obtained by hiding any polynomial having that literal in its set of dependencies.

The linear arithmetic module can add a set of literals, producing a new polynomial database, and it can be asked if the polynomial database so obtained is inconsistent, i.e. if some impossible linear arithmetic consequence, e.g. $2 \leq 0$, can be derived from the database. If so, an impossible polynomial is returned as witness. The linearization hypotheses of this polynomial must be incorporated in the interpretation of the inconsistency.

In our framework L-sequents have the form:

(L) $\quad h \vdash_{\mathrm{L}} ti;\ pi \rightarrow pi'$

where the local context, ti, is a typeset information structure and the focus information, $pi \rightarrow pi'$, consists of a pair of polynomial information structures that represents a pair containing (the logical part of) a polynomial database and a set of literals to be translated into the linear arithmetic world. The sequent asserts that pi' is equivalent to pi in the theory h under the assumptions in ti.

The internal rules for Rth_{L} correspond to linearization (transforming literals to a canonical polynomial form) and simple linear arithmetic reasoning such as "cross-multiply" polynomials to obtain new polynomials. We will not discuss these rules further. We include here only one rule, for the sake of completeness, as it is used in the examples. This is the L-rule **transla** that expresses transitivity of the $\rightarrow$ relation on polynomial information structures.

$$\texttt{transla}\quad \frac{h \vdash_{\mathrm{L}} ti;\ pi_0 \rightarrow pi_1 \qquad h \vdash_{\mathrm{L}} ti;\ pi_1 \rightarrow pi_2}{h \vdash_{\mathrm{L}} ti;\ pi_0 \rightarrow pi_2}$$

13.2. The modified Rth_{pNQTHM} Sequent System

In NQTHM the sweeper is given, in addition to a clause, the polynomial database built from the negations of the literals of this clause. Correspondingly, the S-sequents of the NQTHM reasoning theory are modified to have as local context the polynomial information structure for the clause being swept. (Notice that in the following, to keep the notation simple, we use S and R instead of S$'$ and R$'$.)

(S) $\quad h \vdash_{\mathrm{S}} pi;\ \widetilde{cl} \rightarrow \widetilde{cl}\,'$

A polynomial database is added to the context of the rewriter. The output of the rewriter contains, in addition to the simplified input term, two sets of literals: the linearization hypotheses and the dependencies tracked for use in polynomials. Correspondingly, the R-sequents of the NQTHM reasoning theory are modified to include a polynomial information structure in the local context, and to add two sets of literals to the focus information.

(R) $\quad h \vdash_{\mathrm{R}} pi, ti;\ t \rightarrow_m \langle t', \widetilde{l}_1, \widetilde{l}_2 \rangle$

An NQTHM R-sequent asserts that t is equivalent (relative to the mode) to t' in the theory h under the additional assumptions in pi, ti, $\widetilde{l}_1$ and $\widetilde{l}_2$.

13.3. The modified Rth_{pNQTHM} Rules

The R-rule **rewR** is modified to propagate the polynomial information component of the local context along with the typeset information, and to accumulate the linearization hypotheses and dependencies generated in relieving the hypotheses. Note that the rule matching constraint is unchanged.

$$\texttt{rewR}\quad \frac{[h \vdash_{\mathrm{R}} ti, pi;\ l_i \rightarrow_{\mathtt{B}} \langle \mathtt{T}, \widetilde{l}_{1,i}, \widetilde{l}_{2,i} \rangle \mid 1 \le i \le n]}{h \vdash_{\mathrm{R}} ti, pi;\ t \rightarrow_{\mathtt{I}} \langle t', \bigcup_{1 \le i \le n} \widetilde{l}_{1,i}, \bigcup_{1 \le i \le n} \widetilde{l}_{2,i} \rangle} \quad \text{if} \quad \mathit{rrMatch}(h, [l_i]_{1\ldots n}, t, t', m)$$

The bridge rule `swInit` is modified to provide for initialization of the polynomial information structure.

$$\texttt{swInit}\quad \frac{h \vdash_{\mathrm{L}} \mathit{initTI}(cl);\ \mathit{initPI}(cl) \rightarrow pi \qquad h \vdash_{\mathrm{S}} pi;\ \{cl\} \rightarrow \widetilde{cl}}{h \vdash_{\mathrm{P}} cl \rightarrow \widetilde{cl}}$$

where $\mathit{initPI}(cl)$ inserts the negations of the literals in cl into the empty polynomial information structure.

The bridge rule `swT` is modified to propagate the restricted polynomial information structure, and to take account of the linearization hypotheses accumulated during rewriting. At this point, the linearization dependencies can be discarded.

$$\texttt{swT}\quad \frac{h \vdash_{\mathrm{R}} \mathit{initTI}(cl), \mathit{hide}(pi, l);\ l \rightarrow_{\mathtt{B}} \langle \mathtt{T}, \widetilde{l}_1, \widetilde{l}_2 \rangle}{h \vdash_{\mathrm{S}} pi;\ \widetilde{cl} \cup \{cl \cup \{l\}\} \rightarrow \widetilde{cl} \cup \mathit{split}(cl \cup \{l\}, \widetilde{l}_1, l)}$$

where $\mathit{hide}(pi, l)$ effectively removes from pi any polynomial with l among its linearization dependencies. Thus if pi is the result of assuming the negation of each literal in $cl \cup \{l\}$, then $\mathit{hide}(pi, l)$ is effectively the result of assuming the negation of each literal in cl. $\mathit{split}(cl \cup \{l\}, \widetilde{l}_1, l)$ is the set of variants of $cl \cup \{l\}$ that must be established to account for the additional hypotheses $\widetilde{l}_1$.

We include one further rule, for the sake of completeness, as it is used in the examples. This is the R-rule **reflr** that expresses reflexivity of the $\rightarrow$ relation on terms, in either mode.

$$\texttt{reflr}\quad \frac{}{h \vdash_{\mathrm{R}} ti, pi;\ t \rightarrow_m \langle t, \emptyset, \emptyset \rangle}$$

13.4. New Bridge Rules

We present three additional bridge rules. They represent respectively the invocation of the linear arithmetic specialist by the top level part of the simplification process, the invocation of the linear arithmetic specialist by the rewriter, and the invocation of the rewriter by the linear arithmetic specialist.

The top level part of the NQTHM simplification process invokes the linear arithmetic specialist to derive an impossible polynomial (i.e. a polynomial containing an impossible inequation, e.g. $2 \leq 0$) from the negation of the clause *cl* given in input to the process. When this succeeds, this means that *cl* has been proved by linear arithmetic reasoning under additional hypotheses corresponding to the linearization hypotheses of the impossible polynomial found. The bridge rule `buildPI` represents in our setting this interaction between the two modules (arrow from Rth_{P} and Rth_{L} in Figure 8).

$$\texttt{buildPI} \quad \frac{h \vdash_{\mathrm{L}} \mathit{initTI}(cl);\ \mathit{initPI}(cl) \to pi}{h \vdash_{\mathrm{P}} cl \to \mathit{split}_c(cl, \widetilde{l}_1)} \quad \text{if} \quad \mathit{impos}(pi, \langle \widetilde{l}_1, \widetilde{l}_2 \rangle)$$

The constraint $\mathit{impos}(pi, \langle \widetilde{l}_1, \widetilde{l}_2 \rangle)$ means there is a polynomial in pi, deduced from the literals in $\widetilde{l}_2$, which is impossible given the additional assumptions $\widetilde{l}_1$. These can be considered as a case split and $\mathit{split}_c(cl, \widetilde{l}_1)$ yields the clauses that must be proved for the remaining cases.

