Inductive Theorem Proving in Theories Specified by Positive/Negative-Conditional Equations

Claus-Peter Wirth

Ulrich Kühler

SEKI-Report SR-95-15 (SFB)

December 13, 1995

Universität Kaiserslautern
Fachbereich Informatik
D-67663 Kaiserslautern

Abstract: We present an inference system for clausal theorem proving w.r.t. various kinds of inductive validity in theories specified by constructor-based positive/negative-conditional equations. The reduction relation defined by such equations has to be (ground) confluent, but need not be terminating. Our constructor-based approach is well-suited for inductive theorem proving in the presence of partially defined functions. The proposed inference system provides explicit induction hypotheses and can be instantiated with various wellfounded induction orderings. While emphasizing a well structured clear design of the inference system, our fundamental design goal is user-orientation and practical usefulness rather than theoretical elegance. The resulting inference system is comprehensive and relatively powerful, but requires a sophisticated concept of proof guidance, which is not treated in this paper.

This research was supported by the Deutsche Forschungsgemeinschaft, SFB 314 (D4-Projekt)

Contents

1 Introduction

1.1 What is Inductive Theorem Proving?

Given a set of first-order axioms 'R', one is often not only interested in propositions 'Γ' that are *deductive theorems* of R, i.e. that hold in all models of R: " R $\models \Gamma$ "; but also in those propositions that only hold in some specific sub-class of the class of models of R. Besides restricting the class of models with some required property, one may also consider those Γ where only " R $\models \Gamma\tau$ " is required for all τ taken from a specific set of (e.g. ground) substitutions. Notions of validity resulting from combinations of restrictions of these two kinds are usually called *inductive*, and the inductively valid propositions are named *inductive theorems*. In general, the set of inductive theorems is a proper superset of the set of deductive theorems because inductive theorems can rely on additional properties induced by the given notion of inductive validity. Typical such properties are term-generatedness ("no junk") and initiality or freeness ("no confusion").[1] Inductive theorems are called "inductive" because their (finite) proofs usually require mathematical induction to utilize term-generatedness. Inductive reasoning can be seen as extending deductive reasoning in that infinite deductive proofs may be represented in a finite cyclic form, as suggested in the following example, where $\Gamma(x_0, y)$ is a proposition over the natural numbers, and where the formulas below each line (sub-goals) imply the formula (goal) above: An infinite deductive proof of $\Gamma(x_0, y)$

$$\dfrac{\Gamma(x_0,y)}{\dfrac{\Gamma(0,y)}{\vdots} \quad \dfrac{\Gamma(\mathsf{s}(x_1),y)}{\dfrac{\Gamma(\mathsf{s}(0),y)}{\vdots} \quad \dfrac{\Gamma(\mathsf{s}(\mathsf{s}(x_2)),y)}{\dfrac{\Gamma(\mathsf{s}(\mathsf{s}(0)),y)}{\vdots} \quad \cdots} \quad \cdots} \quad \cdots}$$

(using $x_i = 0 \vee \exists x_{i+1}\colon\ x_i = \mathsf{s}(x_{i+1})$) should be captured in something like

$$\dfrac{\Gamma(x_0,y)}{\dfrac{\Gamma(0,y)}{\vdots} \quad \dfrac{\Gamma(\mathsf{s}(x_1),y)}{\dfrac{\Delta \Rightarrow \Gamma(\mathsf{s}(x_1),y)}{\vdots} \quad \dfrac{\Pi \Rightarrow \Gamma(\mathsf{s}(x_1),y)}{\vdots}}}$$

$$\dfrac{\Delta' \vee \Gamma(x_1,s)}{\text{(back to top)}} \qquad \dfrac{\Pi' \vee \Gamma(x_1,t)}{\text{(back to top)}} \quad \dfrac{\Pi''}{\vdots}$$

using $\Delta \vee \Pi$. This kind of *cyclic* argument — namely inferring $\Gamma(x_1, s)$ and $\Gamma(x_1, t)$ from $\Gamma(x_0, y)$ — is sound if for each (ground) instantiation of the theorem (here: $\Gamma(\mathsf{s}^m(\mathsf{0}), \mathsf{s}^n(\mathsf{0}))$) the deductive proof terminates. This can be guaranteed by requiring that each cycle in the proof (graph) *terminate*, i.e. its preconditions (usually called *induction hypotheses* — here: $\Gamma(x_1, s)$ and $\Gamma(x_1, t)$) be smaller than the *"induction" conclusion* (here: $\Gamma(x_0, y)$ or $\Gamma(\mathsf{s}(x_1), y)$) w.r.t. some wellfounded ordering, called *induction ordering* (here e.g. the

[1] Note that, in general, term-generatedness (just like wellfoundedness) is not first-order axiomatizable (cf. Enderton (1973), Gödel (1931)) and freeness is not even then finitely first-order axiomatizable when term-generatedness may be assumed (cf. Berghammer (1993)).

usual ordering on the natural numbers applied to the first argument of Γ). Thus, while the property of being an inductive theorem depends only on the specification (i.e. a term language and a set of axioms) and the choice of a specific notion of inductive validity, an inductive proof of an inductive theorem also depends on an additional parameter, namely some induction ordering which must be chosen during the proof appropriately.

1.2 Explicit versus Implicit Induction

Although there is no generally accepted characterization of the two paradigms of *explicit* and *implicit* induction in the research community, in Walther (1994) the following is said:

> Research on automated induction these days is based on two competing paradigms: *Implicit induction* (also termed *inductive completion*, *inductionless induction*, or, less confusingly, *proof by consistency*) evolved from the *Knuth-Bendix Completion Procedure* .... The other research paradigm ... is called *explicit induction* and resembles the more familiar idea of induction theorem proving using induction axioms.

In accordance with this view, we call the latter paradigm "*explicit*" because, in the underlying inference systems, every cyclic argument must be made explicit in a single inference step applying a so-called *induction rule*. Besides generating *induction base* formulas, this step joins induction hypotheses and conclusions in *induction step* formulas and explicitly guarantees the termination of their cycles by a sub-proof or -mechanism for the wellfoundedness of the induction ordering resulting from the step formulas. In the explicit induction proof corresponding to the example induction proof of section 1.1 the induction base formula is $\Gamma(0, y)$ and the induction step formulas are $\Delta \Rightarrow (\Gamma(x_1, s) \Rightarrow \Gamma(\mathsf{s}(x_1), y))$ and $\Pi \Rightarrow (\Gamma(x_1, t) \Rightarrow \Gamma(\mathsf{s}(x_1), y))$. The explicit induction proof then has the following form:

$$\dfrac{\Gamma(x_0, y)}{\dfrac{\Gamma(0, y)}{\vdots} \quad \dfrac{\Delta \Rightarrow (\Gamma(x_1, s) \Rightarrow \Gamma(\mathsf{s}(x_1), y))}{\vdots} \quad \dfrac{\Pi \Rightarrow (\Gamma(x_1, t) \Rightarrow \Gamma(\mathsf{s}(x_1), y))}{\vdots}}$$

Note that the first inference in this proof is an application of an *induction axiom* in the sense of Walther (1994), which is a comprehensive survey on explicit induction.

As the example induction proof of section 1.1 illustrates, the cyclic arguments and their termination in *implicit* induction proofs need not be confined to single inference steps, as is in explicit induction. Therefore, the induction axioms corresponding to the cyclic arguments in a finite implicit induction proof can only be determined by analyzing the whole proof, whereas in the case of explicit induction each applied induction axiom is given by an application of the induction rule. Since Bachmair (1988), refutational completeness (cf. Appendix A) has been emphasized by various authors in the field of implicit induction: In general, the set of inductively valid theorems is not enumerable for all significant notions of inductive validity; therefore refutational completeness is an optimal theoretical quality of inference systems for inductive theorem proving. Moreover, non-terminating proof attempts in refutationally complete inference systems could be considered successful proofs. While

the ability of an inductive theorem prover to detect invalid formulas is important under a practical aspect (cf. section 3.10), refutational completeness does not help in finding finite proofs for inductively valid formulas.

To succeed in proving an inductive theorem in finite time, implicit inductive theorem provers have to solve the same problems as explicit inductive theorem provers, namely to find a finite cyclic representation for an infinite deductive proof as well as an induction ordering guaranteeing the termination of its cycles. Therefore, if a theorem prover with sufficient deductive power fails to show an inductive theorem, possible reasons may be a failure to construct the proper reasoning cycles or to establish their termination. As will be shown in the following, inference systems for implicit induction can be made powerful enough to express the appropriate cyclic arguments, but a certain kind of bookkeeping (cf. the "weights" in section 3.1) of the ordering information for controlling the various applications of induction hypotheses is necessary for establishing their termination, cf. the examples 27, 28, 29, and 30 in section 3. Inference systems for explicit induction on the other hand do not require this kind of bookkeeping since the ordering information is used only locally within single explicit induction steps. Inference systems for explicit induction have the disadvantage, however, that adequate instantiations of the induction hypotheses need to be guessed when the induction step formulas are synthesized, cf. the terms s and t in our above example. In general, it can be rather difficult to choose these instantiations before the induction conclusion has been substantially simplified, cf. Protzen (1994) for a simple example.

All in all, we feel that, from an abstract point of view and beyond the technicalities usually involved, most induction proofs can be expressed in either of the two paradigms with essentially the same cyclic arguments. Moreover, we think that a combination of methods, techniques, and heuristics from both research paradigms is possible and will be beneficial for the automation of inductive theorem proving.

1.3 Our Approach

In this paper we present an inference system for proving inductive theorems w.r.t. constructor-based positive/negative-conditional equational specifications and w.r.t. various important notions of inductive validity. Such specifications were introduced in Wirth & Gramlich (1993) and notions of inductive validity were discussed in Wirth & Gramlich (1994b).

Based on experiences with more or less automated inductive theorem provers, such as NQTHM, INKA, RRL, UNICOM, SPIKE, etc.,[2] we have come to adopt a rather pragmatic viewpoint with respect to inductive theorem proving: Successful use of an inductive theorem prover in "real-life" problem domains has not been possible yet without a knowledgeable human user who can interact with the system on various levels. Accordingly, we think that even the development of the *theoretical* concepts of a new theorem prover — including its inference system — should begin with a clear emphasis on user interaction, whereas automatic proof guidance is seen as a long-term goal. Therefore, the following two requirements are main design goals for our inference system:

[2] Cf. Boyer & Moore (1988), Biundo & al. (1986), Kapur & Zhang (1989), Gramlich & Lindner (1991), Bouhoula & Rusinowitch (1995), resp.

1. We expect the inference system to comply with human (inductive) proof techniques in that it enables users to naturally express their proof ideas in terms of the inference system.

2. Users should have no difficulties in understanding and searching for formal proofs represented with the inference system, no matter whether they try to follow them on the mere syntactic level or try to grasp overall "strategic" aspects of the proofs.

Refining the first design goal we obtain the following requirements:

All proof problems and sub-problems, defining equations, lemmas, and hypotheses should be represented homogeneously, such that all requirements of any inference rule are expressed in the same language. This enables the user to utilize the full power of the whole inference system on all problems and sub-problems and to choose freely and flexibly between eager or lazy strategies for any proof problem.

Another important point is that the inference system includes *inference rules for most elementary proof steps* such that the user can force the prover to follow his proof ideas as closely as possible. We consider transparency of atomic inference steps to be more important than having (derived) higher-level inference rules. Applied in non-trivial proof problems higher-level inference rules tend to be too restrictive, while an inference system comprising a multitude of simple "fine-grain" inference rules can be used to (interactively) construct even very difficult proofs.

Refining the second design goal we obtain the following requirements:

The inference system should support a *natural flow of information* in the sense that a certain decision can be delayed until the state of the proof provides sufficient information to make a successful decision. One example of unnatural flow of information is instantiating induction hypotheses in induction step formulas of explicit induction long before the hypotheses become applicable, cf. Protzen (1994).[3] Another example of unnatural flow of information is the γ-rule of sequent calculi or semantic tableaux (without free variables), cf. Smullyan (1968), where instantiations have to be guessed long before it can be recognized which instantiations will make a proof attempt successful. Together with the homogeneous and flexible formulation of the inference rules, the natural flow of information should make it possible to replace a certain amount of proof planning based on some special abstractive representation with heuristic search based on the homogeneous representation in the inference system.

Moreover, in order to enable the user to understand and manipulate the current proof state a *mutual independence of the proof sub-problems* represented by the formulas at the open nodes of the proof graph is helpful. This independence (or locality) facilitates distributed and localized proof construction. Note that this kind of independence is not trivially satisfied. E.g., the branches of a free-variable tableau (cf. Fitting (1990), Baaz & Fermüller (1995)) need not be mutually independent, leading to the problem of *simultaneous* rigid E-unification which is undecidable in general, cf. Degtyarev & Voronkow (1995).

Another requirement that is very important for efficiency of automated as well as non-automated theorem proving is *goal-directedness*. Goal-directedness means that every

[3] Note that we do appreciate the heuristic knowledge in the induction rule of explicit induction systems. We just do not accept that the proof fails if the guessed instantiation was wrong.

problem in the graph of a proof attempt is connected with the theorem to be proved. For *inductive* theorem proving this is even more important than for deductive theorem proving, since without goal-directness finding appropriate induction hypotheses and lemmas that bridge the gap between induction hypotheses and conclusion seems impossible. Note that in inductive theorem proving, cutting with these *bridge lemmas* cannot be eliminated in the form given by Gentzen's Hauptsatz for deductive proofs.

As a first step towards achieving the above design goals, we have an inference system in mind that explicitly provides the concepts of *induction hypothesis* and *induction ordering*. With the latter concept, we associate means of supplying induction ordering conditions with sufficient expressiveness (i.e. explicit *weights*, cf. section 3.1) and methods of choosing adequate induction orderings among semantic orderings (resulting in "induction on values"), syntactic orderings (resulting in "induction on terms") or combinations of these types of orderings.

Concerning the concept of induction hypothesis, we intend an inference system that does not "hide" (applications of) induction hypotheses in single inference steps. We rather think of an inference system that "knows" what an induction hypothesis is, i.e. it includes inference rules that provide or apply induction hypotheses, given that certain ordering conditions resulting from these applications can be met by an induction ordering. Obviously, such an inference system is an inference system for implicit induction (as explained above). Furthermore, the intended inference system supports lazy generation of induction hypotheses and mutual induction. As a consequence, we obtain the following essential constraint for our inference system:

> The formulas used by the inference system must be capable of representing an induction hypothesis as a whole and in recognizable form. Not an inference rule nor input normalization may decompose a conjectured inductive theorem (into "sub"-formulas) before the induction hypotheses have been extracted from it.

This constraint helps settling the following question: *Which deductive inference system is best suited for an integration of implicit induction?* Note that in the more restricted framework of explicit induction, obtaining an inference system for inductive theorem proving from a deductive inference system is far simpler: Since induction is concentrated in a single inference rule then, this induction rule can be just added to the given deductive inference system without affecting the rest of the inference system. When integrating *implicit* induction, however, the whole inference system is affected.

Satisfying the above indented constraint with *refutational inference systems based on resolution or paramodulation* (cf. e.g. Bachmair & Ganzinger (1994)) seems difficult if not impossible. Suppose we want to prove the following inductive theorem stating a transitivity-like property of the "less"-predicate 'less' on the natural numbers:

$$\forall x\colon\ \forall y\colon\ \forall z\colon\ \Big(\ \mathsf{less}(\mathsf{s}(x), z) \vee \overline{\mathsf{less}(x, y)} \vee \overline{\mathsf{less}(y, z)}\ \Big).$$

Now for a refutational proof we first have to negate this, yielding

$$\exists x\colon\ \exists y\colon\ \exists z\colon\ \Big(\ \overline{\mathsf{less}(\mathsf{s}(x), z)} \wedge \mathsf{less}(x, y) \wedge \mathsf{less}(y, z)\ \Big).$$

Next we must skolemize such that satisfiability is invariant. For deductive satisfiability this simply yields the three unit clauses (with new constants 'a', 'b', 'c')

$$\overline{\mathsf{less}(\mathsf{s}(\mathsf{a}), \mathsf{c})}, \qquad \mathsf{less}(\mathsf{a}, \mathsf{b}), \qquad \mathsf{less}(\mathsf{b}, \mathsf{c}).$$

For inductive satisfiability we must instead refute something like (k, l, m denoting arbitrary globally constant natural numbers)

$$\overline{\mathsf{less}(\mathsf{s}(\mathsf{s}^k(0)),\mathsf{s}^m(0))}, \qquad \mathsf{less}(\mathsf{s}^k(0),\mathsf{s}^l(0)), \qquad \mathsf{less}(\mathsf{s}^l(0),\mathsf{s}^m(0)).$$

Besides the fact that this is beyond our term language, it is obvious that the original input theorem has been decomposed and when we want to use it as an induction hypothesis it is not recognizable as a whole anymore.

Sergey Yu. Maslov's inverse method (cf. Lifschitz (1989)) generalizes resolution in that input formulas can be super-clauses (i.e. disjunctions of super-literals) and need not be normalized to clausal form. Hence, one might suppose that the inverse method were less destructive by admitting

$$\overline{\mathsf{less}(\mathsf{s}(\mathsf{s}^k(0)),\mathsf{s}^m(0))} \wedge \mathsf{less}(\mathsf{s}^k(0),\mathsf{s}^l(0)) \wedge \mathsf{less}(\mathsf{s}^l(0),\mathsf{s}^m(0))$$

as the super-literal (i.e. a conjunction of literals) of a unit super-clause. The inverse method, however, does not address the problem of inductive skolemization. Moreover, *Maslov's general idea of inversion* (cf. Maslov (1971)) seems not to be appropriate for *inductive* theorem proving: How can we invert proofs of the form as presented in section 1.1? Inversion here means beginning at the bottom.[4] An infinite deductive proof has no (finite) bottom at which to start. The bottom of an inductive proof (graph) includes formulas subsumed by the theorem itself. Thus, it is obvious that the theorem must be a prover formula, i.e. of the type of formulas that may occur at the nodes of proof graphs, as opposed to the formulas generating inference rules (such as $x_i = 0 \vee \exists x_{i+1}\colon\ x_i = \mathsf{s}(x_{i+1})$ or $\Delta \vee \Pi$ in our inductive proof of section 1.1) (cf. Lifschitz (1989)). As does resolution, the inverse method restricts prover formulas to be clauses. Therefore, when trying to integrate induction into the inverse method, we find ourselves in the more familiar framework of *non-refutational resolution and paramodulation*, where the resolution rule is applied to axioms only, and the goal is to infer a formula which subsumes the theorem. Non-refutational resolution seems not very appropriate for *deductive* theorem proving because it is not goal-directed. In the context of *inductive* theorem proving, however, non-refutational resolution could be considered goal-directed because it starts with the axioms *plus the theorem* (as induction hypotheses), and a formula that subsumes the theorem (as induction conclusion) is to be inferred. Nevertheless, the goal-directedness given by the induction hypotheses is not sufficient. Since numerous bridge lemmas may be applicable and applications of bridge lemmas tend to be "closer" to induction hypotheses than to the conclusion, it is practically impossible to find the appropriate ones unless the conclusion has been expanded to a large degree. Furthermore, non-refutational resolution is not goal-directed in the base cases and in those parts of the proof that lie below applications of induction hypotheses (see e.g. the right-most branch of the inductive proof in section 1.1). Finally, since most inductive proofs follow the recursive definitions of the specification, non-refutational resolution requires to paramodulate with the defining rules from right to left. This can result in a high branching degree for some of the non-recursive cases and make a proper combination of the cases of the definition difficult (i.e. "unfolding" (or "expanding") a function definition is easier than "folding" it). All in all, we conclude that non-refutational resolution and paramodulation are not adequate for *inductive* theorem proving either. Cf. Ganzinger & Stuber (1992), however, for a first step towards the difficult integration of implicit induction into a refined deductive paramodulation inference system.

[4] We do apologize for this brute force simplification of Sergey Yu. Maslov's fascinating idea.

Hilbert calculi are not adequate for (inductive) theorem proving because they are not goal-directed.

Natural deduction calculi (cf. Gentzen (1935), Prawitz (1965)) augment the proofs with assumptions that conflict with our concept of induction hypotheses.

Finally, since the central idea of *semantic tableaux* (cf. Fitting (1990)) is decomposing formulas, they inevitably destroy induction hypotheses (see the indented constraint above).

Now the only remaining family of well-known first-order deductive inference systems is that of *sequent calculi*, cf. Gentzen (1935), Lifschitz (1971). Fortunately, sequent cacluli do admit an integration of implicit induction in a way that is adequate in our opinion:

If provided with an inference rule for lemma application (cf. section 5.5), sequent calculi are *goal-directed* w.r.t. deductive theorem proving. By adding a powerful inference rule for applying induction hypotheses (cf. section 7.2), we obtain an inference system for inductive theorem proving that is optimally goal-directed w.r.t. the induction conclusion and sufficiently goal-directed w.r.t. the induction hypotheses.

Starting with the formula[5] to be proved, the problem of proving a formula (goal) is reduced to the problem of proving another set of formulas (sub-goals), which can be considered to be its sons[6]. Applying such reduction steps recursively results in a tree-like proof structure, which is augmented with cyclic arguments resulting from inference rules generated by induction hypotheses, similar to our inductive proof in section 1.1. A proof having the simple structure of a tree is *easy to understand* for users and a good basis for automatic tactics and heuristic proof search. Note that sequent calculi lie somewhere between the other two families of deductive inference systems that admit comprehensible proofs, namely natural deduction calculi and semantic tableaux: The close relation to natural deduction was already exhibited in Gentzen (1935), and tableaux can be regarded as the dual of Gentzen's sequent calculus, cf. Smullyan (1968).[7]

Concluding the discussion of the question which deductive inference system is best-suited for an integration of implicit induction, we indicate how *the above indented constraint can be satisfied* with sequent calculi. Contrary to semantic tableaux (where the formulas at some ancestor nodes must be included), any formula in a proof tree of a sequent calculus is completely *self-descriptive*. Therefore, any formula in the proof tree can be used as an induction hypothesis without regarding its position in the tree. It is obvious then that the hypotheses can be extracted before any inference rule or input normalization has decomposed the conjectured inductive theorems. Moreover, the induction hypotheses are represented as a whole and in a recognizable form.

[5] We simply speak of "*(prover) formulas*" instead of "sequents".

[6] or daughters

[7] Compared with natural deduction or tableaux, proofs in Gentzen's sequent calculus may be a little tiresome to write down by hand due to the many repetitions, but a computer can overcome this overhead by using a more tableaux-like hidden internal representation. For our convenience, we let the proof trees grow downwards, as is usual with tableaux. Thus, while readers familiar with tableaux must think in the dual in this paper, they do not have to invert their trees. On the other hand, readers familiar with sequent calculi must invert their trees and inference rules, whereas they do not have to think in the dual.

1.4 Organization of the Paper

Having presented our basic notions and notations in section 2, we describe inference systems for proof by mathematical induction on an abstract level in section 3. In section 4 we concretize this description for the case of clausal theorem proving w.r.t. various kinds of inductive validity in theories specified by constructor-based positive/negative-conditional equations. In sections 5 and 6 we present deductive, and in section 7 inductive inference rules for our concrete inference system. The main results of these sections are collected in section 9. After demonstrating the usefulness of a simple failure predicate in section 8 and after indicating future work in section 10 we conclude with section 11. The less important subject of refutational completeness is dealt with in Appendix A. A lengthy description of a sequent calculus for establishing deductive completeness results (cf. Theorem 99) can be found in Appendix B. The proofs of all the non-trivial lemmas and theorems are given in Appendix C.

2 Basic Notions and Notations

We assume the reader to be familiar with the basics of term rewriting (cf. e.g. Avenhaus & Madlener (1989), Klop (1992)) and algebraic specification (cf. e.g. Ehrig & Mahr (1985)).

We use '$\uplus$' for the union of disjoint classes and 'id' for the identity function. '$\mathbb{N}$' denotes the set of natural numbers and we define $\mathbb{N}_+ := \{\, n \in \mathbb{N} \mid 0 \neq n \,\}$. For a class R we define *domain*, *range*, and *image of a class* A by $\mathrm{dom}(R) := \{\, a \mid \exists b\colon (a,b) \in R \,\}$; $\mathrm{ran}(R) := \{\, b \mid \exists a\colon (a,b) \in R \,\}$; $R[A] := \{\, b \mid \exists a \in A\colon (a,b) \in R \,\}$. For example, if '$\sim$' is a (binary) relation, $\mathrm{dom}(\sim) \cup \mathrm{ran}(\sim)$ denotes the *field* of the relation. If it is an equivalence relation (cf. below), then $\mathrm{dom}(\sim) = \mathrm{ran}(\sim)$ and and the equivalence class of some $x \in \mathrm{dom}(\sim)$ is $\sim[\{x\}]$. This use of "$[\ldots]$" should not be confused with our habit of stating two definitions, lemmas, or theorems (and their proofs etc.) in one, where the parts between '[' and ']' are optional and are meant to be all included or all omitted. Furthermore, we use '$\emptyset$' to denote the empty set as well as the empty function or empty word.

Since our approach is based on the rigorous syntactic distinction of constructors, we have to be quite explicit about terms, substitutions, and algebras.

2.1 Terms

We will consider terms of fixed arity over many-sorted signatures. A *signature*

$$\mathrm{sig} = (\mathbb{F}, \mathbb{S}, \alpha)$$

consists of an enumerable set of function symbols $\mathbb{F}$, a finite set of sorts $\mathbb{S}$ (disjoint from $\mathbb{F}$), and a computable arity-function $\alpha\colon \mathbb{F} \to \mathbb{S}^+$. For $f \in \mathbb{F}$ its arity $\alpha(f)$ is the list of argument sorts augmented by the sort of the result of f; to ease reading we will sometimes insert a '$\longrightarrow$' between a nonempty list of argument sorts and the result sort.

A *constructor sub-signature of the signature* $(\mathbb{F}, \mathbb{S}, \alpha)$ is a signature

$$\mathrm{cons} = (\mathbb{C}, \mathbb{S}, \alpha|_{\mathbb{C}})$$

such that the set $\mathbb{C}$ is a decidable subset of $\mathbb{F}$. $\mathbb{C}$ is called the set of *constructor symbols*; the complement $\mathbb{N} = \mathbb{F} \setminus \mathbb{C}$ is called the set of *non-constructor symbols*.

Example 1 (Signature with Constructor Sub-Signature)

$$\begin{aligned}
\mathbb{C} &= \{\mathsf{0}, \mathsf{s}, \mathsf{false}, \mathsf{true}, \mathsf{nil}, \mathsf{cons}\} \\
\mathbb{N} &= \{\mathsf{dl}, \mathsf{rc}, \mathsf{br}, +, -, \mathsf{ack}, \mathsf{switch}, \mathsf{swatch}, \mathsf{leq}, \mathsf{less}, \mathsf{mbp}, \mathsf{p}, \mathsf{q}\} \\
\mathbb{S} &= \{\mathsf{nat}, \mathsf{bool}, \mathsf{list}\}
\end{aligned}$$

$$\begin{aligned}
\alpha(\mathsf{0}) &= && && \mathsf{nat} \\
\alpha(\mathsf{s}) = \alpha(\mathsf{switch}) = \alpha(\mathsf{swatch}) &= \mathsf{nat} && \longrightarrow && \mathsf{nat} \\
\alpha(+) = \alpha(-) = \alpha(\mathsf{ack}) &= \mathsf{nat}\ \mathsf{nat} && \longrightarrow && \mathsf{nat} \\
\alpha(\mathsf{false}) = \alpha(\mathsf{true}) &= && && \mathsf{bool} \\
\alpha(\mathsf{p}) &= \mathsf{nat} && \longrightarrow && \mathsf{bool} \\
\alpha(\mathsf{leq}) = \alpha(\mathsf{less}) = \alpha(\mathsf{q}) &= \mathsf{nat}\ \mathsf{nat} && \longrightarrow && \mathsf{bool} \\
\alpha(\mathsf{mbp}) &= \mathsf{nat}\ \mathsf{list} && \longrightarrow && \mathsf{bool} \\
\alpha(\mathsf{nil}) &= && && \mathsf{list} \\
\alpha(\mathsf{cons}) = \alpha(\mathsf{dl}) = \alpha(\mathsf{rc}) &= \mathsf{nat}\ \mathsf{list} && \longrightarrow && \mathsf{list} \\
\alpha(\mathsf{br}) &= \mathsf{list}\ \mathsf{list} && \longrightarrow && \mathsf{list}
\end{aligned}$$

A *variable-system for a signature* $(\mathbb{F}, \mathbb{S}, \alpha)$ is an $\mathbb{S}$-sorted family of decidable sets of variable symbols which are mutually disjoint and disjoint from $\mathbb{F}$. By abuse of notation we will use the symbol 'X' for an $\mathbb{S}$-sorted family to denote not only the family $X = (X_s)_{s\in\mathbb{S}}$ itself, but also the union of its *ranges*: $\bigcup_{s\in\mathbb{S}} X_s$. As the basis for our terms throughout the whole paper we assume two fixed disjoint variable-systems $\mathrm{V}_{\mathrm{SIG}}$ of *general variables* and $\mathrm{V}_{\mathrm{CONS}}$ of *constructor variables* such that $\mathrm{V}_{\mathrm{SIG},s}$ as well as $\mathrm{V}_{\mathrm{CONS},s}$ contain infinitely many elements for each $s \in \mathbb{S}$. $\mathcal{T}(\mathrm{sig}, \mathrm{V}_{\mathrm{SIG}} \uplus \mathrm{V}_{\mathrm{CONS}})$ denotes the $\mathbb{S}$-sorted family of all (well-sorted) *(variable-mixed) terms* over $\mathrm{sig}/\mathrm{V}_{\mathrm{SIG}} \uplus \mathrm{V}_{\mathrm{CONS}}$, while $\mathcal{GT}(\mathrm{sig})$ denotes the $\mathbb{S}$-sorted family of all (well-sorted) *ground terms* over sig. Similarly, $\mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{SIG}} \uplus \mathrm{V}_{\mathrm{CONS}})$ denotes the $\mathbb{S}$-sorted family of all *(variable-mixed) constructor terms*, $\mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$ denotes the $\mathbb{S}$-sorted family of all *pure constructor terms*, while $\mathcal{GT}(\mathrm{cons})$ denotes the $\mathbb{S}$-sorted family of all *constructor ground terms*. To avoid problems with empty sorts, we assume $\mathcal{GT}(\mathrm{cons})$ to have nonempty ranges only.

As exhibited in Avenhaus & Becker (1992), it is adequate to describe our terms, substitutions, and algebras within the order-sorted framework in the style of Gogolla (1983) or Smolka & al. (1989): Take $\{\mathrm{SIG}, \mathrm{CONS}\}\times\mathbb{S}$ for the sorts with the sort declaration that for each $s \in \mathbb{S}$ the sort (CONS, s) is a sub-sort of the sort (SIG, s); and replace each arity declaration of the form $\alpha(f) = s_0 \ldots s_{n-1} \longrightarrow s_n$ with the arity declaration

$$\alpha(f) \ni (\mathrm{SIG}, s_0) \ldots (\mathrm{SIG}, s_{n-1}) \longrightarrow (\mathrm{SIG}, s_n);$$

moreover, for each $f \in \mathbb{C}$ add the arity declaration

$$\alpha(f) \ni (\mathrm{CONS}, s_0) \ldots (\mathrm{CONS}, s_{n-1}) \longrightarrow (\mathrm{CONS}, s_n).$$

According to this order-sorted framework we define the following. A *variable-system for a signature* $(\mathbb{F}, \mathbb{S}, \alpha)$ *with constructor sub-signature* is a $\{\mathrm{SIG}, \mathrm{CONS}\}\times\mathbb{S}$-sorted family $\mathrm{X} = (\mathrm{X}_{\varsigma,s})_{(\varsigma,s)\in\{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}}$ of decidable sets that are mutually disjoint and disjoint from $\mathbb{F}$. Note that $\mathrm{V} := (\mathrm{V}_{\varsigma,s})_{(\varsigma,s)\in\{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}}$ provides us with such a variable-system. For all other such variable-systems X in this paper we require $\forall(\varsigma, s)\in\{\mathrm{SIG}, \mathrm{CONS}\}\times\mathbb{S}$: $\mathrm{X}_{\varsigma,s}\subseteq\mathrm{V}_{\varsigma,s}$. We use $\mathcal{V}(A)$ to denote the $\{\mathrm{SIG}, \mathrm{CONS}\}\times\mathbb{S}$-sorted family of variables occurring in a structure A (e.g. a term or a set or list of terms). We define $\mathcal{T}(\mathrm{X}) = (\mathcal{T}(\mathrm{X})_{\varsigma,s})_{(\varsigma,s)\in\{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}}$ by $(s\in\mathbb{S})$: $\mathcal{T}(\mathrm{X})_{\mathrm{SIG},s} := \mathcal{T}(\mathrm{sig}, \mathrm{X})_s$ and $\mathcal{T}(\mathrm{X})_{\mathrm{CONS},s} := \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}}\cap\mathrm{X})_s$. To avoid confusion: Note that $\mathcal{T}(\mathrm{X})_{\mathrm{CONS},s} \subseteq \mathcal{T}(\mathrm{X})_{\mathrm{SIG},s}$ for $s \in \mathbb{S}$, whereas $\mathrm{V}_{\mathrm{CONS},s} \cap \mathrm{V}_{\mathrm{SIG},s} = \emptyset$. Furthermore, we use $\mathcal{GT}$ as a shorthand for $\mathcal{T}(\emptyset)$ as well as $\mathcal{T}$ for $\mathcal{T}(\mathrm{V})$. Our custom of reusing the symbol of a family for the union of its ranges now allows us to write $\mathcal{T}$ as a shorthand for $\mathcal{T}(\mathrm{sig}, \mathrm{V}_{\mathrm{SIG}} \uplus \mathrm{V}_{\mathrm{CONS}})$.

For a term $t \in \mathcal{T}$ we denote by $\mathcal{POS}(t)$ the *set of its positions* (which are lists of positive natural numbers) and by t/p the subterm of t at position p. We partition $\mathcal{POS}(t)$ into the *set of variable positions* $\mathrm{V}\mathcal{POS}(t) := \{\ p\in\mathcal{POS}(t) \mid t/p\in\mathrm{V}\ \}$ and the *set of non-variable* (or function) *positions* $\mathbb{F}\mathcal{POS}(t) := \{\ p\in\mathcal{POS}(t) \mid t/p\notin\mathrm{V}\ \}$. By $t[p \leftarrow t']$ we denote the result of replacing t/p with t' at position p in t. p and q are called *parallel*, written $p \parallel q$, if neither p is a prefix of q, nor q a prefix of p. For $\Pi \subseteq \mathcal{POS}(t)$ with $\forall p, q \in \Pi$: $(p = q \vee p \parallel q)$ we denote by $t[\, p \leftarrow t'_p \mid p \in \Pi\,]$ the result of replacing, for each $p \in \Pi$, the subterm at position p in the term t with the term t'_p. t is *linear* if $\forall p, q \in \mathrm{V}\mathcal{POS}(t)$: $(t/p = t/q \Rightarrow p = q)$.

2.2 Substitutions

The set of *substitutions* from X to a $\{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}$-sorted family of sets $T = (T_{\varsigma,s})_{(\varsigma,s)\in\{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}}$ is defined to be

$$\mathcal{SUB}(\mathrm{X},T) := \{\ \sigma\colon\ \mathrm{X}\to T\ \mid\ \forall(\varsigma,s)\in\{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}\colon\ \forall x\in\mathrm{X}_{\varsigma,s}\colon\ \sigma(x)\in T_{\varsigma,s}\ \}.$$

Important sets of substitutions are $\mathcal{SUB}(\mathrm{V},\mathcal{T})$ and $\mathcal{SUB}(\mathrm{V},\mathcal{GT})$. The definition is consistent with the notion of substitutions in our order-sorted framework from above. A fortiori we get $\forall\sigma\in\mathcal{SUB}(\mathrm{V},\mathcal{T})\colon\ \forall(\varsigma,s)\in\{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}\colon\ \forall t\in\mathcal{T}_{\varsigma,s}\colon\ \ t\sigma\in\mathcal{T}_{\varsigma,s}$.

Sometimes it makes sense to allow the substitution of constructor variables with non-constructor terms: The set of *generalized substitutions* from X to T is defined to be

$$\mathcal{GENSUB}(\mathrm{X},T) := \{\ \sigma\colon\ \mathrm{X}\to T\ \mid\ \forall(\varsigma,s)\in\{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}\colon\ \forall x\in\mathrm{X}_{\varsigma,s}\colon\ \sigma(x)\in T_{\mathrm{SIG},s}\ \}.$$

In our applications we will always have $\forall s\in\mathbb{S}\colon\ T_{\mathrm{CONS},s}\subseteq T_{\mathrm{SIG},s}$ and therefore $\mathcal{SUB}(\mathrm{X},T)\subseteq\mathcal{GENSUB}(\mathrm{X},T)$. Thus, the name "generalized" is justified.

Let E be a finite set of equations (i.e. pairs of terms of equal sort) and X a finite subset of V. A substitution $\sigma\in\mathcal{SUB}(\mathrm{V},\mathcal{T})$ is called *a unifier for* E if $E\sigma\subseteq\mathrm{id}$. Such a unifier is called *most general on* X if for each unifier μ for E there is some $\tau\in\mathcal{SUB}(\mathrm{V},\mathcal{T})$ such that $(\sigma\tau)|_{\mathrm{X}} = \mu|_{\mathrm{X}}$. If E has a unifier, then it also has a most general unifier[8] on X, denoted by $\mathrm{mgu}(E,\mathrm{X})$.

Two terms t,t' are *unifiable* if $\{(t,t')\}$ has a unifier. They are *clashing* if there is some $p\in\mathbb{FPOS}(t)\cap\mathbb{FPOS}(t')$ such that the head function symbols of t/p and t'/p differ. Note that two terms cannot be both unifiable and clashing.

2.3 Relations

A binary relation '$\longrightarrow$' is *irreflexive* if $\mathrm{id}\cap\longrightarrow\ =\emptyset$. It is *A-reflexive* if $\mathrm{id}|_A\subseteq\longrightarrow$. Simply speaking of a *reflexive* relation we refer to the biggest A that is appropriate in the local context.

The reflexive, symmetric, transitive, and reflexive & transitive closure of $\longrightarrow$ will be denoted by $\xrightarrow{=}$, $\longleftrightarrow$, $\xrightarrow{+}$, and $\xrightarrow{*}$, respectively.[9]

$\longrightarrow$ is *confluent below* u if $\forall v,w\colon\ (v\xleftarrow{*}u\xrightarrow{*}w\Rightarrow v{\downarrow}w)$. $\longrightarrow$ is *confluent* if it is confluent below all u.

$\longrightarrow$ is called *terminating below* u if there is no $s\colon\ \mathbb{N}\to\mathrm{dom}(\longrightarrow)$ such that $u=s_0\ \wedge\ \forall i\in\mathbb{N}\colon\ s_i\longrightarrow s_{i+1}$. It is *terminating* if it is terminating below all u.

---

[8] For this most general unifier σ we could, as usual, even require $\sigma\sigma=\sigma$ but *not* $\mathcal{V}(\sigma[\mathcal{V}(E)])\subseteq\mathcal{V}(E)$. To see that the latter requirement is not possible, consider $x,y\in\mathrm{V}_{\mathrm{CONS},\mathsf{nat}}$; $Y\in\mathrm{V}_{\mathrm{SIG},\mathsf{nat}}$; a most general unifier for $\{(x,\ \mathsf{s}(Y))\}$ has to be something like $\{\ x\mapsto\mathsf{s}(y),\ Y\mapsto y\ \}$. Note that this problem would not occur if we restricted the variables in our terms either to be from $\mathrm{V}_{\mathrm{SIG}}$ only (cf. Wirth & Gramlich (1993)) or from $\mathrm{V}_{\mathrm{CONS}}$ only (cf. Avenhaus & Becker (1992)).

[9] Note that this is actually an abuse of notation since A^+ now denotes the transitive closure of A as well as the set of nonempty words over A and since A^* now denotes the reflexive & transitive closure of A as well as the set of words over A.

Let $\mathrm{X} \subseteq \mathrm{V}$. Let $\mathrm{T} \subseteq \mathcal{T}$. A relation R on $\mathcal{T}$ is called:

sort-invariant if $\forall n \in \mathbb{N}\colon\ \forall (t_0, \ldots, t_{n-1}) \in R\colon\ \exists s \in \mathbb{S}\colon\ \forall i \prec n\colon\ t_i \in \mathcal{T}_{\mathrm{SIG},s}$.

X-*stable (w.r.t. substitution)* if
$$\forall n \in \mathbb{N}\colon\ \forall (t_0, \ldots, t_{n-1}) \in R\colon\ \forall \sigma \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}))\colon\ (t_0\sigma, \ldots, t_{n-1}\sigma) \in R.$$

T-*monotonic* if $\forall (t', t'') \in R\colon\ \forall t \in \mathcal{T}\colon\ \forall p \in \mathcal{POS}(t)\colon$
$$\left(\begin{pmatrix} \exists s \in \mathbb{S}\colon\ t/p, t', t'' \in \mathcal{T}_{\mathrm{SIG},s} \\ \wedge\ \ t[p \leftarrow t'] \in \mathrm{T} \end{pmatrix} \Rightarrow \begin{pmatrix} (t[p \leftarrow t'], t[p \leftarrow t'']) \in R \\ \wedge\ \ t[p \leftarrow t''] \in \mathrm{T} \end{pmatrix} \right).$$

2.4 Orderings

By an (irreflexive) *ordering* '$<$' (on A) we mean an irreflexive and transitive binary relation (with $\mathrm{dom}(<) \cup \mathrm{ran}(<) \subseteq A$), sometimes called "strict partial ordering" etc. by other authors. As with all our asymmetric relation symbols we define $a>b$ if $b<a$. A *reflexive ordering* '$\leq$' *on* A is an A-reflexive, antisymmetric, and transitive relation. A *quasi-ordering* '$\lesssim$' *on* A is an A-reflexive and transitive relation. An *equivalence on* A is an A-reflexive, symmetric, and transitive relation.

The *equivalence* $\approx$ (on A) *of a quasi-ordering* $\lesssim$ (on A) is $\lesssim \cap \gtrsim$. The *ordering* $<$ *of a quasi-ordering or a reflexive ordering* $\lesssim$ is $\lesssim \setminus \gtrsim$. The *reflexive ordering on* A *of an ordering* $<$ on A is $< \cup \mathrm{id}|_A$. The *reflexive ordering of a quasi-ordering* $\lesssim$ on A is the reflexive ordering on A of the ordering of $\lesssim$.

An ordering $<$ or $>$ is called *wellfounded* if $>$ is terminating. A quasi-ordering or a reflexive ordering is called *wellfounded* if its ordering is wellfounded.

A *reduction ordering on* $\mathcal{T}$ is a V-stable, $\mathcal{T}$-monotonic, and wellfounded ordering. The *subterm ordering* '$\lhd_{\mathrm{ST}}$' *on* $\mathcal{T}$ (with reflexive ordering '$\trianglelefteq_{\mathrm{ST}}$') is the V-stable and wellfounded ordering defined by: $t \trianglelefteq_{\mathrm{ST}} t'$ if $\exists p \in \mathcal{POS}(t')\colon\ t = t'/p$. A *simplification ordering on* $\mathcal{T}$ is a reduction ordering on $\mathcal{T}$ containing $\lhd_{\mathrm{ST}}$. '$\prec$' denotes the ordering on the ordinal numbers. For further details on orderings cf. Dershowitz (1987).

2.5 Algebras

We define a sig/cons-*algebra* $\mathcal{A}$ *over the signature* $\mathrm{sig} = (\mathbb{F}, \mathbb{S}, \alpha)$ *with constructor sub-signature* $\mathrm{cons} = (\mathbb{C}, \mathbb{S}, \alpha|_{\mathbb{C}})$ to be a function defined on $\mathbb{F} \uplus (\{\mathrm{SIG}, \mathrm{CONS}\} \times \mathbb{S})$ with $\forall s \in \mathbb{S}\colon\ (\ \emptyset \neq \mathcal{A}_{\mathrm{CONS},s} \subseteq \mathcal{A}_{\mathrm{SIG},s}\ )$ and
$$f^{\mathcal{A}} : \mathcal{A}_{\mathrm{SIG},s_0} \times \cdots \times \mathcal{A}_{\mathrm{SIG},s_{n-1}} \rightarrow \mathcal{A}_{\mathrm{SIG},s_n} \quad \text{for } f \in \mathbb{F} \text{ with } \alpha(f) = s_0 \ldots s_n,$$
$$c^{\mathcal{A}}[\mathcal{A}_{\mathrm{CONS},s_0} \times \cdots \times \mathcal{A}_{\mathrm{CONS},s_{n-1}}] \subseteq \mathcal{A}_{\mathrm{CONS},s_n} \text{ for } c \in \mathbb{C} \text{ with } \alpha(c) = s_0 \ldots s_n.$$
We write $f^{\mathcal{A}}$ instead of $\mathcal{A}(f)$ for $f \in \mathbb{F}$. For $s \in \mathbb{S}$ we call $\mathcal{A}_{\mathrm{SIG},s}$ the *universe of* $\mathcal{A}$ *for the sort* s and $\mathcal{A}_{\mathrm{CONS},s}$ the *constructor sub-universe of* $\mathcal{A}$ *for the sort* s. The sig/cons-algebras of this definition are nothing but the order-sorted algebras over the order-sorted signature exhibited in section 2.1. A sig/cons-algebra $\mathcal{A}$ is called *trivial* if $\forall s \in \mathbb{S}\colon\ |\mathcal{A}_{\mathrm{SIG},s}| = 1$.

A (total) sig/cons-*homomorphism* $h::\mathcal{A}\to\mathcal{B}$ from a sig/cons-algebra $\mathcal{A}$ to a sig/cons-algebra $\mathcal{B}$ is an $\mathbb{S}$-sorted family $h = (h_s)_{s\in\mathbb{S}}$ of functions $h_s\colon \mathcal{A}_{\mathrm{SIG},s} \to \mathcal{B}_{\mathrm{SIG},s}$ which are compatible with sig and cons: For $f\in\mathbb{F}$; $\alpha(f)=s_0\dots s_n$; $\forall i\prec n\colon a_i\in\mathcal{A}_{\mathrm{SIG},s_i}$:

$$h_{s_n}(f^{\mathcal{A}}(a_0,\dots,a_{n-1})) = f^{\mathcal{B}}(h_{s_0}(a_0),\dots,h_{s_{n-1}}(a_{n-1}))\quad ;$$

and for all $s\in\mathbb{S}$: $\quad h_s[\mathcal{A}_{\mathrm{CONS},s}]\subseteq\mathcal{B}_{\mathrm{CONS},s}$.

Taking the class of sig/cons-algebras for the class of objects and the class of sig/cons-homomorphisms for the class of arrows, we get the sig/cons-*homomorphism category of* sig/cons-*algebras*. The composition $hk::\mathcal{A}\to\mathcal{C}$ of $h::\mathcal{A}\to\mathcal{B}$ and $k::\mathcal{B}\to\mathcal{C}$ is defined by $hk := (h_s\circ k_s)_{s\in\mathbb{S}}$ and the identity homomorphism for $\mathcal{A}$ is $(\mathrm{id}|_{\mathcal{A}_{\mathrm{SIG},s}})_{s\in\mathbb{S}}::\mathcal{A}\to\mathcal{A}$.

Let $\mathrm{X}\subseteq\mathrm{V}$. We enlarge the $\{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}$-sorted family $\mathcal{T}(\mathrm{X})$ of section 2.1 to the *term algebra over* X *and* sig/cons/V by defining ($f\in\mathbb{F}$; $\alpha(f)=s_0\dots s_n$; $\forall i\prec n$: $t_i\in\mathcal{T}(\mathrm{X})_{\mathrm{SIG},s_i}$):

$$f^{\mathcal{T}(\mathrm{X})}(t_0,\dots,t_{n-1}) := f(t_0,\dots,t_{n-1}).$$

An $\mathcal{A}$-*valuation* κ *of* X is an element of

$$\mathcal{SUB}(\mathrm{X},\mathcal{A}) = \mathcal{SUB}(\ (\mathrm{X}\cap\mathrm{V}_{\varsigma,s})_{(\varsigma,s)\in\{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}}\ ,\ (\mathcal{A}_{\varsigma,s})_{(\varsigma,s)\in\{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}}\ )\ .$$

For $\kappa\in\mathcal{GENSUB}(\mathrm{X},\mathcal{A})$ we define the *evaluation function* $\mathcal{A}_\kappa$ on $\mathcal{T}(\mathrm{X})$ recursively by

$$\mathcal{A}_\kappa(x) = \kappa(x) \qquad (x\in\mathrm{X});$$
$$\mathcal{A}_\kappa(f(t_0,\dots,t_{n-1})) = f^{\mathcal{A}}(\mathcal{A}_\kappa(t_0),\dots,\mathcal{A}_\kappa(t_{n-1})) \quad (f\in\mathbb{F}).$$

In case of $\kappa\in\mathcal{SUB}(\mathrm{X},\mathcal{A})$ we get the *evaluation homomorphism* $\mathcal{A}_\kappa::\mathcal{T}(\mathrm{X})\to\mathcal{A}$. Note that here '$\mathcal{A}_\kappa$' actually denotes $(\mathcal{A}_\kappa|_{\mathcal{T}(\mathrm{X})_{\mathrm{SIG},s}})_{s\in\mathbb{S}}$. Similarly, we sometimes write 'κ' when we actually mean $(\kappa|_{\mathrm{V}_{\mathrm{SIG},s}\uplus\mathrm{V}_{\mathrm{CONS},s}})_{s\in\mathbb{S}}$.