The linear arithmetic module is invoked by the rewriter for proving by linear arithmetic reasoning that certain expressions are true or false. This interaction can be explained informally as follows. Suppose the rewriter is trying to prove that a literal l is true (the case for false is analogous). Then the linear arithmetic module is invoked to find an impossible polynomial from the negation of l (as the linear arithmetic module works by refutation) and the current polynomial database and typeset information. When this succeeds, it means that l is true under the additional assumptions corresponding to the the set of linearization hypotheses and dependencies of the impossible polynomial found. These assumptions are accumulated and returned at the end of the rewriting process. The bridge rule `rewT` represents in our framework this interaction (arrow from Rth_{R} to Rth_{L} in Figure 8).

$$\texttt{rewT} \quad \frac{h \vdash_{\mathrm{L}} ti;\ \mathit{assertneg}(pi, l) \to pi_1}{h \vdash_{\mathrm{R}} ti, pi;\ l \to_{\mathrm{B}} \langle \mathtt{T}, \widetilde{l}_1, \widetilde{l}_2 \rangle} \quad \text{if} \quad \mathit{impos}(pi_1, \langle \widetilde{l}_1, \widetilde{l}_2 \rangle)$$

where $\mathit{assertneg}(pi, l)$ adds the negation of l to the literals assumed in pi.

The linear arithmetic module invokes the rewriter to find rule instances of a suitable form (called linear rules) in the current theory that might be used to generate additional polynomials. If a suitable rule instance is found, and its hypotheses can be established in the current context, then its conclusion is rewritten and linearized, the linear hypotheses and dependencies accumulated while establishing the hypotheses and rewriting the conclusion are added, and the resulting polynomial is added to the polynomial database. The bridge rule for this interaction, **laR**, is defined as follows (arrow from Rth_{L} to Rth_{R} in Figure 8).

$$
\texttt{laR}\ \frac{\begin{array}{l}[h \vdash_{\mathrm{R}} ti, pi;\ l_i \rightarrow_{\mathtt{B}} \langle \mathtt{T}, \widetilde{l}_{1,i}, \widetilde{l}_{2,i} \rangle \mid 1 \leq i \leq n] \\ h \vdash_{\mathrm{R}} ti, pi;\ x \rightarrow_{\mathtt{I}} \langle x_1, \widetilde{l}_{1,n+1}, \widetilde{l}_{2,n+1} \rangle \\ h \vdash_{\mathrm{R}} ti, pi;\ y \rightarrow_{\mathtt{I}} \langle y_1, \widetilde{l}_{1,n+2}, \widetilde{l}_{2,n+2} \rangle\end{array}}{h \vdash_{\mathrm{L}} ti;\ pi \rightarrow addPoly(pi, \langle L(x_1, y_1), \widetilde{l}_1, \widetilde{l}_2 \rangle)} \quad \text{if} \quad lrMatch(h, [l_i]_{1 \ldots n}, x, y, t)
$$

where $\widetilde{l}_1 = \bigcup_{1 \leq i \leq n+2} \widetilde{l}_{1,i}$, $l_2 = \bigcup_{1 \leq i \leq n+2} \widetilde{l}_{2,i}$, $addPoly(pi, p)$ adds the polynomial p to pi and $lrMatch(h, [l_i]_{1 \ldots n}, x, y, t)$ is the linear rule matching constraint. It holds if there is a linear rule in h with hypotheses $\wedge_{1 \leq i \leq n} l'_i$, conclusion $L(x', y')$; s is a substitution (actually in NQTHM this substitution has to satisfy some further heuristic conditions); l_i is the result of applying s to l'_i, for $1 \leq i \leq n$; and x, y are the results of applying s respectively to x', y'.

14. Examples of Rth_{NQTHM} Reasoning Structures

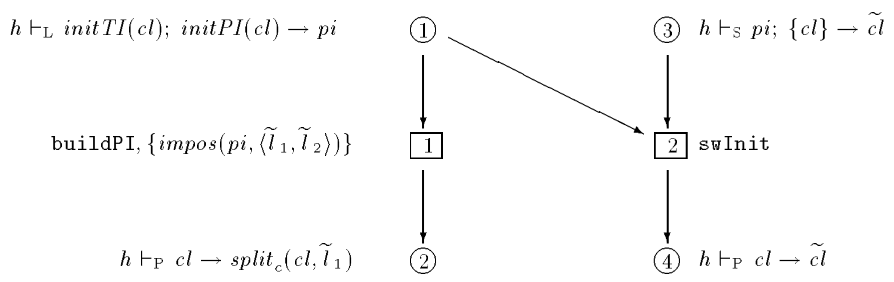

Figure 9. Rth_{NQTHM} Reasoning Structure Fragment

We mentioned above that, to simplify a clause, the top level part of the NQTHM simplification process invokes the linear arithmetic module to attempt to derive an impossible polynomial from its negation. When this fails, the sweeper is invoked to rewrite each of the literals. The sweeper is given both the clause and the polynomial database returned by the linear arithmetic module. In our setting this corresponds to a reuse of the derivation which constructs the polynomial database as one of premisses in the rule `swInit`. Figure 9 shows a schematic reasoning structure where this reuse is made explicit by sharing of graph structure. This reasoning structure provides also an example of provisional reasoning (see the constraint contained in the justification of link node 1).

As a more concrete example we show how to represent the NQTHM proof described on p. 29-30 in [8]. (In the following reasoning structures are presented and discussed "backward", in order to follow the reasoning steps performed by NQTHM.) The conjecture to prove comes from a simple step in the proof of the correctness of the Boyer-Moore fast string searching algorithm [6].

$$(lp + lt \leq maxint \wedge i \leq lt) \;\;\Rightarrow\;\; (i + delta1(pat, lp, c) \leq maxint)$$

where lp, lt, $maxint$ and i are natural numbers and $delta1$ satisfies the axiom

$$delta1(pat, lp, c) \leq lp.$$

A sequence of commands (events) that will lead NQTHM to prove the conjecture is given below. The first command declares the function `DELTA1`. Then the axiom `DELTA` is added to the rewrite rules (it is stored as a linear rule by NQTHM) and the last command is used to prove the theorem.

```
(DCL DELTA1 (X Y Z))
(ADD-AXIOM DELTA (REWRITE) (NOT (LESSP LP (DELTA1 PAT LP C))))
(PROVE '(IMPLIES (AND (LEQ (PLUS LP LT) MAXINT)
                      (LEQ I LT))
                 (LEQ (PLUS I (DELTA1 PAT LP C)) MAXINT)))
```

The output we get from NQTHM is:

> *This formula simplifies, using linear arithmetic and applying the lemma DELTA, to: T.*
>
> *Q.E.D.*

Figure 10 summarizes the notation used to abbreviate complex expressions in the reasoning structures representing this proof.