Lemma 2 (Homomorphism-Lemma)
Let $h::\mathcal{B}\to\mathcal{C}$ *be a* sig/cons-*homomorphism and* $\kappa\in\mathcal{GENSUB}(\mathrm{X},\mathcal{B})$. *Now:*
$\mathcal{B}_\kappa h=\mathcal{C}_{\kappa h}$; *i.e.* $\forall s\in\mathbb{S}\colon\ \forall t\in\mathcal{T}(\mathrm{X})_{\mathrm{SIG},s}\colon\ h_s(\mathcal{B}_\kappa(t))=\mathcal{C}_{\kappa h}(t)$.

Lemma 3 (Substitution-Lemma)
Let $\mathcal{A}$ *be a* sig/cons-*algebra and* κ *an* $\mathcal{A}$-*valuation of* X. *Now:*

$$\forall t\in\mathcal{T}\colon\ \forall\mu\in\mathcal{GENSUB}(\mathrm{V},\mathcal{T}(\mathrm{X}))\colon\ \mathcal{A}_\kappa(t\mu)=\mathcal{A}_{\mu\mathcal{A}_\kappa}(t).$$

For being able to express that a sig/cons-algebra has no "junk" in its universes, we define: For $\mathrm{DUNNO}\in\{\mathrm{SIG},\mathrm{CONS}\}$; $\mathrm{dunno}\in\{\mathrm{sig},\mathrm{cons}\}$; a sig/cons-algebra $\mathcal{A}$ is called DUNNO:dunno-*term-generated* if $\forall s\in\mathbb{S}\colon\ \forall a\in\mathcal{A}_{\mathrm{DUNNO},s}\colon\ \exists t\in\mathcal{GT}(\mathrm{dunno})_s\colon\ a=\mathcal{A}(t)$. $\mathcal{A}$ is called dunno-*term-generated* if it is SIG:dunno-term-generated.

Since it is also important to compare sig/cons-algebras w.r.t. to their "confusion", we define $\lesssim_{\mathrm{H}}$ and $\lesssim_{\mathrm{CONS}}$ as (proper class) relations on sig/cons-algebras by $\mathcal{A}\lesssim_{\mathrm{H}}\mathcal{B}$ if there is a sig/cons-homomorphism from $\mathcal{A}$ to $\mathcal{B}$. $\mathcal{A}\lesssim_{\mathrm{CONS}}\mathcal{B}$ if there is a cons-homomorphism from the cons-algebra $\mathcal{A}|_{\mathbb{C}\uplus(\{\mathrm{CONS}\}\times\mathbb{S})}$ to $\mathcal{B}|_{\mathbb{C}\uplus(\{\mathrm{CONS}\}\times\mathbb{S})}$. We trivially get $\lesssim_{\mathrm{H}}\ \subseteq\ \lesssim_{\mathrm{CONS}}$ (by restriction of the homomorphism); and $\lesssim_{\mathrm{H}}$, $\lesssim_{\mathrm{CONS}}$ are quasi-orderings (by the homomorphisms exhibited for the category above). The corresponding equivalences, orderings, and reflexive orderings will be denoted by $\approx$, $<$, $\leq$, resp., with the corresponding subscript.

A sig/cons-*congruence* $\sim$ *on* $\mathcal{A}$ is an ordinary sig-congruence for $\mathcal{A}$ when $\mathcal{A}$ is considered to be a sig-algebra, i.e. an $\mathbb{S}$-sorted family $\sim\ =(\sim_s)_{s\in\mathbb{S}}$ of equivalences $\sim_s$ on $\mathcal{A}_{\mathrm{SIG},s}$ satisfying for each non-constant function symbol $f\in\mathbb{F}$ with $\alpha(f)=s_0\dots s_ns_{n+1}$ and $\forall i\preceq n\colon\ a_i\in\mathcal{A}_{\mathrm{SIG},s_i}$: If $a_j\sim_{s_j}b$ for some $j\preceq n$, then

$$f^{\mathcal{A}}(a_0,\dots,a_{j-1},a_j,a_{j+1},\dots,a_n)\ \sim_{s_{n+1}}\ f^{\mathcal{A}}(a_0,\dots,a_{j-1},b,a_{j+1},\dots,a_n)\ .$$

The *factor algebra of* $\mathcal{A}$ *modulo* $\sim$ is the sig/cons-algebra $\mathcal{B}$ (denoted by $\mathcal{A}/\sim$) given by:

$$\mathcal{B}_{\varsigma,s} := \{\ \sim_s[\{a\}] \mid\ a \in \mathcal{A}_{\varsigma,s}\ \} \quad (\ (\varsigma,s) \in \{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}\ );$$
$$f^{\mathcal{B}}(\sim_{s_0}[\{a_0\}],\ldots,\sim_{s_{n-1}}[\{a_{n-1}\}]) := \sim_{s_n}[\{f^{\mathcal{A}}(a_0,\ldots,a_{n-1})\}]$$
$$(f \in \mathbb{F};\ \ \alpha(f) = s_0\ldots s_n;\ \ \forall i \prec n\colon\ a_i \in \mathcal{A}_{\mathrm{SIG},s_i}).$$

The *canonical* sig/cons-*epimorphism of* $\mathcal{A}$ *modulo* $\sim$ is the sig/cons-homomorphism $k{::}\mathcal{A}\to\mathcal{A}/\sim$ given by $(s \in \mathbb{S};\ a \in \mathcal{A}_{\mathrm{SIG},s})\colon\ k_s(a) := \sim_s[\{a\}]$.

For a sig/cons-homomorphism $h{::}\mathcal{A}\to\mathcal{B}$ we define its *kernel* to be the sig/cons-congruence $\ker(h)$ given by $(s \in \mathbb{S};\ a,b \in \mathcal{A}_{\mathrm{SIG},s})\colon\ (a,b) \in \ker(h)_s$ if $h_s(a) = h_s(b)$.

Theorem 4 (Homomorphism-Theorem)
Let $h{::}\mathcal{A}\to\mathcal{C}$ *be a* sig/cons-*homomorphism. Let* $\sim$ *be a* sig/cons-*congruence on* $\mathcal{A}$ *with* $\forall s \in \mathbb{S}\colon\ \sim_s\ \subseteq \ker(h)_s$. *Define* $\mathcal{B} := \mathcal{A}/\sim$. *Let* $k{::}\mathcal{A}\to\mathcal{B}$ *be the canonical* sig/cons-*epimorphism of* $\mathcal{A}$ *modulo* $\sim$. *Now* $h = kl$ *uniquely defines an* $\mathbb{S}$*-sorted family of functions* $l = (l_s)_{s\in\mathbb{S}}$ *with* $\forall s \in \mathbb{S}\colon\ (l_s\colon\ \mathcal{B}_{\mathrm{SIG},s} \to \mathcal{C}_{\mathrm{SIG},s})$. *Furthermore, this* l *is a* sig/cons-*homomorphism* $l{::}\mathcal{B}\to\mathcal{C}$. *Moreover, if* $\sim\ = \ker(h)$ *holds, then* l_s *is injective for each* $s \in \mathbb{S}$, *i.e.* $l{::}\mathcal{B}\to\mathcal{C}$ *is monic in the* sig/cons-*homomorphism category of* sig/cons-*algebras.*

By concretion of notions of category theory to full sub-categories of the sig/cons-homomorphism category of sig/cons-algebras and to the forgetful functor we define for a class K of sig/cons-algebras; a sig/cons-algebra $\mathcal{A}$; $\mathrm{X}\subseteq\mathrm{V}$; and $\iota \in \mathcal{SUB}(\mathrm{X},\mathcal{A})$: $\mathcal{A}$ is *initial in* K if $\mathcal{A} \in \mathrm{K}$ and for each $\mathcal{B} \in \mathrm{K}$ there is a unique $h{::}\mathcal{A}\to\mathcal{B}$. $\mathcal{A}$ is *free for* K *over* X *w.r.t.* ι if for each $\mathcal{B} \in \mathrm{K}$ and $\kappa \in \mathcal{SUB}(\mathrm{X},\mathcal{B})$ there is a unique $h{::}\mathcal{A}\to\mathcal{B}$ with $\kappa = \iota h$. $\mathcal{A}$ is *free in* K *over* X *w.r.t.* ι if $\mathcal{A} \in \mathrm{K}$ and $\mathcal{A}$ is free for K over X w.r.t. ι.

Finally, we explain a technical trick that enables us to simplify our notions and shorten our proofs. Some algebras we will have to deal with are factor algebras of term algebras but others are not. Let $\sim$ be some sig/cons-congruence on $\mathcal{T}(\mathrm{X})$ and define $\mathcal{A} := \mathcal{T}(\mathrm{X})/\sim$. When we define $\iota \in \mathcal{SUB}(\mathrm{X},\mathcal{A})$ by $((\varsigma,s) \in \{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S};\ x \in \mathrm{X}_{\varsigma,s})\ \ x\iota := \sim_s[\{x\}]$, then (since $\mathcal{A}_\iota$ is surjective on $\mathcal{A}$ and by the Axiom of Choice) each $\mathcal{A}$-valuation $\kappa \in \mathcal{SUB}(\mathrm{V},\mathcal{A})$ can be written in the form $\sigma\mathcal{A}_\iota$ for some $\sigma \in \mathcal{SUB}(\mathrm{V},\mathcal{T}(\mathrm{X}))$. Some of our inference rules for factor algebras of term algebras depend on this σ. For general algebras, however, such a σ cannot exist in general. Instead of having one set of notions for term algebras and another set for general algebras, and instead of making a case distinction on this again and again, we prefer to assume all algebras to be factor algebras of term algebras by the following trick: For each sig/cons-algebra $\mathcal{B}$ we assume some $\mathcal{B}$-valuation $\iota_{\mathcal{B}} \in \mathcal{SUB}(\mathrm{X},\mathcal{B})$ such that $\mathcal{B}_{\iota_{\mathcal{B}}}{::}\mathcal{T}(\mathrm{X})\to\mathcal{B}$ is surjective. In the case of $\mathcal{B}$ being the factor algebra of a term algebra like $\mathcal{A}$ above, we assume this $\iota_{\mathcal{B}}$ to be just the ι from above. When, for some $(\varsigma,s) \in \{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}$, the cardinality of $\mathrm{X}_{\varsigma,s}$ is too small for the existence of a surjective mapping on $\mathcal{B}_{\varsigma,s}$ then we assume that $\mathrm{X}_{\varsigma,s}$ is extended by new variables until its cardinality is big enough. To indicate this we sometimes write $\mathrm{X}^{\mathcal{B}}$ etc. instead of X. Now, due to $\mathcal{B}_{\iota_{\mathcal{B}}}{::}\mathcal{T}(\mathrm{X}^{\mathcal{B}})\to\mathcal{B}$ being surjective and due to the Homomorphism-Theorem, $\mathcal{B}$ is isomorphic to $\mathcal{T}(\mathrm{X}^{\mathcal{B}})/\ker(\mathcal{B}_{\iota_{\mathcal{B}}})$. Thus w.l.o.g. we may assume that each sig/cons-algebra $\mathcal{B}$ is a factor algebra $\mathcal{T}(\mathrm{X})/\sim$ of a term algebra $\mathcal{T}(\mathrm{X})$ and that each $\mathcal{B}$-valuation $\kappa \in \mathcal{SUB}(\mathrm{V},\mathcal{B})$ can be written in the form $\sigma\mathcal{B}_{\iota_{\mathcal{B}}}$ for some $\sigma \in \mathcal{SUB}(\mathrm{V},\mathcal{T}(\mathrm{X}))$.

2.6 Atoms and Literals

Let $\mathrm{X} \subseteq \mathrm{V}$. Let '=' (binary, symmetric, sort-invariant), 'Def' (unary), and '$<$' (binary) be predicate symbols. Let 'Weight' be some abstract set of *weights*, cf. section 4.2.

The set of *atoms* over terms from $\mathcal{T}(\mathrm{sig},\mathrm{X})$ is defined as

$$\mathcal{AT}(\mathrm{sig},\mathrm{X}) := \{\ (u{=}v),\ (\mathrm{Def}\,u)\ \mid\ \exists s \in \mathbb{S}\colon\ u,v \in \mathcal{T}(\mathrm{sig},\mathrm{X})_s\ \}.$$

The set of *generalized atoms* is defined as

$$\mathcal{GENAT}(\mathrm{sig},\mathrm{X}) := \mathcal{AT}(\mathrm{sig},\mathrm{X}) \cup \{\ (\beth{<}\aleph)\ \mid\ \beth,\aleph \in \mathrm{Weight}\ \wedge\ \mathcal{V}(\beth,\aleph){\subseteq}\mathrm{X}\ \}.$$

The set of *literals* is defined as

$$\mathcal{LIT}(\mathrm{sig},\mathrm{X}) := \{\ A,\ \overline{A}\ \mid\ A \in \mathcal{AT}(\mathrm{sig},\mathrm{X})\ \}.$$

The set of *generalized literals* is defined as

$$\mathcal{GENLIT}(\mathrm{sig},\mathrm{X}) := \{\ A,\ \overline{A}\ \mid\ A \in \mathcal{GENAT}(\mathrm{sig},\mathrm{X})\ \}.$$

The set of *condition literals* is defined as

$$\mathcal{CONDLIT}(\mathrm{sig},\mathrm{X}) := \mathcal{LIT}(\mathrm{sig},\mathrm{X}) \setminus \{\ \overline{(\mathrm{Def}\,u)}\ \mid\ u \in \mathcal{T}(\mathrm{sig},\mathrm{X})\ \}.$$

For "$\overline{(u{=}v)}$" we also write "$(u{\neq}v)$". For $A \in \mathcal{GENAT}(\mathrm{sig},\mathrm{V})$ let "$\overline{\overline{A}}$" denote "A". A literal that is an atom is called a *positive* literal; otherwise it is called *negative*.

We extend the concept of "positions" from terms to literals by ($s \in \mathbb{S}$; $u_1,u_2 \in \mathcal{T}_{\mathrm{SIG},s}$; $A \in \mathcal{GENAT}(\mathrm{sig},\mathrm{V})$; $\beth,\aleph \in \mathrm{Weight}$; $i \in \{1,2\}$; $p \in \mathbb{N}_+^*$):

$$\begin{array}{lll|ll}
\mathcal{POS}(u_1{=}u_2) &:=& \{\ iq \mid i \in \{1,2\}\ \wedge\ q \in \mathcal{POS}(u_i)\ \} & (u_1{=}u_2)/ip &:= u_i/p \\
\mathcal{POS}(\mathrm{Def}\,u_1) &:=& \{\ 1q \mid q \in \mathcal{POS}(u_1)\ \} & (\mathrm{Def}\,u_1)/1p &:= u_1/p \\
\mathcal{POS}(\beth{<}\aleph) &:=& \emptyset & & \\
\mathcal{POS}(\overline{A}) &:=& \mathcal{POS}(A) & \overline{A}/p &:= A/p
\end{array}$$

Note that for $\lambda \in \mathcal{GENLIT}(\mathrm{sig},\mathrm{V})$ and $p \in \mathcal{POS}(\lambda)$ we have $\lambda/p \in \mathcal{T}$ and that we treat '$<$'-literals as "black boxes". A literal $\lambda \in \mathcal{LIT}(\mathrm{sig},\mathrm{V})$ is called *linear* if $\forall p,q \in \mathrm{V}\mathcal{POS}(\lambda)$: $(\lambda/p = \lambda/q \Rightarrow p = q)$.

The set $\mathcal{TERMS}(\lambda_0 \ldots \lambda_{n-1})$ of (top level) *terms of a list of literals* $\lambda_0 \ldots \lambda_{n-1}$ is recursively defined by:

$$\begin{array}{lll}
\mathcal{TERMS}(u_1{=}u_2) &:=& \{u_1,u_2\} \\
\mathcal{TERMS}(\mathrm{Def}\,u_1) &:=& \{u_1\} \\
\mathcal{TERMS}(\beth{<}\aleph) &:=& \emptyset \\
\mathcal{TERMS}(\overline{A}) &:=& \mathcal{TERMS}(A) \\
\mathcal{TERMS}(\lambda_0 \ldots \lambda_{n-1}) &:=& \bigcup_{i \prec n} \mathcal{TERMS}(\lambda_i)
\end{array}$$

Definition 5 (Rewriting Literals)
Let $s \in \mathbb{S}$ and $l,r \in \mathcal{T}_{\mathrm{SIG},s}$. Let $\lambda,\lambda' \in \mathcal{LIT}(\mathrm{sig},\mathrm{V})$.

λ *rewrites to* λ' *with the rule* $l{=}r$ if $\exists P$: $\left(\begin{array}{ll} & P \subseteq \mathcal{POS}(\lambda) \cap \mathcal{POS}(\lambda') \\ \wedge & \forall p \in P\colon\ \lambda/p = l \\ \wedge & \lambda' = \lambda[\,p \leftarrow r \mid p \in P\,] \end{array}\right)$.

2.7 Syntax of Rules and Formulas

Many authors impose rather strong restrictions on constructor equations, such as "no equations between constructors" ("free constructors") or "unconditional equations between constructors only". Compared to these, our restrictions are rather weak. In the definition below we restrict our constructor rules to contain no non-constructor function symbols, to be extra-variable free, and to contain no negative literals. This is important for our approach (cf. Lemma 20, Lemma 21, and Lemma 22) and should be kept in mind. To get a provisional idea, the reader should think a rule $l{=}r{\longleftarrow}C$ to express that l reduces to r when, roughly speaking, all literals in C hold.

Definition 6 (Syntax of CRS)
A (positive/negative-) *conditional rule system* (*CRS*) R over sig/cons/V is a finite subset of the *set of rules* over sig/cons/V, which is defined by $\mathcal{RUL}(\mathrm{sig},\mathrm{cons},\mathrm{V}) :=$

$$\left\{ \Big(\ (l,r),\ C\ \Big) \;\middle|\; \left(\begin{array}{ll} & (l{=}r) \in \mathcal{AT}(\mathrm{sig},\mathrm{V}) \\ \wedge & C \in (\mathcal{CONDLIT}(\mathrm{sig},\mathrm{V}))^* \\ \wedge & \left(\begin{array}{l} l \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{SIG}} \uplus \mathrm{V}_{\mathrm{CONS}}) \ \Rightarrow \\ \left(\begin{array}{ll} & \{r\} \cup \mathcal{TERMS}(C) \ \subseteq \ \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{SIG}} \uplus \mathrm{V}_{\mathrm{CONS}}) \\ \wedge & \mathcal{V}(\{r\} \cup \mathcal{TERMS}(C)) \ \subseteq \ \mathcal{V}(l) \\ \wedge & C \in (\mathcal{AT}(\mathrm{sig},\mathrm{V}))^* \end{array} \right) \end{array} \right) \end{array} \right) \right\}.$$

A rule $((l,r),\emptyset)$ with an empty condition will be written $l{=}r$. Note that $l{=}r$ differs from $r{=}l$ whenever the equation is used as a rule. A rule $((l,r),C)$ with condition C will be written $l{=}r{\longleftarrow}C$. We call l the *left-hand side* and r the *right-hand side* of the rule $l{=}r{\longleftarrow}C$. A rule $l{=}r{\longleftarrow}C$ is called a *constructor rule* if its left-hand side is a constructor term, i.e. $l \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{SIG}} \uplus \mathrm{V}_{\mathrm{CONS}})$. A rule is said to be *left-linear* (or else *right-linear*) if its left-hand (or else right-hand) side is a linear term. A rule $l{=}r{\longleftarrow}C$ is said to be *extra-variable free* if $\mathcal{V}(\{r\} \cup \mathcal{TERMS}(C)) \ \subseteq \ \mathcal{V}(l)$. The whole CRS R is said to have one of these properties if each of its rules has it.

Formulas are implicitly universally quantified disjunctive lists of generalized literals:

Definition 7 (Syntax of Formulas)
Let $\mathrm{X} \subseteq \mathrm{V}$. The elements of the set 'Form(sig, X)' of *formulas* (or *Gentzen clauses*) *over* sig/X are lists of generalized literals:

$$\mathrm{Form}(\mathrm{sig},\mathrm{X}) := (\mathcal{GENLIT}(\mathrm{sig},\mathrm{X}))^* \ .$$

Definition 8 (Equals and Contains)
We say that an atom A *equals* an atom B if $A = B$ or $\exists t,t'$: $\left(\begin{array}{ll} & A = (t{=}t') \\ \wedge & B = (t'{=}t) \end{array} \right)$.

A literal λ *equals* a literal λ' if the atom λ equals the atom λ' or the atom $\overline{\lambda}$ equals the atom $\overline{\lambda'}$. We say that a formula Γ *contains* a formula Π if for each literal λ occuring in Π there is a literal λ' in Γ such that λ' equals λ. We say that Γ *equals* Π if Γ contains Π and Π contains Γ.

In the following example we define a delete-function "$\mathsf{dl}(x,l)$" deleting all occurrences of x in the list l, a remove-copies-function "$\mathsf{rc}(x,l)$" removing repeated occurrences of x in the list l, a brushing function "$\mathsf{br}(k,l)$" removing repeated occurrences in the list l for all elements of the list k. Moreover, on the natural numbers, the operations of addition '+' and subtraction '–' (defined only partially on the constructor ground terms of the sort nat, due to a non-complete defining case distinction), the Ackermann function 'ack', the "less or equal" predicate 'leq' and the "less" predicate 'less'. The member-predicate "$\mathsf{mbp}(x,l)$" tests for x occurring in the list l. That the predicates 'p' and 'q' are also total on the constructor ground terms of the sort nat is not obvious but will be formally proved in Example 129.

Example 9 (continuing Example 1)

Let $x,y \in \mathrm{V}_{\mathrm{CONS},\mathsf{nat}}$ *and* $k,l \in \mathrm{V}_{\mathrm{CONS},\mathsf{list}}$.

R_9:

$$
\begin{array}{lll}
(\mathsf{dl}\,1) & \mathsf{dl}(x,\mathsf{nil}) & = \mathsf{nil} \\
(\mathsf{dl}\,2) & \mathsf{dl}(x,\mathsf{cons}(y,l)) & = \mathsf{dl}(x,l) \quad \longleftarrow \; x{=}y \\
(\mathsf{dl}\,2) & \mathsf{dl}(x,\mathsf{cons}(y,l)) & = \mathsf{cons}(y,\mathsf{dl}(x,l)) \quad \longleftarrow \; x{\neq}y \\
(\mathsf{rc}\,1) & \mathsf{rc}(x,\mathsf{nil}) & = \mathsf{nil} \\
(\mathsf{rc}\,2) & \mathsf{rc}(x,\mathsf{cons}(y,l)) & = \mathsf{cons}(y,\mathsf{dl}(x,l)) \quad \longleftarrow \; x{=}y \\
(\mathsf{rc}\,3) & \mathsf{rc}(x,\mathsf{cons}(y,l)) & = \mathsf{cons}(y,\mathsf{rc}(x,l)) \quad \longleftarrow \; x{\neq}y \\
(\mathsf{br}\,1) & \mathsf{br}(\mathsf{nil},l) & = l \\
(\mathsf{br}\,2) & \mathsf{br}(\mathsf{cons}(x,k),l) & = \mathsf{br}(k,\mathsf{rc}(x,l)) \\
(+1) & x+\mathsf{0} & = x \\
(+2) & x+\mathsf{s}(y) & = \mathsf{s}(x+y) \\
(-1) & x-\mathsf{0} & = x \\
(-2) & \mathsf{s}(x)-\mathsf{s}(y) & = x-y \\
(\mathsf{ack}\,1) & \mathsf{ack}(\mathsf{0},y) & = \mathsf{s}(y) \\
(\mathsf{ack}\,2) & \mathsf{ack}(\mathsf{s}(x),\mathsf{0}) & = \mathsf{ack}(x,\mathsf{s}(\mathsf{0})) \\
(\mathsf{ack}\,3) & \mathsf{ack}(\mathsf{s}(x),\mathsf{s}(y)) & = \mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y)) \\
(\mathsf{switch}\,1) & \mathsf{switch}(\mathsf{0}) & = \mathsf{s}(\mathsf{0}) \\
(\mathsf{switch}\,2) & \mathsf{switch}(\mathsf{s}(\mathsf{0})) & = \mathsf{0} \\
(\mathsf{switch}\,3) & \mathsf{switch}(\mathsf{s}(\mathsf{s}(x))) & = \mathsf{s}(\mathsf{s}(\mathsf{switch}(x))) \\
(\mathsf{swatch}\,1) & \mathsf{swatch}(\mathsf{s}(x)) & = \mathsf{s}(\mathsf{switch}(x)) \\
(\mathsf{leq}\,1) & \mathsf{leq}(\mathsf{0},y) & = \mathsf{true} \\
(\mathsf{leq}\,2) & \mathsf{leq}(\mathsf{s}(x),\mathsf{0}) & = \mathsf{false} \\
(\mathsf{leq}\,3) & \mathsf{leq}(\mathsf{s}(x),\mathsf{s}(y)) & = \mathsf{leq}(x,y) \\
(\mathsf{less}\,1) & \mathsf{less}(\mathsf{0},\mathsf{s}(y)) & = \mathsf{true} \\
(\mathsf{less}\,2) & \mathsf{less}(x,\mathsf{0}) & = \mathsf{false} \\
(\mathsf{less}\,3) & \mathsf{less}(\mathsf{s}(x),\mathsf{s}(y)) & = \mathsf{less}(x,y) \\
(\mathsf{mbp}\,1) & \mathsf{mbp}(x,\mathsf{nil}) & = \mathsf{false} \\
(\mathsf{mbp}\,2) & \mathsf{mbp}(x,\mathsf{cons}(y,l)) & = \mathsf{true} \quad \longleftarrow \; x{=}y \\
(\mathsf{mbp}\,3) & \mathsf{mbp}(x,\mathsf{cons}(y,l)) & = \mathsf{mbp}(x,l) \quad \longleftarrow \; x{\neq}y
\end{array}
$$

$$
\begin{array}{lll}
(\mathsf{p}1) & \mathsf{p}(0) & = \mathsf{true} \\
(\mathsf{p}2) & \mathsf{p}(\mathsf{s}(x)) & = \mathsf{true} \quad \longleftarrow \quad \mathsf{p}(x){=}\mathsf{true},\ \ \mathsf{q}(x,\mathsf{s}(x)){=}\mathsf{true} \\
(\mathsf{q}1) & \mathsf{q}(x,0) & = \mathsf{true} \\
(\mathsf{q}2) & \mathsf{q}(x,\mathsf{s}(y)) & = \mathsf{true} \quad \longleftarrow \quad \mathsf{q}(x,y){=}\mathsf{true},\ \ \mathsf{p}(x){=}\mathsf{true}
\end{array}
$$

Example 10 *(continuing examples 1 and 9)*

The following are formulas w.r.t. the signature of Example 1. Moreover, they are all type-C inductively valid (cf. Definition 65) w.r.t. R_9*. Let* $x, y, z \in \mathrm{V}_{\mathrm{CONS},\mathsf{nat}}$*;* $k, l \in \mathrm{V}_{\mathrm{CONS},\mathsf{list}}$*; and* $b \in \mathrm{V}_{\mathrm{CONS},\mathsf{bool}}$*. The first column indicates the number of the example where the formula is proved, the second a name of the formula which will be used in what follows.*

	(Tertium non datur)	$b = \mathsf{true},\ \ b = \mathsf{false}$
122	*(*$0 \neq \mathsf{s}$*)*	$0 \neq \mathsf{s}(x)$
123	*(Inject* s*)*	$\mathsf{s}(x) \neq \mathsf{s}(y),\ \ x = y$
125	*(Irrefl* s*)*	$x \neq \mathsf{s}(x)$
	*(*Def dl*)*	Def $\mathsf{dl}(x, l)$
	*(*Def rc*)*	Def $\mathsf{rc}(x, l)$
	*(*Def br*)*	Def $\mathsf{br}(k, l)$
	*(*Def $+$*)*	Def $(x + y)$
121	*(*Def $-$*)*	Def $(x - y),\ \ \mathsf{leq}(y, x) \neq \mathsf{true}$
	*(*Def ack*)*	Def $\mathsf{ack}(x, y)$
	*(*Def switch*)*	Def $\mathsf{switch}(x)$
	*(*Def swatch*)*	Def $\mathsf{swatch}(\mathsf{s}(x))$
	*(*Def leq*)*	Def $\mathsf{leq}(x, y)$
	*(*Def less*)*	Def $\mathsf{less}(x, y)$
	*(*Def mbp*)*	Def $\mathsf{mbp}(x, l)$
	(Del dl*)*	$\mathsf{dl}(x, l) = l,\ \ \mathsf{mbp}(x, l) = \mathsf{true}$
	(Del rc*)*	$\mathsf{rc}(x, l) = l,\ \ \mathsf{mbp}(x, l) = \mathsf{true}$
	(Del dl mbp*)*	$\mathsf{mbp}(x, \mathsf{dl}(y, l)) = \mathsf{mbp}(x, l),\ \ x = y$
	(Del rc mbp*)*	$\mathsf{mbp}(x, \mathsf{rc}(y, l)) = \mathsf{mbp}(x, l)$
27	*(Rip* cons br*)*	$\mathsf{br}(k, \mathsf{cons}(x, l)) = \mathsf{cons}(x, \mathsf{br}(k, l)),\ \ \mathsf{mbp}(x, l) = \mathsf{true}$
	(Del 0 $+$*)*	$0 + x = x$
	(Rip s $+$*)*	$\mathsf{s}(x) + y = \mathsf{s}(x + y)$
	(Comm $+$*)*	$x + y = y + x$
	(Pos ack*)*	$\mathsf{less}(0, \mathsf{ack}(x, y)) = \mathsf{true}$
114	*(Irrefl 2* ack*)*	$\mathsf{less}(y, \mathsf{ack}(x, y)) = \mathsf{true}$
	(Del switch switch*)*	$\mathsf{switch}(\mathsf{switch}(x)) = x$
	(Del swatch swatch*)*	$\mathsf{swatch}(\mathsf{swatch}(\mathsf{s}(x))) = \mathsf{s}(x)$
105	*(Del* swatch^3*)*	$\overline{\text{Def}\ \mathsf{swatch}(0)},\ \ \mathsf{swatch}(\mathsf{swatch}(\mathsf{swatch}(x))) = \mathsf{swatch}(x)$
	(True leq*)*	$\mathsf{leq}(x, x) = \mathsf{true}$
	(False leq*)*	$\mathsf{leq}(\mathsf{s}(x), x) = \mathsf{false}$
	(True less*)*	$\mathsf{less}(x, \mathsf{s}(x)) = \mathsf{true}$
	(False less*)*	$\mathsf{less}(x, x) = \mathsf{false}$
	(Strict Trans less*)*	$\mathsf{less}(\mathsf{s}(x), z) = \mathsf{true},\ \ \mathsf{less}(x, y) \neq \mathsf{true},\ \ \mathsf{less}(y, z) \neq \mathsf{true}$
129	*(True* p*)*	$\mathsf{p}(x) = \mathsf{true}$

2.8 Semantics of Rules and Formulas

Instead of giving semantics for sets of rules directly, we define semantics of formulas and then say which formula a rule denotes. In the following definition we define the semantics of formulas in a partial way that is sufficient for the semantics of formulas without '<'-literals and of rules. The definition will be completed in Definition 72. As usual, the fixed meaning of '=' is the equality in a sig/cons-algebra $\mathcal{A}$. 'Def', however, is the "definedness" predicate which states that the evaluation of its argument belongs (with invariant sort) to the constructor sub-universe of $\mathcal{A}$ which contains the set of evaluation values of constructor ground terms and which is intended to supply a domain for (possibly partially) specifying functions on it.

Definition 11 (Validity of Formulas in sig/cons**-algebras; Part I)**

Let $\mathrm{X} \subseteq \mathrm{V}$; $\mathcal{A}$ be a sig/cons-algebra; $\kappa \in \mathcal{SUB}(\mathrm{X}, \mathcal{A})$; and R a CRS over sig/cons/V.

An atom $(u{=}v) \in \mathcal{AT}(\mathrm{sig}, \mathrm{V})$ is *true w.r.t.* $\mathcal{A}_\kappa$ if $\mathcal{V}(u, v) \subseteq \mathrm{X}$ and $\mathcal{A}_\kappa(u) = \mathcal{A}_\kappa(v)$; and an atom $(\mathrm{Def}\, u) \in \mathcal{AT}(\mathrm{sig}, \mathrm{V})$ (with $u \in \mathcal{T}_{\mathrm{SIG},s}$; $s \in \mathbb{S}$) is *true w.r.t.* $\mathcal{A}_\kappa$ if $\mathcal{V}(u) \subseteq \mathrm{X}$ and $\mathcal{A}_\kappa(u) \in \mathcal{A}_{\mathrm{CONS},s}$.
An atom $(u{=}v) \in \mathcal{AT}(\mathrm{sig}, \mathrm{V})$ is *false w.r.t.* $\mathcal{A}_\kappa$ if $\mathcal{V}(u, v) \subseteq \mathrm{X}$ and $\mathcal{A}_\kappa(u) \neq \mathcal{A}_\kappa(v)$; and an atom $(\mathrm{Def}\, u) \in \mathcal{AT}(\mathrm{sig}, \mathrm{V})$ (with $u \in \mathcal{T}_{\mathrm{SIG},s}$; $s \in \mathbb{S}$) is *false w.r.t.* $\mathcal{A}_\kappa$ if $\mathcal{V}(u) \subseteq \mathrm{X}$ and $\mathcal{A}_\kappa(u) \notin \mathcal{A}_{\mathrm{CONS},s}$.

A literal $\overline{A}$ (with $A \in \mathcal{GENAT}(\mathrm{sig}, \mathrm{V})$) is *true w.r.t.* $\mathcal{A}_\kappa$ if A is false w.r.t. $\mathcal{A}_\kappa$.
A literal $\overline{A}$ (with $A \in \mathcal{GENAT}(\mathrm{sig}, \mathrm{V})$) is *false w.r.t.* $\mathcal{A}_\kappa$ if A is true w.r.t. $\mathcal{A}_\kappa$.

A formula $\Gamma \in \mathrm{Form}(\mathrm{sig}, \mathrm{V})$ is *true w.r.t.* $\mathcal{A}_\kappa$ if $\mathcal{V}(\Gamma) \subseteq \mathrm{X}$ and there is some literal λ in Γ that is true w.r.t. $\mathcal{A}_\kappa$.
A formula $\Gamma \in \mathrm{Form}(\mathrm{sig}, \mathrm{V})$ is *false w.r.t.* $\mathcal{A}_\kappa$ if all literals λ in Γ are false w.r.t. $\mathcal{A}_\kappa$.

$\Gamma \in \mathrm{Form}(\mathrm{sig}, \mathrm{V})$ is *valid in* $\mathcal{A}$ if $\forall \kappa \in \mathcal{SUB}(\mathcal{V}(\Gamma), \mathcal{A})$: ($\Gamma$ is true w.r.t. $\mathcal{A}_\kappa$).

Note that the following obvious lemma does not hold for a generalized literal λ.

Lemma 12 (Substitution-Lemma for Non-Generalized Literals)
Let $\mathcal{A}$ *be a* sig/cons*-algebra;* $\lambda \in \mathcal{LIT}(\mathrm{sig}, \mathrm{V})$; $\mu \in \mathcal{GENSUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}))$; *and* $\kappa \in \mathcal{SUB}(\mathrm{Y}, \mathcal{A})$ *for some* Y *with* $\mathcal{V}(\mathrm{ran}(\mu)) \subseteq \mathrm{Y}$.
Now the following two statements are logically equivalent:

- $\lambda\mu$ *is true w.r.t.* $\mathcal{A}_\kappa$.
- λ *is true w.r.t.* $\mathcal{A}_{\mu\mathcal{A}_\kappa}$.

Moreover, if λ *rewrites to* λ' *with the rule* $l{=}r$ *and if* $(l{=}r)\mu$ *is true w.r.t.* $\mathcal{A}_\kappa$, *the following is also logically equivalent:*

- $\lambda'\mu$ *is true w.r.t.* $\mathcal{A}_\kappa$.

As we have negative equations in the conditions of our rules, in general we cannot hope to get a minimum model because we can express things like "$\mathsf{a}{=}\mathsf{b} \vee \mathsf{b}{=}\mathsf{c}$", which has the incomparable minimal models "$\mathsf{a}{=}\mathsf{b}{\neq}\mathsf{c}$" and "$\mathsf{a}{\neq}\mathsf{b}{=}\mathsf{c}$". What we will get instead is a model which is the (up to isomorphism) uniquely determined minimum of all sig-term-generated models that are minimal w.r.t. the identification of constructor ground terms (cf. Corollary 26). For formally expressing these minimality-properties, we need the following definition.

Definition 13 (Model)

When talking about semantics of rules we identify the rule

$$l{=}r \longleftarrow \lambda_0 \ldots \lambda_{n-1}$$

with the formula

$$(l{=}r)\ \overline{\lambda_0}\ \ldots\ \overline{\lambda_{n-1}}.$$

A sig/cons-algebra $\mathcal{A}$ is a sig/cons-*model* of a CRS R over sig/cons/V if all rules in R (considered as formulas) are valid in $\mathcal{A}$.

A sig/cons-algebra $\mathcal{A}$ will be called a *minimum model* (or else a *constructor-minimum model*) of R if $\mathcal{A}$ is a $\lesssim_{\mathrm{H}}$-minimum (or else $\lesssim_{\mathrm{CONS}}$-minimum) of the class of all sig/cons-models of R.

Similarly, $\mathcal{A}$ is a *minimal model* (or else a *constructor-minimal model*) of R if $\mathcal{A}$ is a sig/cons-model of R and there is no sig/cons-model $\mathcal{B}$ of R with $\mathcal{B} <_{\mathrm{H}} \mathcal{A}$ (or else $\mathcal{B} <_{\mathrm{CONS}} \mathcal{A}$).

2.9 The Reduction Relation

In this section we are going to define a reduction relation $\longrightarrow$ which is convenient for the semantics defined in section 2.8. The overall idea is to reduce a left-hand side of a rule to its right-hand side only if the condition of this rule can somehow be shown valid by means of the same reduction relation again. The reader who is not interested in the details of our reduction relation should read only Definition 24 and Theorem 25 of this section.

When we are now going to define our reduction relation, please keep in mind that we intend to require it to be confluent in the sequel, whereas we do not require confluence for the definition because we cannot prove confluence criteria if the non-confluent case is undefined. Therefore, we have to be explicit about how we test the condition literals — even if this testing is not straightforward when confluence is not provided. Our "operational" semantics for testing condition literals is the following: "$u{=}v$" is fulfilled if u, v have reducts $\hat{u}, \hat{v}$, resp., which are syntactically equal. "Def u" is fulfilled if u has a constructor ground reduct, which means that our reduction relation depends on the constructor sub-signature 'cons' beyond the signature 'sig' — just as our notion of "sig/cons-model" does. Finally, "$u{\neq}v$" is fulfilled if u, v have constructor ground reducts $\hat{u}, \hat{v}$, resp., which are not joinable. Thus, two terms in a condition literal are operationally equal if they are joinable, whereas they are unequal if they are not joinable after some reduction to constructor ground terms. The non-joinability alone of two terms is not sufficient for regarding them as unequal because we are never sure about the inequality of "undefined" terms. Note that our operational logic is four-valued, i.e. '$=$' and '$\neq$' can independently be fulfilled or not. In case of confluence, however, it is impossible that both "$u{=}v$" and "$u{\neq}v$" are fulfilled simultaneously; in case of free or confluent constructors, such a simultaneous fulfilledness occurs only if we have something like an ambiguous function definition.

Definition 14 (Fulfilledness)
A list $D \in (\mathcal{CONDLIT}(\mathrm{sig}, \mathrm{V}))^*$ of condition literals is said to be *fulfilled w.r.t. some relation* $\longrightarrow$ if

$$\forall u, v \in \mathcal{T}\colon \left(\begin{array}{rlll} & (\ ((\ u{=}v\ ) \text{ in } D) & \Rightarrow & u{\downarrow}v\ ) \\ \wedge & (\ ((\mathrm{Def}\, u) \text{ in } D) & \Rightarrow \quad \exists \hat{u} \in \mathcal{GT}(\mathrm{cons})\colon & u \xrightarrow{*} \hat{u}\ ) \\ \wedge & (\ ((\ u{\neq}v\ ) \text{ in } D) & \Rightarrow \exists \hat{u}, \hat{v} \in \mathcal{GT}(\mathrm{cons})\colon & u \xrightarrow{*} \hat{u}\, {\updownarrow}\, \hat{v} \xleftarrow{*} v\ ) \end{array} \right).$$

Usually one gets a minimal reduction relation by taking the closure over a finitary generating relation. This is not possible here, because we have a negative condition ($\hat{u}{\updownarrow}\hat{v}$). Since our constructor rules have "Horn"-form and contain no non-constructor function symbols, however, this negative condition does not influence the reduction of constructor terms; and (in Definition 14) '$\updownarrow$' is applied to constructor (ground) terms only. Thus, to avoid a non-monotonic behaviour due to our negative condition, we define our intended minimal reduction relation $\longrightarrow_{\mathrm{R,X}}$ via a double closure: First we define $\longrightarrow_{\mathrm{R,X},\omega}$ by using the constructor rules only. Then we define $\longrightarrow_{\mathrm{R,X},\omega+\omega}$ via a second closure including all rules, knowing the constructor reduction to remain unchanged.

Definition 15 ($\longrightarrow_{\mathrm{R,X}}$)

Let R be a CRS over sig/cons/V. Let X$\subseteq$V. For $\beta \preceq \omega{+}\omega$ the reduction relations $\longrightarrow_{\mathrm{R,X},\beta}$ on $\mathcal{T}(\mathrm{sig},\mathrm{X})$ are inductively defined as follows:[10]

For a limit ordinal $\alpha \in \{0, \omega, \omega{+}\omega\}$: $\longrightarrow_{\mathrm{R,X},\alpha} := \bigcup_{\beta \prec \alpha} \longrightarrow_{\mathrm{R,X},\beta}$.

For for the non-limit ordinals $\beta{+}1$ with $\beta \prec \omega{+}\omega$ and for $s,t \in \mathcal{T}(\mathrm{sig},\mathrm{X})$:

$$s \longrightarrow_{\mathrm{R,X},\beta+1} t \quad \text{if} \quad \exists \left\langle \begin{array}{l} p \in \mathcal{POS}(s) \\ ((l,r),C) \in \mathrm{R} \\ \sigma \in \mathcal{SUB}(\mathrm{V},\mathcal{T}(\mathrm{X})) \end{array} \right\rangle : \left(\begin{array}{ll} & s/p = l\sigma \\ \wedge & t = s[\,p \leftarrow r\sigma\,] \\ \wedge & C\sigma \text{ is fulfilled w.r.t. } \longrightarrow_{\mathrm{R,X},\beta} \\ \wedge & \left(\beta \prec \omega \;\Rightarrow\; l \in \mathcal{T}(\mathrm{cons}, \mathrm{V_{SIG}} \uplus \mathrm{V_{CONS}}) \right) \end{array} \right).$$

$\longrightarrow_{\mathrm{R,X}} := \longrightarrow_{\mathrm{R,X},\omega+\omega}$.

We will drop "R, X" in $\longrightarrow_{\mathrm{R,X}}$ and $\longrightarrow_{\mathrm{R,X},\beta}$ etc. when referring to some fixed R, X. Instantiations of X which are important in theory and practice are at least $\emptyset$, $\mathrm{V_{SIG}}$, and V. We have introduced the parameter X since it is more convenient than triplicating statements about properties (e.g. confluence) for " $\mathrm{X} = \emptyset$ " (e.g. ground confluence), " $\mathrm{X} = \mathrm{V_{SIG}}$ ", and " $\mathrm{X} = \mathrm{V}$ ".

Lemma 16

$\longrightarrow_{\mathrm{R,X},\omega}$ is the minimum (w.r.t. set-inclusion) of all relations $\rightsquigarrow$ on $\mathcal{T}$ satisfying for all $s,t \in \mathcal{T}(\mathrm{sig},\mathrm{X})$:

$$s \rightsquigarrow t \quad \textit{if} \quad \exists \left\langle \begin{array}{l} p \in \mathcal{POS}(s) \\ ((l,r),C) \in \mathrm{R} \\ \sigma \in \mathcal{SUB}(\mathrm{V},\mathcal{T}(\mathrm{X})) \end{array} \right\rangle : \left(\begin{array}{ll} & s/p = l\sigma \\ \wedge & t = s[\,p \leftarrow r\sigma\,] \\ \wedge & C\sigma \textit{ is fulfilled w.r.t. } \rightsquigarrow \\ \wedge & l \in \mathcal{T}(\mathrm{cons}, \mathrm{V_{SIG}} \uplus \mathrm{V_{CONS}}) \end{array} \right).$$

Lemma 17 *Let $S_{\mathrm{R,X}}$ be the set of all relations $\rightsquigarrow$ on $\mathcal{T}$ satisfying*

1. $(\rightsquigarrow \cap (\mathcal{GT}(\mathrm{cons}) \times \mathcal{T})) \subseteq \longrightarrow_{\mathrm{R,X},\omega}$ *as well as*
2. *for all $s,t \in \mathcal{T}(\mathrm{sig},\mathrm{X})$:*

$$s \rightsquigarrow t \quad \textit{if} \quad \exists \left\langle \begin{array}{l} p \in \mathcal{POS}(s) \\ ((l,r),C) \in \mathrm{R} \\ \sigma \in \mathcal{SUB}(\mathrm{V},\mathcal{T}(\mathrm{X})) \end{array} \right\rangle : \left(\begin{array}{ll} & s/p = l\sigma \\ \wedge & t = s[\,p \leftarrow r\sigma\,] \\ \wedge & C\sigma \textit{ is fulfilled w.r.t. } \rightsquigarrow \end{array} \right).$$

Now $\longrightarrow_{\mathrm{R,X}}$ is the minimum (w.r.t. set-inclusion) in $S_{\mathrm{R,X}}$, and $S_{\mathrm{R,X}}$ is closed under nonempty intersection.

[10] Note that compared to Wirth & Gramlich (1994a) etc. we have removed some redundant " or $s \longrightarrow_{\mathrm{R,X},\omega} t$ " in the definition of $\longrightarrow_{\mathrm{R,X},\beta+1}$ for the case of $\omega \preceq \beta$ because this makes the definition and the proofs shorter.

Corollary 18 (Monotonicity of $\longrightarrow$ w.r.t. Replacement)
$\longrightarrow_{\mathrm{R},\mathrm{X},\beta}$ *(for* $\beta \preceq \omega{+}\omega$*) and* $\longrightarrow_{\mathrm{R},\mathrm{X}}$ *are* $\mathcal{T}(\mathrm{sig},\mathrm{X})$*-monotonic as well as* $\xrightarrow{*}_{\mathrm{R},\mathrm{X}}[\mathrm{T}]$*-monotonic for each* $\mathrm{T} \subseteq \mathcal{T}(\mathrm{sig},\mathrm{X})$.

Corollary 19 (Stability of $\longrightarrow$)
$\longrightarrow_{\mathrm{R},\mathrm{X},\beta}$ *(for* $\beta \preceq \omega{+}\omega$*),* $\longrightarrow_{\mathrm{R},\mathrm{X}}$*, and their respective fulfilledness-predicates are* X*-stable.*

Lemma 20 *For* $\mathrm{Y} \subseteq \mathrm{X} \subseteq \mathrm{V}$*:*
$$\forall n \in \mathbb{N}\colon\ \forall s \in \mathcal{T}(\mathrm{cons},\mathrm{Y})\colon\ \forall t\colon\ \Big(\ s \xrightarrow{n}_{\mathrm{R},\mathrm{X}} t\ \Rightarrow\ (s \xrightarrow{n}_{\mathrm{R},\mathrm{X},\omega} t \in \mathcal{T}(\mathrm{cons},\mathrm{Y}))\ \Big)$$

Lemma 21 $\quad \downarrow \cap\ (\,\mathcal{T}(\mathrm{cons},\mathrm{V}_{\mathrm{SIG}} \uplus \mathrm{V}_{\mathrm{CONS}}) \times \mathcal{T}(\mathrm{cons},\mathrm{V}_{\mathrm{SIG}} \uplus \mathrm{V}_{\mathrm{CONS}})\,) \quad \subseteq \quad \downarrow_{\omega}$

Lemma 22 (Monotonicity of $\longrightarrow_{\beta}$ and of Fulfilledness w.r.t. $\longrightarrow_{\beta}$ in β)
For $\beta \preceq \gamma \preceq \omega{+}\omega$*:* $\longrightarrow_{\beta} \subseteq \longrightarrow_{\gamma} \subseteq \longrightarrow$*; and if* C *is fulfilled w.r.t.* $\longrightarrow_{\beta}$ *and* $\left(\begin{array}{ll} & \omega \preceq \beta \\ \vee & \forall u,v\colon ((u{\neq}v) \text{ is not in } C) \end{array}\right)$*, then* C *is fulfilled w.r.t.* $\longrightarrow_{\gamma}$ *and w.r.t.* $\longrightarrow$.

Note that monotonicity of fulfilledness is not given in general for $\beta \prec \omega$ and a negative literal which may become invalid during the growth of the reduction relation on constructor terms.