We use the vertical flexibility of reasoning structures to present different levels of detail of the proof. The top level reasoning structure corresponds roughly to the information that the user gets from the prover. This is shown in Figure 11. We use two presentations: linear and graphical. The first uses notation similar to Kleene's presentation of proofs [39]. The columns in the table of the figure correspond respectively to a sequent node, its label, the link node having the sequent node as conclusion, the subgoals of the link node and its justification. The nested reasoning structures rs_a and rs_c use pure linear arithmetic and we omit details. The nested reasoning structure rs_b uses the lemma `DELTA`. This reasoning structure is shown in Figure 12. This illustrates the use of lemmas during linear reasoning.

In Figure 11, the repeated use of `transla` clutters the presentation of the proof. In Figure 13 we show a reasoning structure which contains the sequent and link

Symbol	Abbreviation for
t	$(lp + lt \leq maxint \wedge i \leq lt) \Rightarrow (i + delta1(pat, lp, c) \leq maxint)$
l_1	$maxint < lp + lt$
l_2	$lt < i$
l_3	$\neg(maxint < i + delta1(pat, lp, c))$
l_4	$\neg(lp < delta1(pat, lp, c))$
cl	$\{l_1, l_2, l_3\} = preprocess(h_1, t)$
t_1	lp
t_2	$delta1(pat, lp, c)$
ti	$initTI(cl)$
p_1	$\langle lp + lt - maxint \leq 0, \emptyset, \{l_1\}\rangle$
p_2	$\langle i - lt \leq 0, \emptyset, \{l_2\}\rangle$
p_3	$\langle 1 - delta1(pat, lp, c) - i + maxint \leq 0, \emptyset, \{l_3\}\rangle$
p_4	$\langle delta1(pat, lp, c) - lp \leq 0, \emptyset, \{l_4\}\rangle$
p_5	$\langle 1 - i - lp + maxint \leq 0, \emptyset, \{l_3, l_4\}\rangle$
p_6	$\langle 1 - lp - lt + maxint \leq 0, \emptyset, \{l_2, l_3, l_4\}\rangle$
p_7	$\langle 1 \leq 0, \emptyset, \{l_1, l_2, l_3, l_4\}\rangle$
pi_0	$initPI(cl)$
pi_i	contains the polynomials p_j for $1 \leq j \leq i$
h_1	the theory resulting from the first two commands

Figure 10. Abbreviations

nodes 5–9 of the reasoning structure in Figure 11, where we use the rule **transla*** to hide sequent and link nodes 7. **transla*** can be considered as simply an abbreviation for a nested reasoning structure that contains the hidden detail. We could also have chosen this as an official rule. Or it could be considered as a derived rule – replacing syntactic sugar by a first class rule. Work is ongoing to develop a theory of admissible derived rules. (See section §16 on the future work.)

sn	sL	ln	sg	lL
1	$h_1 \vdash_{\mathrm{U}} t$	1	2	**clausify**
2	$h_1 \vdash_{\mathrm{W}} \{cl\}$	2	$3, 4$	**select**
3	$h_1 \vdash_{\mathrm{P}} cl \rightarrow \emptyset$	3	5	**buildPI**
4	$h_1 \vdash_{\mathrm{W}} \emptyset$	4	$\emptyset$	**qed**
5	$h_1 \vdash_{\mathrm{L}} ti;\ pi_0 \rightarrow pi_7$	5	$6, 7$	**transla**
6	$h_1 \vdash_{\mathrm{L}} ti;\ pi_0 \rightarrow pi_3$	6	$\emptyset$	$\langle \emptyset, \mathbf{idi}, [[\ ], 6'], rs_a \rangle$
7	$h_1 \vdash_{\mathrm{L}} ti;\ pi_3 \rightarrow pi_7$	7	$8, 9$	**transla**
8	$h_1 \vdash_{\mathrm{L}} ti;\ pi_3 \rightarrow pi_4$	8	$\emptyset$	$\langle \emptyset, \mathbf{idi}, [[\ ], 8'], rs_b \rangle$ (using axiom **DELTA**)
9	$h_1 \vdash_{\mathrm{L}} ti;\ pi_4 \rightarrow pi_7$	9	$\emptyset$	$\langle \emptyset, \mathbf{idi}, [[\ ], 9'], rs_c \rangle$

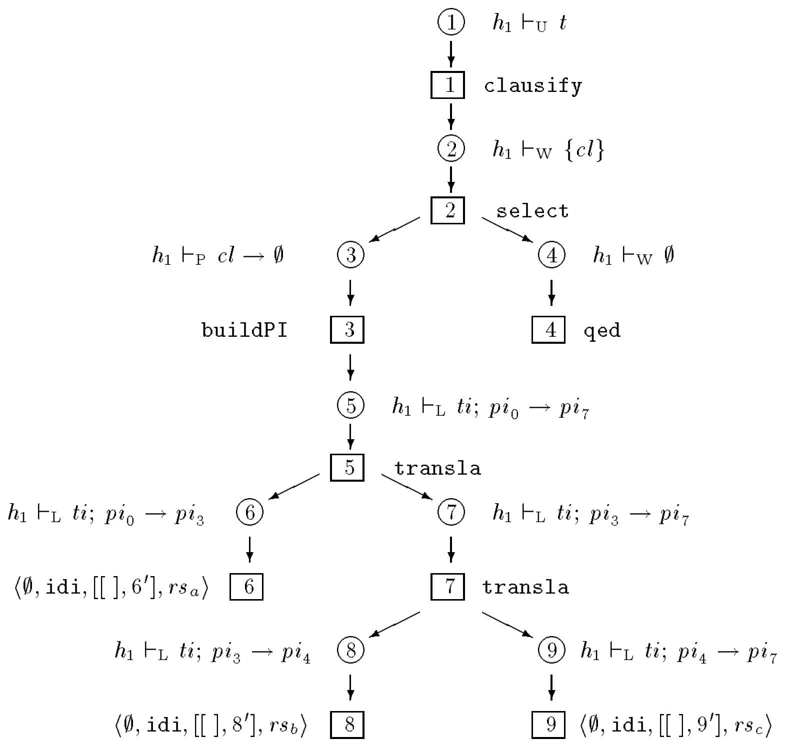

Figure 11. The user-level reasoning structure

IV Concluding Matter

sn	*sL*	*ln*	*sg*	*lL*
$8'$	$h_1 \vdash_{\mathrm{L}} ti;\ pi_3 \rightarrow pi_4$	10	10, 11	**laR** (using axiom **DELTA**)
10	$h_1 \vdash_{\mathrm{R}} ti, pi_3;\ t_1 \rightarrow_{\mathrm{I}} \langle t_1, \emptyset, \emptyset \rangle$	11	$\emptyset$	**reflr**
11	$h_1 \vdash_{\mathrm{R}} ti, pi_3;\ t_2 \rightarrow_{\mathrm{I}} \langle t_2, \emptyset, \emptyset \rangle$	12	$\emptyset$	**reflr**