Lemma 23
Let $\mathrm{X} \subseteq \mathrm{Y} \subseteq \mathrm{V}$*. Now:*
For all $\beta \preceq \omega{+}\omega$ *and* $n \in \mathbb{N}$ *with* $n{\neq}0$*:* $\xrightarrow{n}_{\mathrm{R},\mathrm{X},\beta} = \xrightarrow{n}_{\mathrm{R},\mathrm{Y},\beta} \cap\ (\mathcal{T}(\mathrm{sig},\mathrm{X}){\times}\mathcal{T}(\mathrm{sig},\mathrm{X}))$*, and for* $C \in (\mathcal{CONDLIT}(\mathrm{sig},\mathrm{X}))^{*}$*:* C *is fulfilled w.r.t.* $\longrightarrow_{\mathrm{R},\mathrm{X},\beta}$ *iff* C *is fulfilled w.r.t.* $\longrightarrow_{\mathrm{R},\mathrm{Y},\beta}$.
Furthermore, $\mathrm{dom}(\longrightarrow_{\mathrm{R},\mathrm{X}}) = \mathrm{dom}(\longrightarrow_{\mathrm{R},\mathrm{Y}}) \cap \mathcal{T}(\mathrm{sig},\mathrm{X})$.
Finally, if $\longrightarrow_{\mathrm{R},\mathrm{Y}}$ *is confluent, then* $\longrightarrow_{\mathrm{R},\mathrm{X}}$ *is confluent, too.*

While the following theorem in its general form is indeed necessary for establishing appropriate notions of inductive validity (cf. Wirth & al. (1993) and Wirth & Gramlich (1994b)), its meaning is easier to grasp from its corollary below, saying that (for Def-moderate CRSs R with confluent $\longrightarrow_{R,\emptyset}$) the factor algebra $\mathcal{GT}/\overset{*}{\longleftrightarrow}_{R,\emptyset}$ is an (up to isomorphism) uniquely determined sig/cons-model being initial in a class of models which, in our opinion, captures the intuition behind constructor-based specifications. Furthermore, this unique model $\mathcal{GT}/\overset{*}{\longleftrightarrow}_{R,\emptyset}$ can be constructed by means of the congruence induced by our reduction relation. Thus $\mathcal{GT}/\overset{*}{\longleftrightarrow}_{R,\emptyset}$ provides a computational model for positive/negative-conditional specifications in a fashion very similar to the initial model (or abstract data type) for unconditional or positive-conditional specifications.

Definition 24 (Def-**Moderate Conditional Rule Systems** (Def-**MCRS**))
A CRS R is a Def-*moderate conditional rule system* (Def-*MCRS*)[11] if

$$\forall((l,r),C)\in \mathrm{R}\colon\ \forall(u_0{\neq}u_1) \text{ in } C\colon\ \forall i \prec 2\colon \begin{pmatrix} u_i \in \mathcal{T}(\mathrm{cons},\mathrm{V_{CONS}}) \\ \vee \quad (\mathrm{Def}\, u_i) \text{ is in } C \end{pmatrix}.$$

Theorem 25 (Minimal Model being Free in the Constructor-Minimal Models)
Let R *be a* Def-*MCRS over* sig/cons/V. *Let* $\mathrm{X}\subseteq\mathrm{V}$. *Let* K *be the class of all constructor-minimal models of* R. *Let* ι *be given by* $(x\in\mathrm{X})$*:* $x \mapsto \overset{*}{\longleftrightarrow}_{R,X}[\{x\}]$.
Now, if $\longrightarrow_{R,\emptyset}$ *is confluent, then* $\mathcal{T}(\mathrm{X})/\overset{*}{\longleftrightarrow}_{R,X}$ *is free for* K *over* X *w.r.t.* ι.
Furthermore, if we assume $\longrightarrow_{R,X}$ *to be confluent and* $\mathrm{X}\subseteq\mathrm{V_{SIG}}$ *, then:*

(1) $\mathcal{T}(\mathrm{X})/\overset{*}{\longleftrightarrow}_{R,X}$ *is a constructor-minimum model of* R.
(2) $\mathcal{T}(\mathrm{X})/\overset{*}{\longleftrightarrow}_{R,X}$ *is free in* K *over* X *w.r.t.* ι.
(3) $\mathcal{T}(\mathrm{X})/\overset{*}{\longleftrightarrow}_{R,X}$ *is a minimal model of* R.

Corollary 26 *Let* R *be a* Def-*MCRS over* sig/cons/V. *Furthermore, assume* $\longrightarrow_{R,\emptyset}$ *to be confluent. Now:* $\mathcal{GT}/\overset{*}{\longleftrightarrow}_{R,\emptyset}$ *is a minimal model of* R*, initial in the class of all constructor-minimal models of* R*, and the (up to isomorphism) unique* ($\lesssim_{H}$-) *minimum of the* sig*-term-generated constructor-minimal models of* R.

[11] Note that for convenience we have slightly changed Definition 24 compared to that of Wirth & Gramlich (1994a) etc.. Def-MCRSs are only interesting for reduction etc. based on $\longrightarrow_{R,X}$ with $\mathrm{X}\subseteq\mathrm{V_{SIG}}$. In this case, however, it is not necessary to include $(\mathrm{Def}\, u_i)$ into C when $u_i\in\mathcal{T}(\mathrm{cons},\mathrm{V_{CONS}})$, because then each $\mathcal{SUB}(\mathrm{V},\mathcal{T}(\mathrm{X}))$-instance of $(\mathrm{Def}\, u_i)$ will be fulfilled w.r.t. $\longrightarrow_{R,X}$ anyway. Note that the Def-MCRS R_9 of Example 9 was not a Def-MCRS according to our old definition.

3 The Abstract Inference System

In this section we present a revised version of Wirth & Becker (1995) because it is essential for developing and understanding the key properties of our inference system. Due to the high level of abstraction, we now begin with a detailed concrete example in order to refer to it in what follows. We do not expect the reader to understand or read this example in detail. Nevertheless, it may help to understand the justification for the way we construct our abstract inference system.

Example 27 *(continuing examples 1, 9, 10)*
Let $x, y, z \in \mathrm{V}_{\mathrm{CONS,nat}}$ *and* $h, k, l \in \mathrm{V}_{\mathrm{CONS,list}}$. *Suppose that the defining rules are given as in Example 9 and that we want to show the following theorem (Rip* cons br*) from Example 10, where " , " denotes "logical or" and " ; " separates the formula from its weight that controls the application of induction hypotheses:*

(27.1) $\mathsf{br}(k, \mathsf{cons}(x, l)) = \mathsf{cons}(x, \mathsf{br}(k, l)),\ \ \mathsf{mbp}(x, l) = \mathsf{true}\ \ ;\ \ \mathsf{w}_{27}(k, x, l)$

The goal (27.1) says that x *occurs in* l *or the wave-front* $\mathsf{cons}(x, \ldots)$ *can be rippled out. Applying a covering set of substitutions to (27.1) (using the Substitution Add rule of section 7.1), we get a base case (27.1.1) for* $\{k \mapsto \mathsf{nil}\}$ *and the case (27.1.2) for* $\{k \mapsto \mathsf{cons}(y, k)\}$*:*

(27.1.1) $\mathsf{br}(\mathsf{nil}, \mathsf{cons}(x, l)) = \mathsf{cons}(x, \mathsf{br}(\mathsf{nil}, l)),\ \ \mathsf{mbp}(x, l) = \mathsf{true}\ \ ;\ \ \mathsf{w}_{27}(\mathsf{nil}, x, l)$

(27.1.2) $\mathsf{br}(\mathsf{cons}(y, k), \mathsf{cons}(x, l)) = \mathsf{cons}(x, \mathsf{br}(\mathsf{cons}(y, k), l)),$
$\mathsf{mbp}(x, l) = \mathsf{true}\ \ ;\ \ \mathsf{w}_{27}(\mathsf{cons}(y, k), x, l)$

*A rewrite step with the defining rule (*br*1) of Example 9 in the left-hand side of the first literal (using the Lemma Rewrite rule of section 5.5) transforms (27.1.1) into*

(27.1.1.1) $\mathsf{cons}(x, l) = \mathsf{cons}(x, \mathsf{br}(\mathsf{nil}, l)),\ \ \mathsf{mbp}(x, l) = \mathsf{true}\ \ ;\ \ \mathsf{w}_{27}(\mathsf{nil}, x, l)$

*A second Lemma Rewrite with the defining rule (*br*1), now in the right-hand side of the first literal, transforms (27.1.1.1) into*

(27.1.1.1.1) $\mathsf{cons}(x, l) = \mathsf{cons}(x, l),\ \ \mathsf{mbp}(x, l) = \mathsf{true}\ \ ;\ \ \mathsf{w}_{27}(\mathsf{nil}, x, l)$

This goal can be removed (using the $=$*-Decompose rule of section 5.2) because the first literal is tautological.*

*Now we return to the goal (27.1.2) left open above. Lemma Rewrite with (*br*2) in the left-hand side of the first literal transforms (27.1.2) into*

(27.1.2.1) $\mathsf{br}(k, \mathsf{rc}(y, \mathsf{cons}(x, l))) = \mathsf{cons}(x, \mathsf{br}(\mathsf{cons}(y, k), l)),$
$\mathsf{mbp}(x, l) = \mathsf{true}\ \ ;\ \ \mathsf{w}_{27}(\mathsf{cons}(y, k), x, l)$

*A second Lemma Rewrite with (*br*2), now in the right-hand side of the first literal transforms (27.1.2.1) into*

(27.1.2.1.1) $\mathsf{br}(k, \mathsf{rc}(y, \mathsf{cons}(x, l))) = \mathsf{cons}(x, \mathsf{br}(k, \mathsf{rc}(y, l))),$
$\mathsf{mbp}(x, l) = \mathsf{true}\ \ ;\ \ \mathsf{w}_{27}(\mathsf{cons}(y, k), x, l)$

*Lemma Rewrite with (*rc*3) in the left-hand side of the first equation transforms (27.1.2.1.1) into the following two goals. The first goal asks us to show that the condition* $x \neq y$ *of (*rc*3) is satisfied. In the second we have carried out the rewrite step.*

(27.1.2.1.1.1) $x \neq y,\ \ \mathsf{br}(k, \mathsf{rc}(y, \mathsf{cons}(x, l))) = \mathsf{cons}(x, \mathsf{br}(k, \mathsf{rc}(y, l))),$
$\mathsf{mbp}(x, l) = \mathsf{true}\ \ ;\ \ \mathsf{w}_{27}(\mathsf{cons}(y, k), x, l)$

(27.1.2.1.1.2) $x = y,\ \ \mathsf{br}(k, \mathsf{cons}(x, \mathsf{rc}(y, l))) = \mathsf{cons}(x, \mathsf{br}(k, \mathsf{rc}(y, l))),$
$\mathsf{mbp}(x, l) = \mathsf{true}\ \ ;\ \ \mathsf{w}_{27}(\mathsf{cons}(y, k), x, l)$

*Lemma Rewrite with (*rc*2) in the left-hand side of the second equation transforms (27.1.2.1.1.1) into:*

(27.1.2.1.1.1.1) $x \neq y,\ \mathsf{br}(k,\mathsf{cons}(x,\mathsf{dl}(y,l))) = \mathsf{cons}(x,\mathsf{br}(k,\mathsf{rc}(y,l))),$
$\mathsf{mbp}(x,l) = \mathsf{true}\ ;\ \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l)$

Now we can use the literal $x \neq y$ *for a rewrite step in the left-hand side of the second literal of (27.1.2.1.1.1.1) (using the Constant Rewrite rule of section 5.4) to yield*

(27.1.2.1^5*)* $x \neq y,\ \mathsf{br}(k,\mathsf{cons}(x,\mathsf{dl}(x,l))) = \mathsf{cons}(x,\mathsf{br}(k,\mathsf{rc}(y,l))),$
$\mathsf{mbp}(x,l) = \mathsf{true}\ ;\ \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l)$

A Lemma Rewrite in the first literal with (Del dl*) from Example 10 yields*

(27.1.2.1^6*)* $x \neq y,\ \mathsf{br}(k,\mathsf{cons}(x,l)) = \mathsf{cons}(x,\mathsf{br}(k,\mathsf{rc}(y,l))),$
$\mathsf{mbp}(x,l) = \mathsf{true}\ ;\ \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l)$

Another Constant Rewrite of the same kind yields

(27.1.2.1^7*)* $x \neq y,\ \mathsf{br}(k,\mathsf{cons}(x,l)) = \mathsf{cons}(x,\mathsf{br}(k,\mathsf{rc}(x,l))),$
$\mathsf{mbp}(x,l) = \mathsf{true}\ ;\ \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l)$

in order that a Lemma Rewrite in the second literal with (Del rc*) from Example 10 can transform this into*

(27.1.2.1^8*)* $x \neq y,\ \mathsf{br}(k,\mathsf{cons}(x,l)) = \mathsf{cons}(x,\mathsf{br}(k,l)),$
$\mathsf{mbp}(x,l) = \mathsf{true}\ ;\ \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l)$

Now we can apply (27.1) as an induction hypothesis using the Hypothesis Apply rule of section 7.2. Since (27.1) subsumes (27.1.2.1^8*) we get only the following ordering condition for the hypothesis application:*

(27.1.2.1^9*)* $\mathsf{w}_{27}(k,x,l) < \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l),$
$x \neq y,\ \mathsf{br}(k,\mathsf{cons}(x,l)) = \mathsf{cons}(x,\mathsf{br}(k,l)),$
$\mathsf{mbp}(x,l) = \mathsf{true}\ ;\ \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l)$

Now we go on with the goal (27.1.2.1.1.2) left open above. Here we can also apply (27.1) as an induction hypothesis, matching the first literal of (27.1) to the second literal of (27.1.2.1.1.2), but this is a little more difficult. Since we have to instantiate the constructor variable l *of (27.1) with the non-constructor term* $\mathsf{rc}(y,l)$ *we have to show that this term is defined:*

(27.1.2.1.1.2.1) $\mathrm{Def}\,\mathsf{rc}(y,l),$
$x = y,\ \mathsf{br}(k,\mathsf{cons}(x,\mathsf{rc}(y,l))) = \mathsf{cons}(x,\mathsf{br}(k,\mathsf{rc}(y,l))),$
$\mathsf{mbp}(x,l) = \mathsf{true}\ ;\ \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l)$

Moreover, since the instantiated literal $\mathsf{mbp}(x,\mathsf{rc}(y,l)) = \mathsf{true}$ *of (27.1) is not contained in (27.1.2.1.1.2) we have to show its complement:*

(27.1.2.1.1.2.2) $\mathsf{mbp}(x,\mathsf{rc}(y,l)) \neq \mathsf{true},$
$x = y,\ \mathsf{br}(k,\mathsf{cons}(x,\mathsf{rc}(y,l))) = \mathsf{cons}(x,\mathsf{br}(k,\mathsf{rc}(y,l))),$
$\mathsf{mbp}(x,l) = \mathsf{true}\ ;\ \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l)$

Finally we have to show that the weight of the instantiated hypothesis (27.1) is smaller than the weight of the goal (27.1.2.1.1.2):

(27.1.2.1.1.2.3) $\mathsf{w}_{27}(k,x,h) < \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l),\ \ h \neq \mathsf{rc}(y,l),$
$x = y,\ \mathsf{br}(k,\mathsf{cons}(x,\mathsf{rc}(y,l))) = \mathsf{cons}(x,\mathsf{br}(k,\mathsf{rc}(y,l))),$
$\mathsf{mbp}(x,l) = \mathsf{true}\ ;\ \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l)$

*The lemma (*Def rc*) from Example 10 subsumes the goal (27.1.2.1.1.2.1), which therefore can be removed by the Lemma Apply rule of section 5.5.*

A Lemma Rewrite with the lemma (Del rc mbp*) from Example 10 applies to the first literal of (27.1.2.1.1.2.2), transforming it into*

(27.1.2.1.1.2.2.1) $\mathsf{mbp}(x,l) \neq \mathsf{true},\ \ x = y,\ \ \mathsf{br}(k,\mathsf{cons}(x,\mathsf{rc}(y,l))) = \mathsf{cons}(x,\mathsf{br}(k,\mathsf{rc}(y,l))),$
$\mathsf{mbp}(x,l) = \mathsf{true}\ ;\ \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l)$

which can then be removed by an $=$*-Decompose of the last literal because it is complementary*

to the first.

Now the proof is completed when we can solve the ordering conditions of (27.1.2.1[9]) and (27.1.2.1.1.2.3), namely

(27.1.2.1[10]) $\mathsf{w}_{27}(k,x,l) < \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l)$

and

(27.1.2.1.1.2.3.1) $\mathsf{w}_{27}(k,x,h) < \mathsf{w}_{27}(\mathsf{cons}(y,k),x,l)$

After defining $\mathsf{w}_{27}(k,x,l) = k$ *this means:*

(27.1.2.1[11]) $k < \mathsf{cons}(y,k)$

and

(27.1.2.1.1.2.3.1.1) $k < \mathsf{cons}(y,k)$

which are satisfied when we choose '<' to be the subterm ordering '$\lhd_{\mathrm{ST}}$'.

3.1 Proof States

Proof states are intended to represent the state of a proof. They record which sub-tasks of a proof have been successfully established, which goals remain to be proved, etc.. Technically, the proof states are the field of our inference relation '$\vdash$'. For *deductive* reasoning we may start with a set 'G' of *goals* which contains the theorems we want to prove, transform them and finally delete them after they have become trivial tautologies. Such an inference relation, starting from the theorems to be proved and reducing them until some termination criterion is satisfied, is called *analytic*, cf. e.g. Bibel & Eder (1993). These theorems have to belong to some set 'Form' which contains the formulas the inference system can treat. If $\vdash$ permits only sound transformation and deletion steps, then from " $G \vdash^* \emptyset$ " (where '$\vdash^*$' denotes the reflexive and transitive closure of $\vdash$) we may conclude that 'G' is valid. For practical reasons, we would like to have a set 'L' of *lemmas* at hand. In this 'L' inference steps can store axioms of the specification, already proved lemmas or the results of any theorem prover called for solving special tasks which might be helpful in our inference process. We can then use 'L' for transformations of 'G' that are only known to be sound relative to 'L'. Thus our proof states should be pairs of sets '(L,G)' such that " $(\emptyset,G) \vdash^* (L,\emptyset)$ " implies validity of 'G' and 'L'. For *inductive* reasoning, we additionally would like to have a set 'H' of *induction hypotheses*. Similar to 'L', the set 'H' may be built up during the inference process and used for transformations of 'G' which are *founded* on 'H' in the sense that they are only known to be sound relative to 'H'. Proof states for inductive theorem proving should be triples of sets '(L,H,G)' such that " $(\emptyset,\emptyset,G) \vdash^* (L,H,\emptyset)$ " implies inductive validity of 'G', 'L', and 'H'. Unlike the lemmas in 'L', however, the hypotheses in 'H' are not known to be valid before the whole induction proof is done (i.e. " $G = \emptyset$ "). Here is a risk of being caught in a cyclic kind of reasoning like: "The goal can be deleted, since it is valid due to the hypothesis, which is valid, if the goal can be deleted, ...", cf. section 1.1. That this cyclic reasoning terminates can be guaranteed by equipping each formula in 'H' or 'G' with a *weight* and allowing a hypothesis to transform a goal only if the weight of the hypothesis is smaller than the weight of the goal w.r.t. a *wellfounded quasi-ordering* '$\lesssim$' which we will call the *induction ordering* in what follows.

Note that we distinguish between the weight of a formula and the actual formula itself: For explicit induction, weights are not needed on the inference system level because each

inductive reasoning cycle is encapsuled in a single inference step which combines induction conclusions with induction hypotheses in step formulas. In implicit induction, however, induction conclusions and hypotheses are not joined in the beginning. Instead, the conclusions are taken as goals and transformed until it becomes obvious which hypotheses will be useful for proving them. At this point, when hypotheses are to be applied to the transformed goals, their weights are needed to transfer ordering information from the original goals that generated the hypotheses to the transformed goals. Roughly speaking, the goals have to store the weights of hypotheses for which they carry the proof work. (This still permits mutual induction.) A possible weight for a formula, which is so natural that there are hardly any other weights in the literature on implicit induction, is (the value of a measure applied to) the formula itself. However, if we require the weight of a goal to be determined by the formula alone, then the chance to transform or delete this goal by means of some fixed hypothesis (which have to be smaller w.r.t. $<$) gets smaller with each transformation of the goal into other goals which are smaller w.r.t. $<$. Such transformation of goals is usually called *simplification*. While simplification of a goal is an important[12] heuristic, the weight of the goal should not change during simplification. This can be stated more generally: For concrete inference rules it is very important that a goal can transmit its weight unchanged to the sub-goals which it is transformed into. This would be generally impossible if the weight of a formula were restricted to be the formula itself. In our approach, therefore, each element 'S' of "$H \cup G$" is some *syntactic construct* from a set 'SynCons'. Besides its formula 'form(S)', 'S' may have some additional contents describing its weight. Conceptually, one can consider each syntactic construct to be a pair made up of a formula expressing some proposition and a weight carrying the ordering information for avoiding non-terminating cycles in the use of inductive arguments. For the description of concrete inference systems within our abstract framework, however, it may be more convenient not to restrict the syntactic constructs to this form because in some of these inference systems the formulas share some structure with the weights: E.g., in Bachmair (1988) the formulas are the weights, and in Becker (1994) the weights restrict the semantics of the formulas by the so-called "reference compatibility". The distinction between formulas and syntactic constructs (i.e. formulas augmented with weights) has the following advantages compared to other inference systems for implicit induction:

No Global Ordering Restriction: Inference steps can be admitted which transform the formula of a goal into another one which is bigger w.r.t. the induction ordering, cf. Gramlich (1989).

High Quality of Ordering Information: The loss of ordering information during simplification of a goal (as described above) is avoided, which (as far as we know) was first described in Wirth (1991), exemplified by a failure of a formal induction proof just caused by this loss of ordering information. There it is also sketched how to store the weight of the goal to avoid this information loss, — an idea which is also found in Becker (1993) and Fraus (1994)[13].

[12] when the induction ordering contains the evaluation ordering of the functional definitions of the specification, cf. Walther (1994)

[13] Note, however, that in Fraus (1994) the weight of a goal is not directly attached to the goal but to each hypothesis in the set of hypotheses that is attached to this goal. Moreover, each time a goal is copied into

Example 28 *(continuing Example 27)*
For the soundness of the induction proof of Example 27 we have to find a wellfounded ordering in which the instance of the induction hypothesis (27.1) via the substitution $\{l \mapsto \mathsf{rc}(y,l)\}$*, i.e. (omitting the weight)*
(28.I) $\mathsf{br}(k,\mathsf{cons}(x,\mathsf{rc}(y,l))) = \mathsf{cons}(x,\mathsf{br}(k,\mathsf{rc}(y,l))),\ \ \mathsf{mbp}(x,\mathsf{rc}(y,l)) = \mathsf{true}$
is (strictly) smaller than the goal (27.1.2.1.1.2) to which it is applied, i.e.
(27.1.2.1.1.2) $x = y,\ \ \mathsf{br}(k,\mathsf{cons}(x,\mathsf{rc}(y,l))) = \mathsf{cons}(x,\mathsf{br}(k,\mathsf{rc}(y,l))),\ \ \mathsf{mbp}(x,l) = \mathsf{true}$
If the weight of a formula is the formula itself (regarded as the multi-set of (its literals considered as the multi-sets of) its terms), this cannot be achieved with a simplification ordering, whereas the evaluation ordering of (the relevant part of) the specification is a sub-relation of a simplification ordering, namely the lexicographic path ordering given by the following precedence on function symbols: $\mathsf{mbp} \succsim \mathsf{true}, \mathsf{false}$; $\mathsf{br} \succsim \mathsf{rc} \succsim \mathsf{dl} \succ \mathsf{cons}$.
The first thing we can do is to use weight pointers to avoid the deterioration of ordering information on the way from (27.1.2) to (27.1.2.1.1.2): Initially we set the weight pointer of (27.1) to (27.1) itself, such that the weight of this formula is this formula itself. The same holds for (27.1.2) because when applying the substitution $\{k \mapsto \mathsf{cons}(y,k)\}$ *the weight is instantiated as well as the formula. During the simplification steps yielding (27.1.2.1), (27.1.2.1.1), and then (27.1.2.1.1.2) the weight remains unchanged, i.e. the weight of the formula (27.1.2.1.1.2) is still the formula (27.1.2), i.e.*
(27.1.2) $\mathsf{br}(\mathsf{cons}(y,k),\mathsf{cons}(x,l)) = \mathsf{cons}(x,\mathsf{br}(\mathsf{cons}(y,k),l)),\ \ \mathsf{mbp}(x,l) = \mathsf{true}$
Now (28.I) is indeed strictly smaller than (27.1.2) in the lexicographic path ordering given by the precedence from above extended with $\mathsf{br} \succ \mathsf{mbp}$.
Thus the high quality of ordering information preserved by separating the formula from its weight when simplifying (27.1.2) into (27.1.2.1.1.2), now permits us to justify the application of (28.I) to (27.1.2.1.1.2) with a simplification ordering.

Focus on Relevant Ordering Information: Some induction proofs are only possible if we do not measure the whole formula (as often is the case when a clause is measured as the multi-set of *all* its literals) but only some sub-formula, subterm or variable of it. Focusing on certain variables is common for human beings (speaking of "induction on variable x", e.g.). While focusing on certain variables (called "*measured variables*" in Boyer & Moore (1979) and Walther (1994)) is standard for the mechanisms usually applied inside the induction rule of explicit induction, a marking concept for focusing in implicit induction was introduced in Wirth (1991). The more general focusing that can be achieved with the syntactic constructs here, permits us not to measure those parts of formulas which (due to unsatisfiable ordering conditions) block the application of useful hypotheses. Thus we can focus on the literals (or even terms, variables) that do get smaller on their way from the induction conclusion to the induction hypothesis. This allows additional applications of induction hypotheses.

---

the set of hypotheses attached to it, the goal gets a new weight for this hypothesis, namely the identity substitution on its variables. When a hypothesis is applied to the goal to that it is attached, then the matching substitution has to be strictly smaller than the weight of the goal for this hypothesis. Note that this weight may be different from the identity substitution then because the substitutions applied to the goal are also applied to the weights stored with its hypotheses.