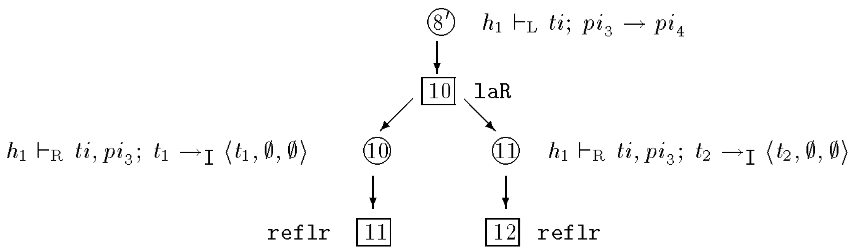

Figure 12. Opening up the reasoning structure rs_b

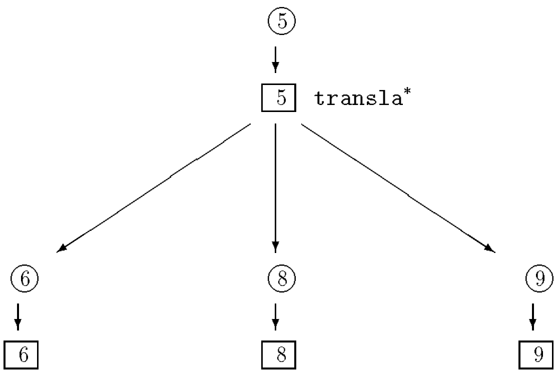

Figure 13. Generalized Transitivity

15. Related Work

Reasoning theories, and in particular composite reasoning theories, are similar in spirit to multi-language systems [21, 19]. They can be considered an intensional view of the deductive aspect of logic (cf. entailment theories and proof calculi [43]). In multi-language systems, sequents may have hypotheses and conclusion from different languages (sequent systems), and bridge rules may discharge hypotheses across languages.

The notion of inference rule generalizes the notion of deduction rule described by Prawitz [54]. There, a deduction rule is considered as a relation on pairs (called *sequents*) consisting of a list of assumption formulas and a conclusion formula, and proofs are tree-like structures. The notion of n-ary rule generator generalizes the notion of rule introduced by Scott [57]. Here (propositional) systems are determined by a set of sentences and a relation $\vdash$ between finite sets of sentences. A rule is expressed using meta variables for sentences combined using finite set formation, $\vdash$, conjunction and implication. A rule is correct for a system if it is valid in the system. Notions of a rule following from a set of rules and a rule being derivable from a set of rules are also defined.

Reasoning structures generalize the deduction graphs used in IMPS [46, 16] in several ways: a richer domain of sequents; using constraints for provisional reasoning; and nesting.

The work presented in this paper is an attempt at an axiomatic presentation of a wide class of deductive systems in the spirit of the work on general logics [43]. Other meta-logical frameworks have focused on notational systems for presenting logics (more precisely, deductive or entailment systems using the classification of [43], as they treat only syntax and deduction, not models). Some examples are: LF [27, 28] – a meta-logical system for describing and prototyping logics; lambda-prolog [45] – a meta programming language for manipulating syntactic entities such as programs, formulas, and proof structures; and Feferman's formalism based on inductive definitions for specifying syntax, proofs, and in some cases models of logical systems [17, 18]. The closest in spirit to our work is the work on LF, as both approaches share the main goal of characterizing formal systems and provability at an abstract level. However there are important differences, all consequences of the different targets: LF focuses on formalizing (the deductive aspects of) logics while the work on reasoning theories focuses on formalizing and specifying provers, and on analyzing the data structures that support their design and integration. The following points provide a more detailed comparison.

1. LF is a formalism for presenting logics [formal systems]. The objective is to unify and abstract commonality from the plethora of logics, and to provide a generator of proof editors/checkers from presentations. Reasoning theories are a mathematical framework for designing and specifying the (operational) semantics of reasoning modules. Reasoning theories are the declarative component of reasoning systems. The objective of this work is to provide a framework for interoperation and integration of logical services.

2. The basic notions of LF are: judgments (formulas as types); proofs as elements of judgements (typed terms); rules as elements of higher-order judgements (higher typed terms). The basic notions of reasoning theories are: sequents (considered as data structures) and constraints; rules (relations on sequents and constraints); and reasoning structures (data structures representing provisional derivation fragments). Presentations of reasoning theories are expected to be first-order theories.

3. An LF presentation is an LF signature assigning types and kinds to a finite set of constant symbols. There is only limited ability to define operations and relations on syntactic entities. In LF binding and schematic abstraction and instantiation are expressed using lambda-abstraction and binding. Variables don't exist at the object level. This provides an elegant treatment of binding, but it is not convenient as a basis for manipulating judgements as data structures. LF does not allow for provisional reasoning.

4. In LF application and abstraction give an elegant account of assumption discharge for certain 'nice' natural deduction systems, however many logics (in particular those used inside mechanized reasoning systems) don't fit this niche. In reasoning theories it is expected that assumptions are carried as part of the sequent structure and there is no bias towards (or against) natural deduction.

16. Future Work

We are at the beginning of a long term project which still needs to be developed in many directions.

One line of research will aim at completing our characterization of reasoning systems. To begin with, this will require a further development of the current notion of reasoning theory. In this paper we have allowed justifications in the form of nested reasoning structures. A natural and important extension is to associate nested reasoning structures to the application of derived inference rules. This will allow us to have a notion of deduction where it is possible to call complex inference procedures as single inference rules (this being already possible in the current notion of reasoning structure), and, under request, to provide the reasoning structures built by such procedures as a justification for their applications. For instance, in §6, we have used constraints whose testing requires a lot of reasoning, e.g. $TautConseq(A_1 \wedge \ldots \wedge A_n, A)$. In the actual FOL implementation of TAUT, the constraint $TautConseq(A_1 \wedge \ldots \wedge A_n, A)$ is decided using a module which takes a formula, and, using truth tables, tests whether it is a tautology. This module implements an efficient generalization to non-clausal formulas of the Davis Putnam procedure. With this extension it will be possible to open up the application of a derived inference rule, e.g. TAUT, and substitute it with the reasoning structures built by the embedded decider, e.g. a David Putnam procedure. Conversely, it will also allow us to hide a complex reasoning structure in the application of a derived inference rule. This feature is very important as, among other things, it will allow us to check (e.g. via a simple proof checker whose correctness we trust) the correctness of complex inference rule applications whose correctness we do not trust. It will also allow us to have a notion of correctness of a proof "modulo the correctness of a subset of the inference rules applied in its development". For more on the issue of the correctness of mechanized reasoning systems see [4]. Technically, this can be done by allowing the use of nested reasoning theories, and of linking nodes with both a rule identifier and a pointer to a nested reasoning structure. Some of the complications are due to the fact that different inferencers use different languages, and to the fact that constraints must propagate correctly.

The second component of a reasoning system is control. Here the goal is to be able to specify complex provers as reasoning systems with a functional (tactical) language calling appropriate inference rules. The problem is that control in general makes use of a lot of state (e.g. the number of times a formula has been used), whose manipulation is usually hardwired in some ad hoc way in the system code. Our idea is to decorate sequents with annotations which make explicit these manipulations. The open question is whether it will be possible to identify (at least a subset of) the state which is used by a large number of systems and a general system independent syntax for describing it.

The third component of a reasoning system is interaction. Here we need to find a general language which can be used to specify the possible requests to a reasoning system, and to provide it with an extensional semantics. Some examples of possible requests are: prove this goal, give me a proof of this goal, add this lemma to your database of rewriting rules, add this axiom to your database of axioms, add this information to your local context, and so on. Then we must be able to translate these operations into the reasoning systems' internal operations, e.g. into calls to appropriate control procedures, and to prove that this translation preserves the semantics. Some preliminary ideas and results are reported in [64].

The applicability of our proposed methodology must be tested against important, non-trivial examples. Currently, work is underway to develop specifications of the logical services provided by NQTHM and its component modules. In [22] we have already provided a specification of the reasoning theory associated to the simplification process. We plan to extend this analysis to consider also control and interaction. Additional case studies are being considered including: the Nelson-Oppen cooperating decision procedures and simplifier (this having the highest priority); Ontic; PVS; resource limited logics; semantic tableaux base provers, resolution or mating based provers, systems which integrate provers and symbolic mathematical systems.

A formal notation is needed for presenting the syntax and semantics of sequent systems and for defining relations between sequent systems. To account uniformly for notions of binding we will build on the work on binding structures [65]. As a starting point we expect to use a notation for first-order theories such as OBJ [24] or Maude [41]. The module composition capability of these systems will also serve as a starting point for developing a richer calculus of reasoning theories.

So far we have considered only the operational and proof theoretic aspects of reasoning systems. We need to add a semantic component to the framework to provide a notion of model for reasoning theories and to reason about how semantics compose when we connect together heterogeneous reasoning systems. The work on general logics [43] provides the starting point for this work. A very powerful notion of module composition is that based on theory mappings [43]. This is a central idea in the work on the Clear specification language[23] and is the basis for the module level of the OBJ language[24]. Theory mappings (also called Views) insure not only syntactic, but also semantic composability of modules.

Some of the problems that the work on reasoning systems addresses are general problems of modularization, composability, and interoperability that are not specific to the domain of automated reasoning. As the ideas and techniques become better developed, we expect that general principles will emerge that can be used in other application domains.

Acknowledgements. The authors would like to thank the Open Architectures for Reasoning Systems working group at Stanford, Jussi Ketonen for many stimulating discussions during the course of the work, Maura Cerioli and Toby Walsh for pointing out errors and omissions in an earlier draft.

This research was partially supported by ARPA grants NAG2-703, NAVY N00014-94-1-0775, NSF grants CCR-8917606, CCR-8915663, ONR grant N00014-94-1-0857, and CNR grant CN 92.03006.CT26.

17. Proofs

17.1. Derivations

Lemma (reach): Let $rs = \langle Sn, Ln, g, sg, sL, lL \rangle$ be a derivation with conclusion node sn_0, then every sequent node is reachable from sn_0 by a chain of subgoal links, i.e. for each $sn \in Sn$ there is a sequence $[ln_j, sn_{j+1} \mid 0 \leq j < n]$ such that $g(ln_j) = sn_j$, $sn_{j+1} \in sg(ln_j)$ for $0 \leq j < n$, and $sn_n = sn$.

Proof (reach): By reductio ad absurdum. Let $sn \in Sn$ be a sequent node that is not reachable from sn_0. The case $sn = sn_0$ is trivial. Suppose that $sn \neq sn_0$. By (3) in the definition of derivation, sn must be the premiss of a link node $ln \in Ln$. Let sn' be the conclusion node of ln. From the hypothesis that sn is not reachable from sn_0, it follows that also sn' is not reachable from sn_0. Iterating this reasoning, we can build an infinite sequence of nodes without repetitions (there are no cycles in rs). But then we get an inconsistency since Sn is finite.

$\square_{\mathbf{reach}}$

Lemma (derivation instantiation): If rs is a derivation of s from $\widetilde{s}$ then $rs[\iota]$ is a derivation of $s[\iota]$ from $\widetilde{s}[\iota]$ for any instantiation ι.

Proof (derivation instantiation):

By induction on the level n of $\mathbf{Rs}_n[Rth, SN, LN]$.

Base case: $rs \in \mathbf{Rs}_0[Rth, SN, LN]$. Let ι be an instantiation. From the fact that rs is a derivation, it is easy to prove that $rs[\iota]$ satisfies conditions (1)-(5) in the definition of derivation. Condition (1) holds trivially because rs has no unsolved constraints. Conditions (2)-(4) are satisfied because rs and $rs[\iota]$ have the same graph structure. Condition (5) holds because rs has no nesting link. The conclusion of $rs[\iota]$ is $s[\iota]$ and its open assumptions are contained in $\widetilde{s}[\iota]$, because, as already observed, the graph structure of rs is preserved under instantiation.

Induction step: $rs \in \mathbf{Rs}_{n+1}[Rth, SN, LN]$. Let ι be an instantiation. As done in the previous case, we can prove that $rs[\iota]$ satisfies conditions (1)-(4) in the definition of derivation. It remains to prove that $rs[\iota]$ satisfies condition (5). Let ln be a nesting link of rs whose associated justification is $\langle \emptyset, \iota', [\overline{sn}, sn], rs' \rangle$. It follows that the justification of ln w.r.t. $rs[\iota]$ is $\langle \emptyset, \iota \circ \iota', [\overline{sn}, sn], rs' \rangle$. As $rs'[\iota'] \in \mathbf{Rs}_n[Rth, SN, LN]$, by induction hypothesis we have that $rs'[\iota'][\iota]$ is a derivation of the sequent labelling sn (in $rs'[\iota'][\iota]$) from the premiss sequents labelling $\overline{sn}$. Finally, as done above, we can prove that the conclusion of $rs[\iota]$ is $s[\iota]$ and that its open assumptions are contained in $\widetilde{s}[\iota]$.

$\square_{\mathbf{derivationinstantiation}}$

Lemma (elimination of nesting): If rs is a $\mathbf{Rs}[Rth, SN, LN]$ derivation of s from $\widetilde{s}$ then we can find a level 0 derivation $rs_0 \in \mathbf{Rs}_0[Rth, SN, LN]$ of s from $\widetilde{s}$.

Proof (elimination of nesting): We prove this lemma by well-founded induction w.r.t. the relation $\prec$ defined on $\mathbf{Rs}[Rth, SN, LN]$ as follows. Let

$$rs_1 = \langle Sn_1, Ln_1, g_1, sg_1, sL_1, lL_1 \rangle$$

and

$$rs_2 = \langle Sn_2, Ln_2, g_2, sg_2, sL_2, lL_2 \rangle$$

be two reasoning structures in $\mathbf{Rs}[Rth, SN, LN]$, then $rs_1 \prec rs_2$ iff one of the following conditions is satisfied:

(1) there exists a number n such that $rs_1 \in \mathbf{Rs}_n[Rth, SN, LN]$ and $rs_2 \notin \mathbf{Rs}_n[Rth, SN, LN]$;

(2) the previous condition is not satisfied and the cardinality of Ln_1 is less than the cardinality of Ln_2.