Example 29 *(continuing Example 28)*
The termination argument in Example 28 leaves room for improvement. The new pair in the precedence (i.e. $\mathsf{br} \succ \mathsf{mbp}$*) is not at all motivated by the evaluation ordering of our specification. After all, our proof of Example 27 is nothing but a structural induction and thus should work out without such a sophisticated ordering. When we have a closer look at the derivation from (27.1) (or (27.1.2)) to (27.1.2.1.1.2), then we notice that the first literal does decrease in the evaluation ordering on the way from our original goal (via the goal the hypothesis is applied to) to the instantiated hypothesis (28.I), but the second literal does not. Thus, if we focus on the first literal by setting the weight to it, then we only have to show that the first literal of (28.I) is smaller than the first literal of (27.1.2) which does not require the unmotivated pair* $\mathsf{br} \succ \mathsf{mbp}$ *in the precedence. Going one step further and setting the weight of (27.1) to the variable* k*, the weight of (27.1.2) and (27.1.2.1.1.2) becomes* $\mathsf{cons}(y, k)$ *which is trivially greater than the weight* k *of (28.I).*

Easier Design of Inference Rules: The design of concrete inference rules (as sub-rules of the abstract inference rules in section 3.4 below) becomes simpler because a transformation of the actual formula does not necessarily include a transformation of its weight (into a smaller one) and is thus not restricted by superfluous ordering conditions.

Example 30 *(continuing Example 29)*
Suppose we want to apply the lemma (Del dl*) of Example 10 in the proof of a formula*
(30.1) $\qquad \Gamma[\mathsf{dl}(x, l)]$
containing $\mathsf{dl}(x, l)$ *as a subterm. A Lemma Rewrite with (Del* dl*) of Example 10 transforms (30.1) into the following two sub-goals:*
(30.1.1) $\qquad \mathsf{mbp}(x, l) \neq \mathsf{true}, \quad \Gamma[\mathsf{dl}(x, l)]$
(30.1.2) $\qquad \mathsf{mbp}(x, l) = \mathsf{true}, \quad \Gamma[l]$
When we restrict the weight of a formula to be determined by the formula itself, then the sub-goals (30.1.1) and (30.1.2) have to be smaller than (or equal to) (30.1) (without focusing on $\Gamma[\mathsf{dl}(x, l)]$*). For (30.1.2) this might be achieved again by an unmotivated extension of the precedence on function symbols (given by the evaluation ordering, cf. Example 28) with* $\mathsf{dl} \succ \mathsf{mbp}$ *(additionally to* $\mathsf{mbp} \succsim \mathsf{true}$ *); for (30.1.1), however, this does not seem to be reasonably possible in general. Thus the design of an inference rule applying the lemma (Del* dl*) in the intended form to (30.1) has to be very difficult without separate weights because the step from (30.1) to (30.1.1) has to be replaced with a very big inference step bridging over all steps following in the proof of (30.1.1) until all branches of this proof have reached a smaller weight, which contradicts our design goal of section 1.3 that "inference rules should have a fine grain". Separating weights from formulas, however, makes the design of such an inference rule very easy: One just sets the weight of (30.1.1) and (30.1.2) to the original weight of formula (30.1).*

All in all, the inference relation $\vdash$ should operate on proof states which are triples '(L, H, G)' of finite sets such that 'L' contains formulas (from 'Form') and 'H' and 'G' contain syntactic constructs (from 'SynCons') whose formulas may be accessed via the function " form: SynCons $\rightarrow$ Form ". While proof states represented by forests of syntactic constructs are more useful in practice, the simpler data structure presented here suffices for the purposes of this paper.

3.2 Counterexamples and Validity

For powerful inductive reasoning we have to be able to restrict the test of whether the weight of a hypothesis is smaller than the weight of a goal (which has to be satisfied for the permission to apply the hypothesis to the goal) to the special case semantically described by their formulas. This can be achieved by considering only such instances of their weights that result from ground substitutions describing invalid instances of their formulas. A syntactic construct augmented with such a substitution providing extra information on the invalidity of its formula is called a *counterexample*. Thus, a syntactic construct whose formula is valid has no counterexamples. We assume the existence of some class 'Info' describing this extra information and require the induction ordering $\lesssim$ to be a wellfounded quasi-ordering not simply on 'SynCons' but actually on 'SynCons$\times$Info'. Furthermore, we require "being a counterexample" to be a well-defined basic property which has to be either true or false for each $(S, I) \in \text{SynCons}\times\text{Info}$. Finally, in order to formally express the relation between counterexamples and our abstract notion of validity, we require for each syntactic construct $S \in$ SynCons that its formula 'form(S)' is valid iff there is no $I \in$ Info such that (S, I) is a counterexample.

Note that our notion of "counterexample" is a semantic one contrary to the notion of "inconsistency proof" used in Bachmair (1988). Generally speaking, an abstract frame inference system that is to be fixed prior to the design of concrete inference rules has to be sufficiently stable and therefore its notions should not rely on our changeable ideas on formal proofs.

Finally note that even with our emphasis on proving valid formulas "positively" (instead of being refutationally complete), the somewhat negative kind of argumentation with counterexamples is handier, somewhat less operationally restricted, and more convenient for defining and proving properties of practically useful inference systems than the less local *formal proofs* used in the positive proving approach of Gramlich (1989) or Reddy (1990).

3.3 Foundedness

In this section we move from counterexamples to an even higher level of abstraction. We use the notion of counterexamples to lift $\lesssim$ from "SynCons$\times$Info" to subsets of 'SynCons' by explaining what we mean by saying that a set H of hypotheses is *founded* on a set G of goals (written $H \curvearrowright G$) or by saying that a set G of goals is *strictly founded* on a set H of hypotheses (written $G \searrow H$ or $H \swarrow G$). Roughly speaking, $H \curvearrowright G$ indicates that the hypotheses are known to be valid if a *final* proof state (i.e. one with an empty set of goals) can be entailed. $H \swarrow G$ indicates that the goals in G can be deleted by the application of smaller hypotheses from H.

Definition 31 (Foundedness) Let $M, H, G \subseteq \text{SynCons}$. Let '$\searrow/\curvearrowright$' be a symbol for a single relation. Now M is said to be *strict/quasi-founded* on (H, G) (denoted by $M \searrow/\curvearrowright (H, G)$) if
$\forall S \in M$: $\forall I \in \text{Info}$:

$$\left(\begin{array}{l} ((S,I) \text{ is a counterexample}) \quad \Rightarrow \\ \\ \exists S' \in H \cup G\text{: } \exists I' \in \text{Info: } \left(\begin{array}{ll} & ((S',I') \text{ is a counterexample}) \\ \wedge & \left(\begin{array}{ll} & \left(\begin{array}{ll} & S' \in H \\ \wedge & (S,I) > (S',I') \end{array} \right) \\ \vee & \left(\begin{array}{ll} & S' \in G \\ \wedge & (S,I) \gtrsim (S',I') \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right).$$

M is said to be *strictly founded* on H (denoted by $M \searrow H$) if $M \searrow/\curvearrowright (H, \emptyset)$.
M is said to be (quasi-) *founded* on G (denoted by $M \curvearrowright G$) if $M \searrow/\curvearrowright (\emptyset, G)$.

Note that (for $S \in \text{SynCons}$) the expressive power of " $\{S\} \searrow/\curvearrowright \ldots$ " is higher than that of " $\{S\} \searrow \ldots$ " and " $\{S\} \curvearrowright \ldots$ " together, since " $\{S\} \searrow H \ \vee \ \{S\} \curvearrowright G$ " implies " $\{S\} \searrow/\curvearrowright (H, G)$ ", but the converse does not hold in general.

Corollary 32 *Let $H \subseteq \text{SynCons}$. Now each of the following seven properties is logically equivalent to validity of* form$[H]$*:*

(1)	$H \curvearrowright \emptyset$	(4)	$\forall G \subseteq \text{SynCons}$: $H \curvearrowright G$
(2)	$H \searrow \emptyset$	(5)	$\forall G \subseteq \text{SynCons}$: $H \searrow G$
(3)	$H \searrow H$ *(due to wellfoundedness of $>$)*	(6)	$\exists G \subseteq \text{SynCons}$: (form$[G]$ *is valid*) $\wedge$ $H \curvearrowright G$)
		(7)	$\exists G \subseteq \text{SynCons}$: (form$[G]$ *is valid*) $\wedge$ $H \searrow G$)

Corollary 33 *Let $H \subseteq G \subseteq \text{SynCons}$. Now: $\emptyset \searrow H \curvearrowright G$.*

Corollary 34 *The following four inclusion-properties hold:*
$\searrow \subseteq \curvearrowright$. $\quad \searrow \circ \curvearrowright \subseteq \searrow$. $\quad \curvearrowright \circ \searrow \subseteq \searrow$. $\quad M \searrow/\curvearrowright (H, G) \Rightarrow M \curvearrowright H \cup G$.

Corollary 35

(1)	$M \searrow/\curvearrowright (H,G)$	$\wedge$	$M' \searrow/\curvearrowright (H',G')$	$\Rightarrow$	$M \cup M' \searrow/\curvearrowright (H \cup H', G \cup G')$
(2)	$M \cup M' \searrow/\curvearrowright (H,G)$			$\Rightarrow$	$M \searrow/\curvearrowright (H \cup H', G \cup G')$
(3)	$G \searrow H$			$\Rightarrow$	$G \searrow H \backslash G$

Note that the last item of the previous as well as the first item of the following corollary rely on the wellfoundedness of $>$.

Corollary 36

(1)	$M \searrow/\curvearrowright (H,G)$	$\wedge$	$H \curvearrowright G \cup M$	$\Rightarrow$	$M \curvearrowright G$
(2)	$M \searrow/\curvearrowright (H,G)$	$\wedge$	$G \searrow/\curvearrowright (H',G')$	$\Rightarrow$	$M \searrow/\curvearrowright (H \cup H', G')$

Corollary 37 $\curvearrowright$ *is a quasi-ordering.*

Corollary 38
$\searrow$ *is a transitive relation, which is neither irreflexive nor generally reflexive.*
Let $\forall i \in \mathbb{N}$: $H_i, G_i \subseteq \text{SynCons}$. *Then (by the wellfoundedness of $>$)* $\forall i \in \mathbb{N}$: $H_i \searrow H_{i+1}$ *implies that* form$[H_k]$ *has to be valid for all $k \in \mathbb{N}$. More generally,* $\forall i \in \mathbb{N}$: $H_i \searrow/\curvearrowright (H_{i+1}, G_{i+1})$ *implies* $H_k \curvearrowright \bigcup_{j>k} G_j$ *for all $k \in \mathbb{N}$. Moreover, the restriction of $\searrow$ to those $H \subseteq \text{SynCons}$ with invalid* form$[H]$ *is a wellfounded ordering.*

3.4 The Frame Inference System

We now present four abstract inference rules defining $\vdash$. Thus, in this and the following three sections, $\vdash$ will be restricted to applications of one of the four following inference rules.

In what follows, let $\Gamma \in$ Form; $S \in$ SynCons; L, L' be finite subsets of 'Form'; and H, H', G, G', and M be finite subsets of 'SynCons':

Expansion: $$\frac{(L \quad , H \quad , G \quad)}{(L \quad , H \quad , G \cup \{ S \} \quad)}$$

Hypothesizing: $$\frac{(L \quad , H \quad , G \quad)}{(L \quad , H \cup \{ S \} \quad , G \quad)}$$
if L is invalid or $\{S\} \curvearrowright H \cup G$

Acquisition: $$\frac{(L \quad , H \quad , G \quad)}{(L \cup \{\Gamma\} \quad , H \quad , G \quad)}$$
if L is invalid or Γ is valid.

Deletion: $$\frac{(L \quad , H \quad , G \cup \{ S \} \quad)}{(L \quad , H \quad , G \quad)}$$
if L is invalid or $\{S\} \searrow\!/\!\curvearrowright (H, G)$

The *Expansion rule* has two typical applications. The first one introduces sub-goals for a goal that is to be deleted, cf. the Transformation rule below. The second one is the very difficult task of introducing new conjectures that are needed for the whole induction proof to work.

The *Hypothesizing rule* makes a new hypothesis S available for the proof. Since forward reasoning on hypotheses is hardly required, it can usually be restricted to the following sub-rule (cf. Corollary 33) which just stores the goals in the set of hypotheses. This storing is indeed necessary because these goals usually have been transformed before the hypotheses derived from them become useful for inductive reasoning.

Memorizing: $$\frac{(L \quad , H \quad , G \cup \{ S \} \quad)}{(L \quad , H \cup \{ S \} \quad , G \cup \{ S \} \quad)}$$

The *Acquisition rule* makes a new lemma Γ available for the proof. The rule may be used to include axioms from the specification or formulas proved in other successful runs of the inference system or by any other sound prover which seems appropriate for some special purposes.

The *Deletion rule* permits the deletion of a goal that is strictly founded on some hypotheses. While we cannot go into details here on how to find this out, the Deletion rule especially permits us to remove a goal if its formula is implied by the formula of an instance of a hypothesis and this instance is smaller than the goal in our induction ordering. More frequently, however, is the Deletion rule used in the following combination with several preceding Expansion steps:

Transformation: $\dfrac{(L \ , H \ , G \cup \{ S \} \ )}{(L \ , H \ , G \cup M \ )}$
if L is invalid or $\{S\} \searrow\!/\!\curvearrowright (H, G \cup M)$

The *Transformation rule* replaces a goal S with a (possibly empty) set M of sub-goals whose completeness may rely on hypotheses from H or lemmas from L. It is the most often applied rule of the frame inference system. The intended design of concrete inference systems for specific kinds of validity mainly consists in finding appropriate sub-rules of the Transformation rule. The Transformation rule has the following technical property which is rather useful for constructing new sub-rules of it from already given sub-rules of it. It is a restricted kind of transitivity of the Transformation rule and a corollary of corollaries 37, 35(1), and 36(2):

Corollary 39
If

$$\frac{(L \ , H \ , G \qquad \cup \{ S \} \qquad)}{(L \ , H \ , G \qquad \cup G' \ \cup \ \{ S' \} \ )}$$

and

$$\frac{(L \ , H \ , G \ \cup \ G' \cup \{ S' \} \qquad)}{(L \ , H \ , G \ \cup \ G' \cup M \qquad)}$$

are both applications of the Transformation rule in the sense that L is invalid or

$$\{ S \} \searrow\!/\!\curvearrowright (H, G \cup G' \cup \{ S' \})$$

and

$$\{ S' \} \searrow\!/\!\curvearrowright (H, G \cup G' \cup M) \ ,$$

then, due to L being invalid or

$$\{ S \} \searrow\!/\!\curvearrowright (H, G \cup G' \cup M) \ ,$$

also the following inference is an application of the Transformation rule:

$$\frac{(L \ , H \ , G \qquad \cup \{ S \} \qquad)}{(L \ , H \ , G \qquad \cup G' \ \cup \ M \qquad)}$$

In the following sections we will present two alternative approaches to explaining why the above inference system implements the ideas presented at the beginning of section 3.1. The first is called "analytic" because it is based on an invariance property that holds for an initial proof state and is kept invariant by the analytic inference steps. The second is called "backwards" because it is based solely on an invariance property which holds for a final (i.e. successful) proof state and is kept invariant when one applies the inference rules in backward direction.

3.5 The Analytic Approach

The analytic approach was first formalized in Wirth (1991). In section 3.1 we indicated that if $\vdash$ permits sound steps only, then from " $(\emptyset, \emptyset, G) \vdash^* (L, H, \emptyset)$ " we may conclude that G is valid. This idea is formalized in:

Definition 40 (Soundness of Inference Step)
The inference step " $(L, H, G) \vdash (L', H', G')$ " is called *sound* if validity of 'form$[G']$' implies validity of 'form$[G]$'.

Corollary 41 *If all inference steps in " $(L, H, G) \vdash^* (L', H', \emptyset)$ " are sound, then* 'form$[G]$' *is valid.*

Besides the soundness of an inference *step* described above, it is also useful to know about invariant properties of a proof *state* because they can be used to justify why an inference step is sound. The following such property is most natural, stating that all lemmas are valid and that the hypotheses are founded on the goals, i.e. that for each counterexample for a hypothesis there is a smaller counterexample for a goal.

Definition 42 (Correctness of Proof State)
A proof state (L, H, G) is called *correct* if L is valid and $H \curvearrowright G$.

While the first part of this definition should be immediately clear, " $H \curvearrowright G$ " states that the goals carry the proof work (which has to be done for the hypotheses) in such a way that the transformation of a goal may make use of hypotheses which are smaller (w.r.t. our induction ordering $<$) than the goal itself since minimal counterexamples for goals cannot be deleted that way. While "correctness of proof states" obviously formulates this idea, it is not the only possible way to do it:

Definition 43 (Weak Correctness of Proof State)
A proof state (L, H, G) is called *weakly correct* if
L is valid and $(H \not\swarrow G \;\Rightarrow\; (\text{form}[H] \text{ is valid}))$.

By the corollaries 34 and 32(3) we get:

Corollary 44 *If a proof state is correct, then it is weakly correct, too.*

As announced above, correctness of proof states really permits us to conclude that the inference steps of our frame inference system are sound:

Lemma 45 (Soundness of Inference Steps)
If (L, H, G) is a [weakly] correct proof state, then an inference step " $(L, H, G) \vdash (L', H', G')$ " (with the above rules) is sound.

Furthermore, correctness holds indeed for an initial state and is kept invariant by the frame inference system:

As a corollary of corollaries 33 and 44 we get:

Corollary 46 (Initial State is Correct)
Let L be valid and $H \subseteq G$. Now (L, H, G) is [weakly] correct.

Lemma 47 (Invariance of Correctness of Proof States)
If " $(L, H, G) \vdash (L', H', G')$ " (with the above rules) and the proof state (L, H, G) is [weakly] correct, then '(L', H', G')' is [weakly] correct, too.

Finally, "correctness of proof states" as an invariance property is not only useful to conclude soundness of single steps, but also globally useful, which can be seen in the following corollary stating that the lemmas and hypotheses gathered in a final proof state are valid:

As a corollary of the corollaries 33, 32(1), and 44 we get:

Corollary 48 (For Final State: Correctness means Validity)
$(L', H', \emptyset)$ *is [weakly] correct iff " $L' \cup \mathrm{form}[H']$ " is valid.*

3.6 The Backwards Approach

The backwards approach was first formalized in Becker (1994).

Definition 49 (Inductiveness and Inductive Soundness)
A proof state (L, H, G) is called *inductive* if $((L \text{ is valid}) \Rightarrow H \not\swarrow G)$.
The inference step " $(L, H, G) \vdash (L', H', G')$ " is called *inductively sound*[14] if inductiveness of (L', H', G') implies inductiveness of (L, H, G).

Inductiveness is a technical notion abstracted from inference systems similar to the frame inference system of section 3.4. Roughly speaking, inductiveness of a state means that an inductive proof of it is possible in the sense that a final (i.e. successful) proof state can be entailed. This is because the goals can be deleted, since there are either false lemmas (ex falso quodlibet) or false hypotheses below all invalid goals.

Inductive soundness can replace soundness of proof steps, by the following argument, which (just like soundness) requires to think $\vdash$ backwards, starting from a final proof state $(L', H', \emptyset)$, which is always inductive. Now, if the steps deriving a final state are inductively sound, then all states involved are inductive. Finally, inductiveness of an initial state implies validity of the initial set of goals.

As a corollary of Corollary 33 we get:

Corollary 50 (Final States are Inductive) $(L', H', \emptyset)$ *is inductive.*

[14] In Becker (1994) this is called *preservation of* (inductive) *counterexamples*, but we cannot use this name here because the notion of "counterexample" is different there.

By Corollary 32(5) for the forward and by the corollaries 33, 34 and 32(3) for the backward direction we get:

Corollary 51 (For Initial State: Inductiveness means Validity of Goals)
*Let L be valid and $H \subseteq G$. Now, '*form$[G]$*' is valid iff (L, H, G) is inductive.*

By the corollaries 50 and 51 we conclude:

Corollary 52 *If all inference steps in "$(\emptyset, \emptyset, G) \vdash^* (L', H', \emptyset)$" are inductively sound, then '*form$[G]$*' is valid.*

Indeed, we have:

Lemma 53 (Inductive Soundness of Inference Steps)
An inference step with the above rules is inductively sound.

Unlike soundness, inductive soundness also captures the basic idea of our frame inference system which for the analytic approach had to be expressed by some correctness property, namely the idea that transformations of goals may make use of hypotheses which are smaller than the goal itself since minimal counterexamples for goals can never be deleted that way. Note, however, that "being not inductive" is no invariance property of $\vdash$ (like correctness is) because one never knows whether it holds for some state or not: If all steps are inductively sound, we only know that the property of "being not inductive" is never removed by an inference step, but this does not mean that it ever holds. Especially for successful proofs it never does, cf. Corollary 50. Instead, inductiveness (i.e. "being inductive") is an invariance property of $\dashv$.

3.7 Results of the Two Approaches

While the relation between the two approaches of the previous two sections is not simple, both seem to be equally useful in capturing the ideas presented in the beginning of section 3.1 as well as in explaining the soundness of our inference system: The following is a corollary of 41, 45, 46, & 47, as well as independently a corollary of 52 & 53:

Corollary 54 (Soundness of "$(\emptyset, \emptyset, G) \vdash^* (L', H', \emptyset)$")
*If "$(\emptyset, \emptyset, G) \vdash^* (L', H', \emptyset)$" (with the above rules), then "*form$[G]$*" is valid.*

The analytic approach even permits a slightly stronger conclusion via Corollary 48:

Corollary 55 *If "$(\emptyset, \emptyset, G) \vdash^* (L', H', \emptyset)$" (with the above rules), then "*form$[G] \cup L' \cup$ form$[H']$*" is valid.*

Considering the design of concrete inference systems by presenting sub-rules of the frame inference rules, another advantage of the analytic approach could be that the additional assumption of correctness of the states could be essential for the sub-rule relationship.

Finally, we compare the analytic and the backwards approach independently of our frame inference system. Here, we consider one approach to be superior to the other when it permits additional successful proofs, whereas we do not respect the fact that one notion may be more appropriate for effective concretion than the other. Invariance of correctness cannot be superior to invariance of weak correctness or to inductive soundness, since a step with the former and without one of the latter properties starts from an invalid set of goals and thus the required soundness of inference steps does not permit additional proofs. Invariance of weak correctness cannot be superior to invariance of correctness (or else to inductive soundness), since a step with the former and without one of the latter properties leads to (or else starts from) an invalid set of goals and thus the required soundness of inference steps does not permit additional proofs. Finally, inductive soundness is very unlikely to be superior to invariance of [weak] correctness, since a step with the former and without one of the latter properties leads to an invalid set of lemmas or hypotheses. Moreover, if we do not consider all proofs but only the existence of proofs, then (on our non-effective level!) all approaches are equivalent: Using the Deletion rule $|G|$ -times we get:

Corollary 56 (Completeness of " $(\emptyset,\emptyset,G)\vdash^*(L',H',\emptyset)$ ")
If " form[G] " is valid, then " $(\emptyset,\emptyset,G)\vdash^*(\emptyset,\emptyset,\emptyset)$ " (with the above rules).

Note, however, that (as far as we know) for the construction of *effective* concrete inference systems based on rules which are no (effective) sub-rules of the rules of our frame inference system, each of the three approaches (i.e.: soundness and invariance of correctness; soundness and invariance of weak correctness; inductive soundness) may be superior to each of the others. The same may hold for generalizing them to inference systems on generalized proof states. E.g., for

Parallelization: $\dfrac{(L,H,G\cup G')}{(L,H,G)\qquad(L,H,G')}$

it is obvious how to generalize inductive soundness, whereas the two other approaches do not seem to permit an appropriate generalization based on local properties of triples (L,H,G).

3.8 The "Switched" Frame Inference System

In section 3.1 we pointed out that we have to avoid non-terminating reasoning cycles between hypotheses and goals. In our formalization we achieved this by founding a hypothesis S' on smaller or equal goals from G, i.e. " $\{S'\}\curvearrowright G$ " (cf. the condition of the Hypothesizing rule), and by applying to a goal S only strictly smaller hypotheses from H, i.e. " $\{S\}\searrow H$ " (cf. the condition of the Deletion rule). From a cyclic reasoning " $H\curvearrowright G\searrow H$ " we immediately get " $H\searrow H$ " and " $G\searrow G$ " by Corollary 34, and then 'form[$H\cup G$]' is valid by Corollary 32(3), which means that the reasoning cycle is sound. Now an alternative way to achieve this is the following: Instead of doing our quasi-decreasing step '$\curvearrowright$' from hypotheses to goals " $H\curvearrowright G$ " and our strictly decreasing step '$\searrow$' from goals to hypotheses " $G\searrow H$ ", we could go from hypotheses to goals with a strict and from goals to hypotheses with a quasi step. More precisely: The condition of the Hypothesizing rule would be changed into " if L is invalid or $\{S\}\searrow/\curvearrowright(G,H)$ ", and the condition of the Deletion rule into " if L is invalid or $\{S\}\curvearrowright G\cup H$ ". The Expansion and the Acquisition rules remain unchanged. A Memorizing rule cannot exist and the Transformation rule is composed of several Expansions, possibly followed by a Hypothesizing, and then a Deletion into one of the following forms:

Memorizing Switched Transformation: $\dfrac{(L \;, H \;, G \cup \{S\}\;)}{(L \;, H \cup \{S\} \;, G \cup M \;)}$

if L is invalid or $\{S\} \searrow\!/\!\curvearrowright (G \cup M, H)$

Simple Switched Transformation: $\dfrac{(L \;, H \;, G \cup \{S\}\;)}{(L \;, H \;, G \cup M \;)}$

if L is invalid or $\{S\} \curvearrowright G \cup M \cup H$

When we then also switch '$\curvearrowright$' and '$\searrow$' (i.e. replace one by the other) in the definitions of "correctness" and "inductiveness" and when we require an initial state additionally to have an empty set of hypotheses, then we analogously get the results of section 3.7 for the soundness also of the *switched* frame inference system. The reasons why we prefer the non-switched version presented here are the following:

From a user's point of view, the non-switched version may be more convenient, because hypotheses become available earlier and easier via the Memorizing rule, which cannot exist for the switched version.