Let rs be a derivation of s from $\tilde{s}$ whose conclusion node is sn. Let ln be the unique link node (from condition (2) in the definition of derivation) such that $g(ln) = sn$ (the case when there is no such a link node is trivial). There are two cases.

(i) ln is a rule application link. Let $sg(ln) = [sn_1, \ldots, sn_k]$ be the sequence of premisses of ln. For each sn_i there is in rs a subderivation rs_i of the sequent labelling sn_i, say s_i, from $\tilde{s}$. Since $rs_i \prec rs$, it follows by induction hypothesis that there exists a level 0 derivation $rs_i' \in \mathbf{Rs}_0[Rth, SN, LN]$ of s_i from $\tilde{s}$. Therefore we can construct a level 0 derivation rs_0 of s from $\tilde{s}$ using as ingredients the sequence of reasoning structures rs_1', ..., rs_k', the link node ln and the sequent node sn.

(ii) ln is a nesting link. Let $lL(ln) = \langle \emptyset, \iota, [\overline{sn}, sn], rs' \rangle$; then, as done before, we obtain by induction hypothesis a sequence of level 0 derivations rs_1', ..., rs_k' such that rs_i' is a derivation of the sequent labelling the i-th subgoal of ln from $\tilde{s}$. Moreover since $rs'[\iota] \prec rs$, by induction hypothesis there exists a level 0 derivation rs'' of s from the set of sequents labelling the subgoals of ln. In this case the ingredients for building a level 0 derivation rs_0 of s from $\tilde{s}$ are the sequence of reasoning structures rs_1', ..., rs_k' and the reasoning structure rs''.

$\Box$**eliminationofnesting**

Lemma (derivation trees): Let rs be a level 0 derivation of s from $\tilde{s}$ then there exists a level 0 derivation rs' of s from $\tilde{s}$ such that $Graph(rs')$ is a tree.

Proof (derivation trees): By induction on the number n of rule application link nodes of rs.

Base case: $n = 0$. Trivial.

Induction step. Let sn be the conclusion node of rs, ln be the unique link node such that $g(ln) = sn$ (from condition (2) in the definition of derivation) and $sg(ln) = [sn_1, \ldots, sn_k]$ be the subgoals of ln. By induction hypothesis for each i, $1 \leq i \leq k$, there exists a level 0 derivation rs_i of the sequent labelling sn_i from $\tilde{s}$ such that $Graph(rs_i)$ is a tree. It follows that we can construct a level 0 derivation

rs' of s from $\widetilde{s}$ such that $Graph(rs')$ is a tree, using the reasoning structures rs_1, ..., rs_k, the link node ln and the sequent node sn.

$\Box_{\mathbf{derivationtrees}}$

17.2. Operations

Theorem (soundness):

(1) `mtrs` is a basic reasoning structure.

(2) The operations `addS`, `linkR`, `solveC`, and `linkN` all map reasoning structures to reasoning structures (when applied to appropriate arguments).

(3) The operations `addS`, `linkR`, and `solveC` all map basic reasoning structures to basic reasoning structures (when applied to appropriate arguments).

Proof (soundness): (1) is trivial. The proof of (2) is straightforward from the definitions of `addS`, `linkR`, `solveC`, and `linkN`. (3) follows from the fact that `addS`, `linkR`, and `solveC` do not add nesting links.

$\Box_{\mathbf{soundness}}$

Theorem (completeness): If $rs \in \mathbf{Rs}[Rth, SN, LN]$, then rs can be constructed from the empty reasoning structure using only the operations `addS`, `linkR`, `solveC`, and `linkN`. The basic reasoning structures are generated by excluding `linkN`.

Proof (completeness):

By induction on the level n of $\mathbf{Rs}_n[Rth, SN, LN]$.

Base case: $rs = \langle Sn, Ln, g, sg, sL, lL\rangle \in \mathbf{Rs}_0[Rth, SN, LN]$. We sketch an algorithm for the construction of rs.

(add sequent nodes) Use `addS` to obtain the reasoning structure [5] (from the empty reasoning structure)

$$rs_{\mathtt{addS}} = \langle Sn, \emptyset, \vec{\emptyset}, \vec{\emptyset}, sL, \vec{\emptyset}\rangle.$$

(add rule application links) Let $Ln = \{ln_1, \ldots, l_n\}$, where $n \geq 0$, and $rs_0, \ldots, rs_n$ be a sequence of reasoning structures defined as follows.

(i) $rs_0 = rs_{\mathtt{addS}}$.

(ii) For $1 \leq i \leq n$, let $lL(ln_i) = \langle id_i, \widetilde{c}_i\rangle$, $\overline{s}_i = sL(sg(ln_i))$, $s_i = sL(g(ln_i))$, and $r_i = \langle \overline{s}_i, s_i, \widetilde{c}'_i\rangle \in \widetilde{r}(id_i)$ such that $\widetilde{c}_i \models \widetilde{c}'_i$. Then

(add link node) $rs'_i = \mathtt{linkR}(rs_{i-1}, sg(ln_i), g(ln_i), r_i)$ introducing ln_i; and

[5] Note that we are imposing a choice on the nodes generated by `addS` (and below by `linkR` and `linkN`). This is not necessary, because we have already observed that equality on reasoning structures is defined modulo node renaming. Anyway we do so in order to simplify the proof.

(solve constraints) $rs_i = \texttt{solveC}(rs'_i, ln_i, \widetilde{c}_i)$.

It is easy to verify that $rs_n = rs$.

Notice that in this algorithm we have not mentioned `linkN`. Hence it follows that the basic reasoning structures are generated by excluding `linkN`.

Induction step: $rs = \langle Sn, Ln, g, sg, sL, lL\rangle \in \mathbf{Rs}_{n+1}[Rth, SN, LN]$. Let $Ln = Ln_1 \cup Ln_2$, where Ln_1 (Ln_2) is the set of rule application (nesting) link nodes.

The construction of rs can be divided in two steps.

(top level) Construct with the process sketched for the base case the reasoning structure

$$rs' = \langle Sn, Ln_1, g \downarrow Ln_1, sg \downarrow Ln_1, sL, lL \downarrow Ln_1\rangle.$$

(add nesting links) Let $Ln_2 = \{ln_1, \ldots, ln_n\}$, where $n \geq 0$, and $rs_0, \ldots, rs_n$ be a sequence of reasoning structures defined as follows.

(i) $rs_0 = rs'$.