From the inference system designer's point of view, the non-switched version is more convenient, due to the following argumentation: With both the switched and the non-switched version of the inference system, a proof can be thought to consist mainly of steps of the kind that a goal S may become available as a hypothesis and is then transformed into sub-goals M. For the non-switched case this can be achieved by an application of the Memorizing and then of the Transformation rule. For the switched inference system this is just a Memorizing Switched Transformation. The shortcoming of the Memorizing Switched Transformation is that the transformation of a goal into sub-goals has to be strictly decreasing instead of quasi-decreasing (as required for the non-switched case). The design of quasi-decreasing transformations, however, is easier than that of strictly decreasing ones, for the same reason as exhibited in section 3.1 ("Easier Design of Inference Rules"): The solution suggested in Example 30 for making (30.1.1) smaller than (30.1) (namely to set the weight of (30.1.1) to the weight of (30.1)) does not solve the problem for the Memorizing Switched Transformation where (30.1.1) has to be *strictly* smaller than (30.1). Therefore, the non-switched inference system admits fine grain inference steps (which is one of our design goals of section 1.3), which in the switched system have to be replaced with a very big inference step bridging over all quasi-decreasing steps until a strictly decreasing step is reached. Moreover, each simplification step with the Memorizing Switched Transformation rule has to decrease the weight of the goal strictly, such that the possibility to apply some fixed smaller hypothesis gets more and more unlikely with each simplification step, cf. section 3.1 ("High Quality of Ordering Information"), esp. Example 28. If we, however, use the *Simple* Switched Transformation instead, then the goal is not made available as a hypothesis. Thus, in order not to lose the possibility to apply a hypothesis, simplification should not be done via inference steps of the switched inference system, but incorporated into the hypotheses applicability test. With the non-switched version, however, the power of the whole inference system can be homogeneously used for simplification, as required by our design goals of section 1.3. Finally, the required decision whether to apply the *Memorizing* or the *Simple* Switched Transformation rule does not comply with our design goal of a "natural flow of information", cf. section 1.3.

On the other hand, the comparison of weights for an applicability test of a hypothesis to a goal is simpler in the switched inference system because there the weight of the hypothesis is often equal to the weight of the goal, in which case the test is successful. While this does not allow for additional proofs with the switched inference system, it may avoid possibly complicated reasoning on ordering properties.

3.9 Classifying Other Work

In this section we give an incomplete sketch of the literature on inference systems for implicit induction and briefly classify these inference systems according to our presentation here.

In Bachmair (1988) our sets of hypotheses and goals are not separated yet. A disadvantage of this is that a success of a proof due to an empty set of goals is more difficult to detect and that understanding the inference system gets more difficult without the concepts of hypotheses and goals. The missing separation into hypotheses and goals also requires both the foundedness step from hypotheses to goals (as in our switched system of section 3.8) and the step from goals to hypotheses (as in the non-switched system of section 3.4) to be strictly decreasing (i.e. '$\searrow$'), which means a combination of the disadvantages of both the switched and the non-switched approach. The soundness of Bachmair's inference system results from the fact that a fair derivation sequence $M_i \vdash M_{i+1}$ $(i \in \mathbb{N})$ always satisfies $M_i \curvearrowright M_{i+1}$ and has a sub-sequence such that $\forall i \in \mathbb{N}$: $M_{j_i} \searrow M_{j_{i+1}}$, which implies that 'form$[M_0]$' is valid, cf. corollaries 38, 37, and 32(6). The foundedness relations are defined by use of sizes of inconsistency proofs for the equations in M_i instead of the counterexamples themselves. As already mentioned in section 3.2, it is in general undesirable to base the notions of a frame inference system on formal proofs instead of semantic notions because the latter impose no operational restrictions and are likely to change less frequently. One of the operational restrictions in Bachmair (1988) is the (ground) confluence requirement for the defining rules. That this restriction can be removed was noted in Gramlich (1989) by defining the foundedness relations by use of the sizes of positive proofs measured via the applications of equations from M_i.

The important separation between hypotheses and goals was introduced in Reddy (1990), where a frame inference system similar to our switched one in section 3.8 is used, the argumentation for soundness follows the analytic approach using operationally restricted versions of soundness and (switched) correctness, and the foundedness notions are still the operationally restricted ones of Gramlich (1989). In Wirth (1991) this operational restriction is overcome by using a semantic foundedness notion.

In Fraus (1993) we have found the first[15] argumentation for soundness following the backwards approach. While the inference system already is of the (superior) non-switched style, the foundedness relations are still operationally restricted (by measuring positive proofs in the natural deduction calculus), — a restriction that is overcome in Fraus (1994). In Becker (1994) we finally find the backwards approach based on semantic foundedness notions as presented here.

[15] This actually goes back to an unpublished manuscript of the year 1988 by Alfons Geser at the University of Passau entitled "An inductive proof method based on narrowing".

3.10 Safe Steps and Failure Recognition

As already mentioned in section 1.2 the ability of an inductive theorem prover to detect invalid formulas is important under a practical aspect; especially for *inductive* theorem proving because of the important role generalizations play in it: An invalid formula can result from a valid input theorem due to over-generalization. The following notions are useful for the discussion of failure recognition and refutational completeness (cf. Appendix A):

Definition 57 (Validity of a State / Safety of Inference Steps and Rules)

A proof state (L, H, G) is *valid* if $L \cup \text{form}[H \cup G]$ is valid.

The inference step " $(L, H, G) \vdash (L', H', G')$ " is *safe* if [16]

validity of (L, H, G) implies validity of (L', H', G').

An inference rule is *safe* if it describes only safe inference steps.

It is not reasonable to require all possible steps of an inductive theorem prover to be safe, since this property is undecidable for generalization steps which play a major role in inductive theorem proving. For concrete inference systems, however, it is usually possible to give interesting sufficient conditions for the application of an inference rule to be safe.

The following is a corollary of Corollary 32(6) [and Corollary 34]:

Corollary 58 *The [Switched] Hypothesizing, Acquisition, [Switched] Deletion, and Memorizing rules are safe.*

Note that the Expansion rule and its derivatives (i.e. the Transformation, Memorizing Switched Transformation, and Simple Switched Transformation rule) are unsafe in general. Nevertheless, nearly all sub-rules of the Transformation rule we will present in the following sections are safe.

Since it is undecidable in general whether a proof state is invalid, regarding the operationalization of invalidity of proof states we had better content ourselves with some incomplete failure predicate 'FAIL' defined on sets[17] of formulas, which should be sound in the following sense:

Definition 59 (Soundness of a Failure Predicate)

The failure predicate 'FAIL' is *sound* if $\forall F \in \text{FAIL}$: ($F$ is invalid).

[16] Note that if we excluded the sets L and L' from the definition of safety, then Corollary 58 and the lemmas 93 and 96 would not hold any more because all rules whose safety is stated there (with the exception of the Memorizing rule) would become unsafe in general. Moreover, if we excluded the sets H and H' from the definition of safety, then the Hypothesis Apply and the Hypothesis Rewrite rule would become unsafe, contrary to what is stated in the lemmas 113 and 117.

[17] Note that this failure predicate is defined on *sets of* formulas for operational reasons, namely in order to be able to recognize that one formula contradicts another one; whereas for a theoretical treatment it would be sufficient to define it on single formulas since one of those formulas has to invalid in a consistent specification; but to find out which formula it is is undecidable in general.

Definition 60 (Failure State)
A proof state (L, H, G) is a *failure state (w.r.t.* FAIL*)* if [18]

$$(L \cup \text{form}[H \cup G]) \in \text{FAIL}.$$

Corollary 61 *If all inference steps in "* $(L, H, G) \vdash^* (L', H', G')$ *" are safe and* (L', H', G') *is a failure state w.r.t. a sound failure predicate, then* (L, H, G) *is invalid.*

Now, when an inductive theorem prover has realized that a proof state (L'', H'', G'') is a failure state, the following questions arise: How far do we have to backtrack its derivation to reach a valid state? Was our original state valid? To recover from this failure (after setting (L', H', G') to (L'', H'', G'')) we may iterate the following:

> If this (L', H', G') is the original input state (with L' known to be valid and $H' \subseteq G'$), then we have refuted our original set of goals G' and should stop proving. Otherwise the step that yielded (L', H', G'), say $(L, H, G) \vdash (L', H', G')$, has to be carefully inspected: If it is known to be safe we backtrack this step (setting (L', H', G') to (L, H, G)) and reiterate. Otherwise it might be possible to find a (minimal) subset of $L'' \cup \text{form}[H'' \cup G'']$ for which the failure predicate FAIL is still known to hold and which also is (implied by) a subset of $L \cup \text{form}[H \cup G]$; in which case we also backtrack this step (setting (L', H', G') to (L, H, G)) and reiterate. Otherwise, when $(L, H, G) \vdash (L', H', G')$ is likely to be an unsafe step which might have caused the invalidity, we backtrack this step and may try to go on with a hopefully safe inference step from (L, H, G) instead.

Note that the notion of safety becomes even more useful for detecting invalid states when (instead of representing our goals by an unstructured set) we use the obvious structure of a forest for our goals which is standard for sequent calculi in the style of Gentzen (1935) and which is indicated in our examples via the labels of the goals.

[18] Note that we have included L and H (instead of just testing G) on the one hand in order to match Definition 57 (Why this definition includes L and H is explained in footnote 16) and on the other hand because we want to be able to detect an invalid lemma or hypothesis when it has just been generated and do not want to have to wait until it will have been harmfully applied to a goal. One is tempted to argue that, in case of [weak] correctness of a proof state, an invalid hypothesis implies the existence of an invalid goal, but this argument again does not respect the operational aspect, i.e. that a decidable failure predicate cannot be complete in general.

4 Towards the Concrete Inference System

In this section we will concretize the abstract notions and the frame inference system of Wirth & Becker (1995) as presented in section 3 of this paper in order to obtain the more concrete inference system for proving inductive validity we want to present in this paper. More precisely, we will concretize the classes 'Form', 'SynCons', 'form', 'Info', the induction ordering, and the notions of "counterexample" and "validity".

The set 'Form' of *formulas* is instantiated to the set 'Form(sig, V)', cf. Definition 7. We assume some abstract set 'Weight' of *weights*, cf. section 4.2. The set of *syntactic constructs* is instantiated to SynCons := Form×Weight. A weight '$\aleph$' in a syntactic construct '$(\Gamma, \aleph)$' is intended to describe some measure controlling the inductive application of 'Γ'. The function " form: SynCons $\rightarrow$ Form " is simply given by " form$((\Gamma, \aleph)) := \Gamma$ ". Next we have to specify what kind of validity we are going to show. While our formulas are well-suited for proving ground confluence (cf. Becker (1993) or Becker (1994)), we want to restrict ourselves here to the case of inductive validity, assuming that a ground confluent rule system is given:

Global Requirement 62 R *is a* Def-*MCRS over* sig/cons/V. $\longrightarrow_{R,\emptyset}$ *is confluent.*

For a comprehensive survey on how to achieve confluence by syntactical means cf. Wirth (1995).

4.1 Counterexamples and Inductive Validity

As exhibited in Wirth & al. (1993), Wirth & Gramlich (1994b) and stated in Wirth & Becker (1995) and section 1.1, it is appropriate to call a formula 'Γ' "inductively valid" if for all "inductive substitutions" 'τ' and all algebras '$\mathcal{A}$' belonging to some special model class 'K', the instantiated formula $\Gamma\tau$ is valid in $\mathcal{A}$. Proceeding in this style in principle, but reducing the technical complexity, we define the notion of "counterexamples" first.

A discussion of the following cases follows Definition 65.

Definition 63 (*A*-/ *B'*-/ *B*-/ *C*-/ *D'*-/ *D*-/ *E*-**case,** Info)
The following cases are said to hold if their associated requirements hold:

A-**case:** K is the class of all sig/cons-models of R.

B'-**case:** K is the class of all CONS:cons-term-generated sig/cons-models of R.

B-**case:** K is the class of all constructor-minimal models of R.

C-**case:** K is the class of all CONS:cons-term-generated, constructor-minimal models of R.

D'-**case:** K is the class of all SIG:cons-term-generated, constructor-minimal models of R.

D-**case:** $\mathrm{X} = \mathrm{V}_{\mathrm{SIG}}$, $\mathcal{I} = \mathcal{T}(\mathrm{X})/\overset{*}{\longleftrightarrow}_{\mathrm{R,X}}$, $\mathrm{K} = \{\mathcal{I}\}$, $\iota_{\mathcal{I}}$ is given by $x \mapsto \overset{*}{\longleftrightarrow}_{\mathrm{R,X}}[\{x\}]$ $(x \in \mathrm{X})$, and $\longrightarrow_{\mathrm{R,X}}$ is confluent.

E-**case:** $\mathrm{X} = \emptyset$, and the rest is like in the *D*-case, i.e., $\mathcal{I} = \mathcal{GT}/\overset{*}{\longleftrightarrow}_{\mathrm{R},\emptyset}$, $\mathrm{K} = \{\mathcal{I}\}$, and $\iota_{\mathcal{I}} = \emptyset$.

'Info', the class of *extra information* intended to exhibit the reason why a formula (of a syntactic construct) is invalid, is defined to be the class of pairs $(\tau, \mathcal{A})$ with $\tau \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{V_{SIG}}^{\mathcal{A}}))$ and $\mathcal{A} \in \mathrm{K}$.
In the *D*- or *E*-case, for $(\tau, \mathcal{A}) \in \mathrm{Info}$ we additionally require $\tau \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}))$.

Unless we are in the *D'*-, *D*- or *E*-case, for the previous and the following definition it is important to have the last paragraph of section 2.5 in mind, where we explained why we can w.l.o.g. consider all algebras to be factor algebras of term algebras. This means that for $(\tau, \mathcal{A}) \in \mathrm{Info}$ depending on $\mathcal{A}$ it may be necessary to extend the '$\mathrm{V_{SIG}}$' in " $\tau \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{V_{SIG}}))$ " with new variables to $\mathrm{V_{SIG}}^{\mathcal{A}}$ until its cardinality is big enough for the existence of an $\mathcal{A}$-valuation $\iota_{\mathcal{A}} \in \mathcal{SUB}(\mathrm{V_{SIG}}^{\mathcal{A}}, \mathcal{A})$ such that $\mathcal{A}_{\iota_{\mathcal{A}}}::\mathcal{T}(\mathrm{V_{SIG}}^{\mathcal{A}}) \to \mathcal{A}$ is surjective. This $\iota_{\mathcal{A}}$ also occurs in the following definition.

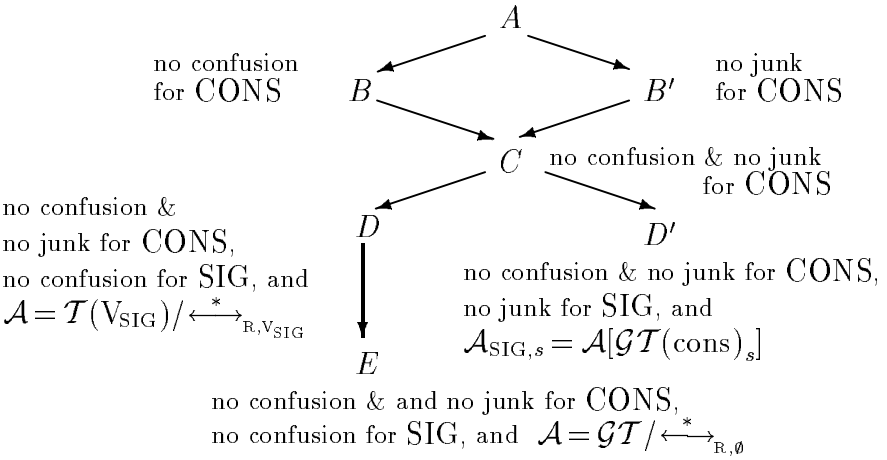
Definition 64 (Counterexamples)
$((\Gamma, \aleph), (\tau, \mathcal{A})) \in \mathrm{SynCons} \times \mathrm{Info}$ is a *counterexample* if $\Gamma\tau$ is false w.r.t. $\mathcal{A}_{\iota_{\mathcal{A}}}$.
Since this concrete notion of counterexamples does not depend on the weight $\aleph$, in the above situation we sometimes simply speak of the "*counterexample* $(\Gamma, \tau, \mathcal{A})$ *(for the formula* Γ*)*".

We now repeat the essential requirement of Wirth & Becker (1995) as presented in section 3.2 of this paper on the relationship between counterexamples and validity. We state it, however, not as a requirement but as the definition of validity. That the resulting notions of validity are nevertheless meaningful and reasonable and that the definition below can be taken as a lemma is described in Wirth & Gramlich (1994b).

Definition 65 (Type-A/ -B'/ -B/ -C/ -D'/ -D/ -E Inductive Validity)
Let $T \in \{A, B', B, C, D', D, E\}$. Assume the T-case to hold.
A formula Γ is *(type-T inductively) valid* if Γ has no counterexample, i.e. there is no $(\tau, \mathcal{A}) \in$ Info such that $(\Gamma, \tau, \mathcal{A})$ is a counterexample.

Type-A formulates the idea that (constructor) variables are meant to denote objects denoted by (constructor) ground terms. However, contrary to all other types, type-A does not restrict the models of the specification that have to be considered, but considers only instances of formulas obtained by inductive substitutions. Type-B' forbids junk in the constructor sub-universe (i.e. the domain of interest). Type-B forbids unnecessary confusion in the constructor sub-universe. Type-C combines both restrictions, which is appealing if one wants to prescribe a precise and fixed knowledge on the basic objects for computation. Type-D' corresponds to the philosophy that a partially defined function is to be interpreted as the set of all possible complete and consistent extensions. Type-D and E finally fix one specific unique minimal model which has neither junk nor confusion in the constructor sub-universe and which can be described constructively in terms of the factor algebra of the term algebra modulo the congruence induced by the reduction relation (provided the latter is confluent) (cf. section 2.9). Type-E uses the ground term algebra $\mathcal{GT}$, which is only adequate (cf. Wirth & Gramlich (1994b)) when no general variables occur in the formula. Therefore, Type-D uses the term algebra $\mathcal{T}(\mathrm{V_{SIG}})$ over $\mathrm{V_{SIG}}$. In a sense, type-D means *inductive semantics* for $\mathrm{V_{CONS}}$ and *free semantics* for $\mathrm{V_{SIG}}$.

The basic characteristics of and the relations between these notions of inductive validity are illustrated in the figure below where an arrow indicates that validity of the one type implies validity of the other type. Missing arrows indicate non-implications.



Note that the types A and B show the following "misbehaviour": Let x be an implicitly universally quantified constructor variable and t be a non-constructor term of the same sort. According to our intuition behind the concept of "constructor variables", validity of (Def t) and Γ should imply validity of $\Gamma\{x\mapsto t\}$. This is the case with deductive validity as well as inductive validity of the types B', C, D', D, and E. It is not the case with inductive validity of types A and B, however, because they admit junk in the constructor sub-universes. The misbehaviour of these two types is not only intuitional but also makes a reasonable treatment of generalized substitutions (cf. section 4.3) by our inference rules impossible. Therefore we impose the following restriction, saying that junk is not admitted in constructor sub-universes.

Global Requirement 66 K *is a sub-class of the class of* CONS:cons-*term-generated* sig/cons-*models of* R.

Note that this requirement is satisfied for the types B', C, D', D, and E; whereas it excludes the types A and B. In order to get a feeling that this is not a severe exclusion, note that all formulas of Example 10 are type-C inductively valid w.r.t. R_9.

We confess that we consider the types C and D to be the most important ones. Moreover, since we favour the idea of "partially specified functions" to that of "functions specified to be partial", type C is our favourite of the two. Nevertheless, type D is important for the specifier when he wants find out which cases of a function definition are left open.

4.2 The Induction Ordering

Since we denote our weights with Hebrew letters which may not be known to the reader we introduce the beginning of the Hebrew alphabet: $\aleph$ aleph, $\beth$ beth, $\gimel$ gimel, and $\daleth$ daleth. The weight $\aleph$ of a syntactic construct $(\Gamma, \aleph)$ is intended to augment the formula Γ with additional structure for expressing ordering conditions to avoid non-terminating cycles in the inductive argumentation. The connection between a formula and its weight may be established via common (implicitly universally quantified) variables. Some of our inference steps will apply some substitution to the formula. In this case the connection between a formula and its weight can be preserved only if the substitution is applied to the weight as well. Therefore the following is appropriate:

Global Requirement 67
Each weight $\aleph \in \mathrm{Weight}$ *has associated with it a certain finite set* $\mathcal{V}(\aleph)$ $(\subseteq \mathrm{V})$ *of* input variables.
Furthermore, there must be some function which maps each $\aleph \in \mathrm{Weight}$ *and* $\mu \in \mathcal{GENSUB}(\mathrm{V}, \mathcal{T}) \cup \bigcup_{\mathcal{A}\in \mathrm{K}} \mathcal{GENSUB}(\mathrm{V}, \mathcal{T}(\mathrm{V}_{\mathrm{SIG}}{}^{\mathcal{A}}))$ *to an object we just denote by '*$\aleph\mu$*'.*
We require $\mathcal{V}(\aleph\mu) \subseteq \mathcal{V}(\mu[\mathcal{V}(\aleph)])$; $\mu|_{\mathcal{V}(\aleph)} = \mu'|_{\mathcal{V}(\aleph)} \Rightarrow \aleph\mu = \aleph\mu'$; *and* $\mu \in \mathcal{GENSUB}(\mathrm{V}, \mathcal{T}) \Rightarrow \aleph\mu \in \mathrm{Weight}$.

Global Requirement 68
For each sig/cons-*algebra* $\mathcal{A} \in \mathrm{K}$ *we require a wellfounded quasi-ordering* $\lesssim_{\mathcal{A}}$ *on*
$$\{\ \beth(\mu\tau) \mid \beth \in \mathrm{Weight} \wedge \mu \in \mathcal{GENSUB}(\mathrm{V}, \mathcal{T}) \wedge (\tau, \mathcal{A}) \in \mathrm{Info}\ \}.$$

Definition 69 (Induction Ordering)
The induction ordering '$\lesssim$' is given as follows (for two counterexamples $((\Delta, \beth), (\pi, \mathcal{B}))$ and $((\Gamma, \aleph), (\tau, \mathcal{A}))$):
$$((\Delta, \beth), (\pi, \mathcal{B})) \lesssim ((\Gamma, \aleph), (\tau, \mathcal{A})) \quad \text{if} \quad \mathcal{B} = \mathcal{A} \ \text{ and } \ \beth\pi \lesssim_{\mathcal{A}} \aleph\tau.$$

Global Requirement 70
For each $(\tau, \mathcal{A}) \in \mathrm{Info}$; $\mu \in \mathcal{GENSUB}(\mathrm{V}, \mathcal{T})$; *and* $\beth \in \mathrm{Weight}$; *we require*
$$(\beth\mu)\tau \approx_{\mathcal{A}} \beth(\mu\tau).$$

Definition 71 ($\mathcal{V}_<(\ldots)$)
We define $\mathcal{V}_<(A)$, the set of variables occurring in '$<$'-literals of A, in the following way: $\mathcal{V}_<(A) := \emptyset$ for $A \in \mathcal{AT}(\mathrm{sig}, \mathrm{V})$; $\mathcal{V}_<(\daleth{<}\aleph) := \mathcal{V}(\daleth) \cup \mathcal{V}(\aleph)$ for $\daleth, \aleph \in \mathrm{Weight}$; and $\mathcal{V}_<(\overline{A}) := \mathcal{V}_<(A)$ for $A \in \mathcal{GENAT}(\mathrm{sig}, \mathrm{V})$. For the formula $\lambda_0 \ldots \lambda_{n-1}$ we define $\mathcal{V}_<(\lambda_0 \ldots \lambda_{n-1}) := \bigcup_{i \prec n} \mathcal{V}_<(\lambda_i)$.