(ii) For $1 \leq i \leq n$, let $lL(ln_i) = \langle \widetilde{c}_i, \iota_i, [\overline{sn}_i, sn_i], rs'_i\rangle$, $\overline{s}_i = sL(sg(ln_i))$, $s = sL(g(ln_i))$. Then

(add link node) $rs'_i = \texttt{linkN}(rs_{i-1}, sg(ln_i), g(ln_i), \iota_i, \overline{s}_i, s_i)$ introducing ln_i,$\overline{sn}_i$ and sn_i;

(solve constraints) $rs''_i = \texttt{solveC}(rs'_i, ln_i, \widetilde{c}_i)$;

(add nested reasoning structure) rs_i is the reasoning structure obtained from rs''_i adding to the justification of ln the nested reasoning structure rs'_i. The construction of rs'_i is obtained with the basic operations using the non-empty path $\overline{ln}_i = [ln_i]$ (this step is guaranteed by induction hypothesis as $rs'_i \in \mathbf{Rs}_n[Rth, SN, LN]$).

It is easy to verify that $rs_n = rs$.

$\Box_{\textbf{completeness}}$

Theorem (independence): (**completeness**) fails if any of the operations in the list are omitted.

Proof :

Without `linkN`, only (and all) level 0 reasoning structures can be constructed.

Without `addS`, only the empty structure can be constructed since the other operations preserve the number of sequent nodes.

Without `linkR`, only structures with no rule application links can be constructed (if the set of rules is empty then `linkR` is not needed).

Without `solveC` reasoning structures with constraints in their nesting links cannot be constructed (if the set of constraints is empty then `solveC` is not needed).

$\Box_{\textbf{independence}}$

18. References

[1] M. Archer, G. Fink, and L. Yang. Linking other theorem provers to HOL using PM: Proof manager. In Claesen and Gordon [13], pages 539–549.

[2] A. Avron. Simple consequence relations. LFCS Report Series, Laboratory for the Foundations of Computer Science, Computer Science Department, University of Edinburgh, 1987.

[3] J. Barwise and J. Etchemendy. Valid inference and visual representation. In W. Zimmerman and S. Cunningham, editors, *Visualization in Mathematics*. Mathematical Association of America, 1990.

[4] D. Basin, F. Giunchiglia, and M. Kaufmann, editors. *Proceedings of the Workshop on Correctness and Metatheoretic Extensibility of Automated Reasoning Systems*, Nancy, France, 1994. Held in conjunction with *CADE-12*. Also IRST-Technical Report 9405-10, IRST, Trento, Italy.

[5] G. Bellin. *Mechanizing Proof Theory: Resource-Aware Logics and Proof-Transformations to Extract Implicit Information*. PhD thesis, Stanford University, 1990. Available as University of Edinburgh Department of Computer Science Report CST-80-91.

[6] R. S. Boyer and J. S. Moore. A fast searching algorithm. *Comm. ACM*, 20(10):762–772, 1977.

[7] R. S. Boyer and J S. Moore. A verification condition generator for FORTRAN. In R. S. Boyer and J S. Moore, editors, *The Correctness Problem in Computer Science*. Academic Press, London, 1981.

[8] R. S. Boyer and Moore. J. S. Integrating decision procedures into heuristic theorem provers: A case study with linear arithmetic. In *Machine Intelligence 11*. Oxford University Press, 1988.

[9] Robert S. Boyer and J. Strother Moore. *A Computational Logic*. Academic Press, 1979.

[10] Robert S. Boyer and J. Strother Moore. *A Computational Logic Handbook*. Academic Press, 1988.

[11] Alexandre Bronstein. *MLP: String-Functional Semantics and Boyer-Moore Mechanization for the Formal Verification of Synchronous Circuits*. PhD thesis, Stanford University, 1989.

[12] A. Bundy. The use of proof plans for normalization. In R.S. Boyer, editor, *Essays in Honor of Woody Bledsoe*, pages 149–166. Kluwer, 1991. Also available from Edinburgh as DAI Research Paper No. 513.

[13] L. J. M. Claesen and M. J. C. Gordon, editors. *Higher Order Logic Theorem Proving and its Applications*. Elsevier Science Publishers B. V. (North-Holland), 1993.

[14] R. L. Constable and et. al. *Implementing mathematics with the Nuprl development system.* Prentice-Hall, 1986.

[15] J.V. Cook, I.V. Filippenko, B.H. Levy, L.G. Marcus, and T.K. Menas. Formal Computer Verification in the State Delta Verification System (SDVS). In *Proceedings of the AIAA Computing in Aerospace Conference.* American Institute of Aeronautics and Astronautics, 1991.

[16] W. M. Farmer, J. D. Guttman, and F. J. Thayer. IMPS: an Interactive Mathematical Proof System. *Jorunal of Automated Reasoning*, 11:213–248, 1993.

[17] S. Feferman. Inductively presented systems and the formalization of metamathematics. In D. van Dalen, D. Lascar, and J. Smiley, editors, *Logic colloquium 80*, pages 95–128. North-Holland, 1982.

[18] S. Feferman. Finitary inductively presented logics. In *Logic colloquium 88*, pages 191–220. North-Holland, 1988.

[19] F. Giunchiglia. Multilanguage systems. In *Proceedings of AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*, 1991. Also IRST-Technical Report no. 9011-17.

[20] F. Giunchiglia. The **GETFOL** Manual - **GETFOL** version 1. Technical Report 92-0010, DIST - University of Genova, Genoa, Italy, 1992.

[21] F. Giunchiglia. Contextual reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, XVI:345–364, 1993.

[22] F. Giunchiglia, P. Pecchiari, and C. Talcott. An analysis of the reasoning structures and rules underlying the integration of linear arithmetic in to the Boyer-Moore prover, in preparation.

[23] J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specifications and programming. *Journal of the ACM*, 39(1):95–146, 1992.

[24] Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. Technical Report SRI-CSL-92-03, SRI International, Computer Science Laboratory, 1993. To appear in J.A. Goguen, editor, *Applications of Algebraic Specification Using OBJ*, Cambridge University Press.

[25] M. Gordon. HOL: A proof generating system for higher-order logic. In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification, and Synthesis.* Kluwer, 1987.

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

[27] R. Harper, H. Honsell, and G. Plotkin. A framework for defining logics. In *Second Annual Symposium on Logic in Computer Science.* IEEE, 1987.

[28] R. Harper, D. Sannella, and A. Tarlecki. Structure and representation in LF. In *Fourth Annual Symposium on Logic in Computer Science*, pages 226–237, 1989.

[29] *HUG'93 HOL User's Group Workshop*, 1993.

[30] *Principles of Hybrid Reasoning*, 1991 Fall Symposium. AAAI, 1991. Working notes distributed to conference attendees.

[31] M. V. Johnson and B. Hayes-Roth. Integrating diverse reasoning methods in the BBI blackboard control architecture. In J. Halpern, editor, *Proceedings of the Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 83–98. Morgan-Kaufman, 1987.

[32] J. Joyce and C. Seger. The HOL-Voss system: Model-Checking inside a General-Purpose Theorem-Prover. In HUG [29], pages 187–200.

[33] D. Kapur, D.R. Musser, and X. Niem. The tecton proof system. In *Proceedings of a Workshop on Formal Methods in Databases and Software Engineering*, Workshops in Computing, pages 54–79. Springer-Verlag, 1992.

[34] D. Kapur and H. Zhang. First-order theorem proving using conditional rewrite rules. In *9th International Conference on Automated Deduction*, number 310 in Lecture Notes in Computer Science, pages 1–20, 1988.

[35] D. Kapur and H. Zhang. An overview of RRL (rewrite rule laboratory). In *Third International Conf. of Rewriting Techniques and Applications*, 1989.

[36] J. Ketonen. EKL: An mathematically oriented proof checker. In R. E. Shostak, editor, *Seventh International Conference on Automated Deduction*, volume 170 of *Lecture Notes in Computer Science*, pages 65–79. Springer-Verlag, 1984.

[37] J. Ketonen and R. W. Weyhrauch. A decidable fragment of predicate calculus. *Theoretical Computer Science*, 32:297–307, 1982.

[38] J. A. Ketonen and J. Weening. EKL: An interactive proof checker. Technical Report CS Report STAN-CS-84-1006, Stanford University, 1984.

[39] S. C. Kleene. *Introduction to metamathematics.* North-Holland, Amsterdam, 1952.

[40] J. Klop. *Combinatory Reduction Systems.* Number 127 in Mathematical Centre Tracts. Mathematisch Centrum, Amsterdam, 1980.

[41] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. Technical Report SRI-CSL-93-05, SRI International, Computer Science Laboratory, August 1993.

[42] D. A. McAllester. *Ontic: A Knowledge Representationt System for Mathematics.* MIT Press, 1989.

[43] José Meseguer. General logics. In H.-D. Ebbinghaus et al., editor, *Logic Colloquium'87*, pages 275–329. North-Holland, 1989.

[44] D. Miller. A logic programming language with lambda abstraction, function variables, and simple unification. In P. Schroeder-Heister, editor, *Extensions of Logic Programming*, volume 475 of *Lecture Notes in Computer Science*, pages 253–281. Springer-Verlag, 1991.

[45] L. G. Monk. Inference rules using local contexts. *Journal of Automated Reasoning*, 4:445–462, 1988.

[46] K. L. Myers. Attachment methods for integration. In Hybrid [30], pages 49–55. Working notes distributed to conference attendees.

[47] J. Nagle and S. Johnson. Pratical program verification for automatic program proving for real-time embedded software. Technical Report WDL-TR9859, Ford Aerospace and Communications Corp., December 1982.

[48] G. Nelson and D. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2), October 1979.

[49] T. Nipkow. Higher-order critical pairs. In *Sixth Annual Symposium on Logic in Computer Science*. IEEE, 1991.

[50] S. Owre, J. Rushby, and N. Shankar. PVS: A prototype verification system, 1992.

[51] L. C. Paulson. *Logic and Computation*. Cambridge University Press, 1987.

[52] L. C. Paulson. Isabelle: The next 700 theorem provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–385. Academic Press, 1990.

[53] D. Prawitz. *Natural Deduction: A Proof-theoretical Study*. Almquist and Wiksell, 1965.

[54] T. S. Redmond. Simplifier description. Technical Report Aerospace Report No. ATR-86A(8554)-2, The Aerospace Corporation, 1987.

[55] K. Schneider, R. Kumar, and T. Kropf. Automating most parts of hardware proofs in HOL. In *Proceedings of the Third Workshop on Computer Aided Verification, CAV 91*, pages 454–465, 1991.

[56] D. S. Scott. Rules and derived rules. In S. Stenlund, editor, *Logical Theory and Semantic Analysis*, pages 147–161. D. Reidel, 1974.

[57] J. R. Shoenfield. *Mathematical Logic*. Addison-Wesley, 1967.

[58] R. E. Shostak. Deciding combinations of theories. Technical Report Technical Report CSL-132, SRI International, 1982.

[59] Vishal Sikka. Integrating specialized procedures in proof systems. Technical Report Logic-94-3, Stanford University, Computer Science Department, 1994.

[60] M. Stickel. KLAUS automated deduction system. In *Proceedings of the ninth conference on automated deduction*, number 310 in Lecture Notes in Computer Science, pages 750–751. Springer-Verlag, 1988.

[61] Ian Sutherland and Richard Platek. A plea for logical infrastructure. In *TTCP XTP-1 Workshop on Effective Use of Automated Reasoning Technology in System Development*, pages 1–3, 1992.

[62] C. Talcott. accessible by anonymous ftp or www – URL = file://sail.stanford.edu/pub/clt/ARS.

[63] C. L. Talcott. Reasoning specialists should be logical services, not black boxes. In *Proceedings of CADE-12 workshop on Theory Reasoning in Automated Deduction*, pages 1–6, 1994.

[64] C.L. Talcott. A theory of binding structures and its applications to rewriting. *Theoretical Computer Science*, 112:99–143, 1993.

[65] The SRI Staff. The SRI specification and verification system, users guide. Technical report, SRI International, 1986.

[66] R. W. Weyhrauch. A Users Manual for Fol. Technical Report STAN-CS-77-432, Stanford University Computer Science Department, 1977.

[67] R. W. Weyhrauch. Prolegomena to a theory of formal reasoning. *Artificial Intelligence*, 13:133–170, 1980.

[68] R. W. Weyhrauch and C. L. Talcott. The logic of fol systems: Formulated in set theory. In M. Hagiya, N. D. Jones, and M. Sato, editors, *Festschrift in honor of Professor Satoru Takasu*, number 792 in Lecture Notes in Computer Science. Springer-Verlag, 1994.

Contents

1. Plug and Play Reasoning Devices – An Impossible Dream? . . 1
2. Logical Services and OMRSs 2

I Analysis of the Problem. 4
3. Existing systems 4
3.1. Single Logic Systems 4
3.2. Multi-logic Systems 5
4. Issues 6
4.1. Sequents 6
4.2. Rules 7
4.3. Reasoning Structures and Derivations 8
4.4. Integration of Reasoning Theories 9

II Technical Development. 10
5. Sequent Systems 10
5.1. Definition 11
5.2. Examples 12
6. Rules 14
6.1. Definition 14
6.2. Examples 14
7. Reasoning Theories 17
7.1. Definition 18
7.2. Examples 18
7.3. Composing Reasoning Theories 18
8. Reasoning Structures and Derivations 20
8.1. Reasoning Structures 20
8.2. Derivations 25
9. Operations on Reasoning Structures 26
9.1. Primitive Operations 26
9.2. Inference Rules as Operations on Reasoning Structures . 29

III. An Analysis of the Integration of Linear Arithmetic in NQTHM. . 31
10. The NQTHM theorem prover 31
11. Outline of our analysis of NQTHM 32
12. The Reasoning Theory Rth_{pNQTHM} 33
12.1. The Rth_{pNQTHM} Sequent System 35
12.2. The Rth_{pNQTHM} Rules 36
13. The Rth_{NQTHM} Reasoning Theory 38
13.1. The Rth_{L} Reasoning Theory 39
13.2. The modified Rth_{pNQTHM} Sequent System 40
13.3. The modified Rth_{pNQTHM} Rules 41
13.4. New Bridge Rules 42
14. Examples of Rth_{NQTHM} Reasoning Structures 43
IV Concluding Matter. 46
15. Related Work 47
16. Future Work 49
17. Proofs 52
17.1. Derivations 52
17.2. Operations 54
18. References 56