Definition 72 (Validity of Formulas in sig/cons**-algebras; Part II)**
(continuing Definition 11)
Let $(\beth{<}\aleph) \in \mathcal{GENAT}(\mathrm{sig}, \mathrm{V})$. Let $(\tau, \mathcal{A}) \in \mathrm{Info}$.
$(\beth{<}\aleph)\tau$ is *true w.r.t.* $\mathcal{A}_{\iota_{\mathcal{A}}}$ if $\beth\tau <_{\mathcal{A}} \aleph\tau$.
$(\beth{<}\aleph)\tau$ is *false w.r.t.* $\mathcal{A}_{\iota_{\mathcal{A}}}$ if $\beth\tau \not<_{\mathcal{A}} \aleph\tau$.

Now we discuss a special type of induction orderings that admits additional inference rules, namely the *semantic* induction orderings. We will treat this type as a global special case for our inference system, similar to the cases for different types of inductive validity. The case that the "induction ordering is semantic" has the following meaning: Suppose that a proof contains an application of an inference rule that requires the induction ordering to be semantic. Then the ordering we may choose for satisfying the ordering conditions represented by the ordering literals of the goals has to be a semantic one.

Definition 73 (Semantic Induction Ordering)
In case of
$\forall \mu,\nu \in \mathcal{GENSUB}(\mathrm{V},\mathcal{T})$: $\forall \tau,\pi$: $\forall \mathcal{A}$: $\forall \aleph \in \mathrm{Weight}$:

$$\left(\begin{array}{ll} & (\tau,\mathcal{A}),(\pi,\mathcal{A}) \in \mathrm{Info} \\ \wedge & (\mu\tau\mathcal{A}_{\iota_{\mathcal{A}}})|_{\mathcal{V}(\aleph)} = (\nu\pi\mathcal{A}_{\iota_{\mathcal{A}}})|_{\mathcal{V}(\aleph)} \end{array} \right) \;\Rightarrow\; (\aleph\mu)\tau \approx_{\mathcal{A}} (\aleph\nu)\pi \;\Bigg)$$

we say that *the induction ordering is semantic.*

In case of a semantic induction ordering the following manipulation of weights becomes reasonable:

Definition 74 (Rewriting Weights and '<'-Literals)
Let $s \in \mathbb{S}$ and $l,r \in \mathcal{T}_{\mathrm{SIG},s}$. Let $\aleph,\aleph',\beth,\beth' \in \mathrm{Weight}$.
$\aleph$ *rewrites to* $\aleph'$ *with the rule* $l{=}r$ if
$\exists \gimel \in \mathrm{Weight}$: $\exists \mu,\mu' \in \mathcal{GENSUB}(\mathrm{V},\mathcal{T})$:

$$\left(\begin{array}{lll} & \aleph = \gimel\mu & \\ \wedge & \aleph' = \gimel\mu' & \\ & & \\ \wedge & \forall x \in \mathcal{V}(\gimel)\text{: } \exists P\text{: } & \left(\begin{array}{ll} & P \subseteq \mathcal{POS}(x\mu) \cap \mathcal{POS}(x\mu') \\ \wedge & \forall p \in P\text{: } x\mu/p = l \\ \wedge & x\mu' = x\mu[\,p \leftarrow r \mid p \in P\,] \end{array} \right) \end{array} \right)$$

A generalized literal $(\beth{<}\aleph)$ *rewrites to* $(\beth'{<}\aleph')$ *with the rule* $l{=}r$ if $\aleph$ rewrites to $\aleph'$ and $\beth$ rewrites to $\beth'$ with $l{=}r$.

Corollary 75
Let $s \in \mathbb{S}$ *and* $l,r \in \mathcal{T}_{\mathrm{SIG},s}$. *Let* $\aleph,\aleph',\beth,\beth' \in \mathrm{Weight}$.
Assume the induction ordering to be semantic and $((l{\neq}r),\tau,\mathcal{A})$ *to be a counterexample. Now:*

1. *If* $\aleph$ *rewrites to* $\aleph'$ *with* $l{=}r$, *then* $\aleph\tau \approx_{\mathcal{A}} \aleph'\tau$.
2. *If* $(\beth{<}\aleph)$ *rewrites to* $(\beth'{<}\aleph')$ *with* $l{=}r$, *then* $((\beth{<}\aleph),\tau,\mathcal{A})$ *is a counterexample iff* $((\beth'{<}\aleph'),\tau,\mathcal{A})$ *is a counterexample.*

Lemma 76
Let $\mu,\nu \in \mathcal{GENSUB}(\mathrm{V},\mathcal{T})$; $(\tau,\mathcal{A}),(\pi,\mathcal{A}) \in \mathrm{Info}$; *and* $\Pi \in \mathrm{Form}(\mathrm{sig},\mathrm{V})$.
Assume that $(\mu\tau\mathcal{A}_{\iota_{\mathcal{A}}})|_{\mathcal{V}(\Pi)} = (\nu\pi\mathcal{A}_{\iota_{\mathcal{A}}})|_{\mathcal{V}(\Pi)}$.
Moreover, assume either $(\mu\tau)|_{\mathcal{V}_<(\Pi)} = (\nu\pi)|_{\mathcal{V}_<(\Pi)}$ *or that the induction ordering is semantic.*
Now the following two statements are logically equivalent:

- $(\Pi\mu,\tau,\mathcal{A})$ *is a counterexample.*
- $(\Pi\nu,\pi,\mathcal{A})$ *is a counterexample.*

4.3 Generalized Substitutions

Suppose we have shown that addition on natural numbers is commutative. More precisely, that we have proved the lemma

$$x + y = y + x$$

of Example 10 where x and y are constructor variables of the sort nat. Now suppose we have to show the goal

$$(u + v) + y = y + (u + v)$$

where u and v are also constructor variables of the sort nat. This seems to be just an instance of the lemma above, but a more careful look at it reveals that we have to instantiate the constructor variable x of the lemma with the non-constructor term $u + v$ of the goal which a substitution is not allowed to do. This is a pity since we also have the lemma (cf. Example 10)

$$\mathrm{Def}\,(x + y)$$

which means that semantically the above instantiation is sound. Fortunately, *generalized substitutions* can do the job. To know, however, that a generalized substitution is semantically sound we must have a careful look at the following set:

Definition 77
For a generalized substitution μ we define the *variables generalized by* μ to be

$$\mathrm{GENDOM}(\mu) := \{\ x \in \mathrm{dom}(\mu) \mid\ x \in \mathrm{V}_{\mathrm{CONS}} \wedge x\mu \notin \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})\ \}.$$

The meaning of the following definition is given by the following lemma which shows that the application of generalized substitutions is sound when they are justified semantically and the weights are treated with special care. As the reader may guess when reading Lemma 76, it is not directly possible to apply a generalized substitution to a constructor variable x occurring in a '<'-literal of a goal $(\Pi, \neg)$, unless we restrict ourselves to semantic induction orderings. Therefore, instead of applying the generalized substitution μ directly to x, we substitute x with a fresh constructor variable y and add the semantic restriction $(y{\neq}x\mu)$ to the formula, expressing that y and $x\mu$ can be assumed to be semantically equal. The same has to be done when x occurs in $\neg$. The 'Z' in the following definition is intended to contain the variables that are already in use when the application of μ to $(\Pi, \neg)$ is considered.

Definition 78
Let $\mu \in \mathcal{GENSUB}(\mathrm{V}, \mathcal{T})$; $(\Pi, \neg) \in \mathrm{SynCons}$; $\mathrm{Z} \subseteq \mathrm{V}$. Now we define:
$(\Pi', \neg', \mathrm{Y}, \Theta)$ *results from application of* μ *to* $(\Pi, \neg)$ *w.r.t.* Z if

$$\mathrm{Y} = \left\{ \begin{array}{ll} \emptyset & \text{if the induction ordering is semantic (cf. Definition 73)} \\ \left(\ \mathcal{V}_<(\Pi) \cup \mathcal{V}(\neg)\ \right) \cap \mathrm{GENDOM}(\mu) & \text{otherwise} \end{array} \right\}$$

and there is some bijection $\xi \in \mathcal{SUB}(\mathrm{V}, \mathrm{V})$ with

$$\xi[\mathrm{Y}] \cap \left(\ \mathrm{Z} \cup \mathcal{V}(\mu[\mathrm{Y}])\ \right) = \emptyset$$

such that

- $(\Pi', \neg')$ differs from $(\Pi, \neg)\mu$ only in that any occurrence of a variable $x \in \mathrm{Y}$ in a '<'-literal of Π or in $\neg$ is replaced with $x\xi$ instead of $x\mu$.
- $\Theta \in \mathrm{Form}$ lists the literals $(x\xi{\neq}x\mu)$ with $x \in \mathrm{Y}$.

Note that in the usual case of $\mathrm{Y}=\emptyset$ we always get $(\Pi',\neg',\mathrm{Y},\Theta)=(\Pi\mu,\neg\mu,\emptyset,\emptyset)$. For $x\in\mathrm{Y}$ the idea is to disconnect x from $x\mu$ syntactically by introducing a new constructor variable $x\xi$ for x and then to express the equality of $x\xi$ and $x\mu$ semantically via $(x\xi{\neq}x\mu)$, cf. Example 114.

Lemma 79

Let $\mu\in\mathcal{GENSUB}(\mathrm{V},\mathcal{T})$, $(\Pi,\neg)\in\mathrm{SynCons}$, $(\tau,\mathcal{A})\in\mathrm{Info}$, $\mathrm{Y}'\subseteq\mathrm{V}$.
Assume that $\forall x\in\mathrm{Y}'\cap\mathrm{GENDOM}(\mu)$: ($(\overline{(\mathrm{Def}\,x\mu)},\tau,\mathcal{A})$ *is a counterexample*).
Now there is some π *such that:*

1. $(\pi,\mathcal{A})\in\mathrm{Info}$.
2. $\pi|_{\mathrm{V}\setminus\mathrm{GENDOM}(\mu)}=(\mu\tau)|_{\mathrm{V}\setminus\mathrm{GENDOM}(\mu)}$.
3. $(\pi\mathcal{A}_{\iota_{\mathcal{A}}})|_{\mathrm{Y}'}=(\mu\tau\mathcal{A}_{\iota_{\mathcal{A}}})|_{\mathrm{Y}'}$.
4. *If* $\left(\begin{array}{l}\mathrm{GENDOM}(\mu)\cap\mathcal{V}_<(\Pi)=\emptyset\\ \vee\quad\textit{the induction ordering is semantic}\end{array}\right)$ *and* $\mathcal{V}(\Pi)\subseteq\mathrm{Y}'$, *then the following two are logically equivalent:*
   - $\Pi\mu\tau$ *is true w.r.t.* $\mathcal{A}_{\iota_{\mathcal{A}}}$.
   - $\Pi\pi$ *is true w.r.t.* $\mathcal{A}_{\iota_{\mathcal{A}}}$.

Additionally assume that $(\Pi',\neg',\mathrm{Y},\Theta)$ *results from application of* μ *to* $(\Pi,\neg)$ *w.r.t.* Z.

5. *If* $\mathrm{Y}\subseteq\mathrm{Y}'$, *then there is some* τ' *such that* $(\tau',\mathcal{A})\in\mathrm{Info}$, $\tau'|_{\mathrm{Z}}=\tau|_{\mathrm{Z}}$, *and* $(\Theta,\tau',\mathcal{A})$ *is a counterexample.*
6. *If* $\mathcal{V}(\Pi,\neg)\subseteq\mathrm{Y}'$ *and if* $(\Theta,\tau,\mathcal{A})$ *and* $(\Pi',\tau,\mathcal{A})$ *are counterexamples, then there is some* ϱ *with* $(\varrho,\mathcal{A})\in\mathrm{Info}$ *such that* $((\Pi,\neg),(\varrho,\mathcal{A}))$ *is a counterexample with* $((\Pi,\neg),(\varrho,\mathcal{A}))\lesssim((\Pi',\neg'),(\tau,\mathcal{A}))$.

4.4 Superposition

In this section we lay the technical foundation for the inference rules that utilize the initiality or freeness of the algebras that establish some specific notion of inductive validity, cf. section 1.1. The reader who is not interested in the proofs should read the following definition only and skip the technical rest of this section.

Definition 80
Let $n \in \mathbb{N}$; $s \in \mathbb{S}$; $t, t' \in \mathcal{T}_{\mathrm{SIG},s}$; $\lambda_0, \ldots, \lambda_{n-1} \in \mathcal{LIT}(\mathrm{sig}, \mathrm{V})$; $\mathrm{Y} \subseteq \mathrm{V}$.
Let $\xi \in \mathcal{SUB}(\mathrm{V}, \mathrm{V})$ be a bijection that maps all variables occurring in rules of R apart from Y (i.e. $\xi[\mathcal{V}(\mathrm{R})] \cap \mathrm{Y} = \emptyset$). Now let $\mathrm{SUPERPOSE}(t, \mathrm{Y})$ be the set containing for any rule $(l{=}r \longleftarrow \lambda_0 \ldots \lambda_{n-1}) \in \mathrm{R}$ and any non-variable position $p \in \mathbb{F}\mathcal{POS}(t)$ such that $l\xi$ and t/p are unifiable the pair

$$(\ (\overline{\lambda_0} \ldots \overline{\lambda_{n-1}})\xi \ , \ \ \mathrm{mgu}(\{(l\xi, t/p)\}, \mathrm{Y} \cup \xi[\mathcal{V}(l{=}r \longleftarrow \lambda_0 \ldots \lambda_{n-1})]) \ ) \ .$$

Define $\mathrm{NARROW}(t, t', \mathrm{Y}) :=$

$$\begin{array}{l} \mathrm{SUPERPOSE}(t, \mathrm{Y}) \cup \\ \mathrm{SUPERPOSE}(t', \mathrm{Y}) \cup \\ \{ \ (\emptyset, \ \mathrm{mgu}(\{(t, t')\}, \mathrm{Y})) \mid t \text{ and } t' \text{ are unifiable } \}. \end{array}$$

Lemma 81
Let $\mathrm{X}' \in \{ \ \emptyset, \mathrm{V}_{\mathrm{SIG}}, {\mathrm{V}_{\mathrm{SIG}}}^{\mathcal{A}} \mid \mathcal{A} \in \mathrm{K} \ \}$ *and* $\mathrm{X} \subseteq \mathrm{X}'$. *Let* Y *be a finite subset of* V. *Let* $t \in \mathcal{T}$ *be a linear term with* $\mathcal{V}(t) \subseteq \mathrm{Y}$. *Let* $\tau \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}'))$.
Define $\mathcal{I} := \mathcal{T}(\mathrm{X})/{\stackrel{*}{\longleftrightarrow}_{\mathrm{R,X}}}$. *Define* $\iota \in \mathcal{SUB}(\mathrm{X}, \mathcal{I})$ *by* $(x \in \mathrm{X})$ $x\iota := {\stackrel{*}{\longleftrightarrow}_{\mathrm{R,X}}}[\{x\}]$.
Now if there is some $t'' \in \mathcal{T}$ *such that* $t\tau \stackrel{*}{\longrightarrow}_{\mathrm{R,X}} t''$ *and*

$$\neg \exists \tau' \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}')) \colon \left(\begin{array}{ll} & t'' = t\tau' \\ \wedge & \forall x \in \mathcal{V}(t) \colon \ x\tau \stackrel{*}{\longrightarrow}_{\mathrm{R,X}} x\tau' \\ \wedge & \forall x \in \mathrm{V} \setminus \mathcal{V}(t) \colon \ x\tau = x\tau' \end{array} \right)$$

then there are some $(\Lambda, \sigma) \in \mathrm{SUPERPOSE}(t, \mathrm{Y})$ *and some* π *such that*

1. $\pi \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}'))$.
2. $(\sigma\pi)|_{\mathrm{Y} \setminus \mathcal{V}(t)} = \tau|_{\mathrm{Y} \setminus \mathcal{V}(t)}$.
3. $(\sigma\pi\mathcal{I}_\iota)|_{\mathcal{V}(t)} = (\tau\mathcal{I}_\iota)|_{\mathcal{V}(t)}$.
4. $t \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$ *implies that all literals in* Λ *are negative.*
5. *If* $t \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$ *or if* $\longrightarrow_{\mathrm{R,X}}$ *is confluent, then* $\Lambda\sigma\pi$ *is false w.r.t.* $\mathcal{I}_\iota$.

Lemma 82
Let $X' \in \{ \emptyset, V_{SIG}, V_{SIG}{}^{\mathcal{A}} \mid \mathcal{A} \in K \}$ *and* $X \subseteq X'$. *Assume* $\longrightarrow_{R,X}$ *to be confluent. Let* Y *be a finite subset of* V. *Let* $t, t' \in \mathcal{T}$ *be linear terms with* $\mathcal{V}(t) \cap \mathcal{V}(t') = \emptyset$ *and* $\mathcal{V}(t,t') \subseteq Y$. *Let* $\tau \in \mathcal{SUB}(V, \mathcal{T}(X'))$.
Define $\mathcal{I} := \mathcal{T}(X)/\overset{*}{\longleftrightarrow}_{R,X}$. *Define* $\iota \in \mathcal{SUB}(X, \mathcal{I})$ *by* $(x \in X)$ $x\iota := \overset{*}{\longleftrightarrow}_{R,X}[\{x\}]$.
Now if $t\tau \overset{*}{\longleftrightarrow}_{R,X} t'\tau$, *then there are some* $(\Lambda, \sigma) \in \mathrm{NARROW}(t, t', Y)$ *and some* π *such that*

1. $\pi \in \mathcal{SUB}(V, \mathcal{T}(X'))$.
2. $(\sigma\pi)|_{Y \setminus \mathcal{V}(t,t')} = \tau|_{Y \setminus \mathcal{V}(t,t')}$.
3. $(\sigma\pi\mathcal{I}_\iota)|_{\mathcal{V}(t,t')} = (\tau\mathcal{I}_\iota)|_{\mathcal{V}(t,t')}$.
4. $t, t' \in \mathcal{T}(\mathrm{cons}, V_{\mathrm{CONS}})$ *implies that all literals in* Λ *are negative.*
5. $\Lambda\sigma\pi$ *is false w.r.t.* $\mathcal{I}_\iota$.

Lemma 83
Let $s \in \mathbb{S}$ *and* $t, t' \in \mathcal{T}_{SIG,s}$. *Let* $(\tau, \mathcal{A}) \in \mathrm{Info}$. *Assume one of the following two:*

- *D- or E-case holds or*
- $t, t' \in \mathcal{T}(\mathrm{cons}, V_{\mathrm{CONS}})$, $X = \emptyset$, *and*
  K *is a sub-class of the class of constructor-minimal models of* R.

Now $\longrightarrow_{R,X}$ *is confluent, and if* $\mathcal{A}_{\iota_{\mathcal{A}}}(t\tau) = \mathcal{A}_{\iota_{\mathcal{A}}}(t'\tau)$ *then:*

1. $t\tau \overset{*}{\longleftrightarrow}_{R,X} t'\tau$.
2. *If* t, t' *are linear terms with* $\mathcal{V}(t) \cap \mathcal{V}(t') = \emptyset$, $\mathcal{V}(t,t') \subseteq Y$, *and* Y *is a finite subset of* V, *then* $\mathrm{NARROW}(t, t', Y) \neq \emptyset$.

5 Contextual Rewriting

5.1 Introduction

While our reduction relation $\longrightarrow_{R,V}$ (cf. section 2.9) is able to rewrite terms containing constructor variables, it is actually of little use at this because the condition of an instantiated rule will still contain these constructor variables and therefore it is very unlikely that this condition is fulfilled for all possible (inductive) instantiations, — and even when this is the case, we do not know how to find it out[19]. Therefore, the appropriate way of rewriting terms containing constructor variables with a conditional rule uses additional assumptions implying fulfilledness of the condition of the rule.

This leads us to a species of rewriting which is called *contextual rewriting* in Ganzinger (1988), Zhang & Kapur (1988), and Bevers & Lewi (1990). While these authors use a finite set of equations for the context of their rewriting steps we generalize these contexts to negative equations and Def-literals by taking a formula Γ instead, in which the negative equations play the role of these authors' finite set of equations. The form of presentation we choose is that of sub-rules of the Transformation rule of Wirth & Becker (1995) as presented in section 3.4 of this paper. On the one hand this uniform presentation satisfies our design goal of a "homogeneous representation" as established in section 1.3 according to which it admits to use the power of our whole inference system inside contextual rewriting flexibly. On the other hand, however, this presentation does not exhibit the need for a special heuristic control for successful application of contextual rewriting in practice.

Since contextual rewriting is based on removal of trivial tautologies, we begin with that. Then, before one really starts with contextual rewriting one should remove redundant literals. After that, we will come to constant rewriting. Finally in this section, we treat features of contextual rewriting that apply rules and lemmas, and then close with a completeness result for contextual rewriting.

[19] Be reminded we require confluence of $\longrightarrow_{R,\emptyset}$ only, or of $\longrightarrow_{R,V_{SIG}}$ at most, but not of $\longrightarrow_{R,V}$

5.2 Decomposition and Tautology Removal

Let $m, n \in \mathbb{N}$ with $n \preceq m$; $\Gamma, \Delta, \Lambda_0, \ldots, \Lambda_{n-1} \in$ Form; $\aleph \in$ Weight; $t \in \mathcal{T}$.
Let $p_0, \ldots, p_{m-1} \in \mathcal{POS}(t)$ be mutually parallel and $u_0, \ldots, u_{m-1}, v_0, \ldots, v_{m-1} \in \mathcal{T}$.
Let L be a finite subset of 'Form', and H, G be finite subsets of 'SynCons':

=-Decompose:

$$\frac{(L\ \ ,H\ \ ,G \cup \{\ (\qquad \Gamma(t[\,p_j \leftarrow u_j \mid j \prec m\,] = t[\,p_j \leftarrow v_j \mid j \prec m\,])\Delta\ \ ,\aleph) \qquad \}\ )}{(L\ \ ,H\ \ ,G \cup \{\ ((u_i = v_i)\Lambda_i \Gamma(t[\,p_j \leftarrow u_j \mid j \prec m\,] = t[\,p_j \leftarrow v_j \mid j \prec m\,])\Delta\ \ ,\aleph) \mid i \prec n\ \}\ )}$$

if

- $\forall i \prec n$: $\Big(\ \Lambda_i$ is contained in $(u_{i-1}{\neq}v_{i-1})(u_{i-2}{\neq}v_{i-2})\ldots(u_0{\neq}v_0)\ \Big)$,
- $\forall i$: $\Big(\ n \preceq i \prec m \ \Rightarrow\ (u_i{\neq}v_i)$ is contained in $\Gamma\Delta\ \Big)$.

Def**-Decompose:**

$$\frac{(L\ \ ,H\ \ ,G \cup \{\ (\qquad \Gamma(\mathrm{Def}\, t[\,p_j \leftarrow v_j \mid j \prec m\,])\Delta\ \ ,\aleph) \qquad \}\ )}{(L\ \ ,H\ \ ,G \cup \{\ (\qquad (\mathrm{Def}\, v_i)\Lambda_i \Gamma(\mathrm{Def}\, t[\,p_j \leftarrow v_j \mid j \prec m\,])\Delta\ \ ,\aleph) \mid i \prec n\ \}\ )}$$

if

- $t \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$,
- $\forall i \prec n$: $\Big(\ \Lambda_i$ is contained in $\overline{(\mathrm{Def}\, v_{i-1})}\ \overline{(\mathrm{Def}\, v_{i-2})}\ldots\overline{(\mathrm{Def}\, v_0)}\ \Big)$,
- $\forall i$: $\Big(\ n \preceq i \prec m \ \Rightarrow\ \overline{(\mathrm{Def}\, v_i)}$ is contained in $\Gamma\Delta\ \Big)$.

≠-Tautology Remove:

$$\frac{(L\ \ ,H\ \ ,G \cup \{\ (\qquad \Gamma(u_0{\neq}u_1)\Delta\ \ ,\aleph) \qquad \}\ )}{(L\ \ ,H\ \ ,G \qquad)}$$

if

- $u_0, u_1 \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$,
- u_0, u_1 are clashing, and
- $\forall i \prec 2$: $\forall p \in \mathbb{F}\mathcal{POS}(u_i)$: $\forall((l, r), C) \in \mathrm{R}$: (u_i/p and l are clashing).

Lemma 84 *The =-Decompose rule is a safe sub-rule of the Transformation rule.*

Lemma 85 *The* Def-*Decompose rule is a safe sub-rule of the Transformation rule.*

Lemma 86 *If* K *is a sub-class of the class of constructor-minimal models of* R*, then the* $\neq$*-Tautology Remove rule is a safe sub-rule of the Transformation rule.*

In the special case of " $m=1$, $n=0$, $p_0=\emptyset$ " the =- and Def-Decompose rule just remove a goal that contains complementary literals. In former versions of this paper we had a Complement-Tautology Remove rule for this purpose.

The special case of " $m=n=1$, $p_0=\emptyset$ " of the =- and Def-Decompose rule is meaningless.

In the special case of " $m=n=0$ " the =-Decompose rule just removes a goal that contains an atom of the form $(t{=}t)$. In former versions of this paper had an =-Tautology Remove rule for this purpose.

In the special case of " $n=0$ " the Def-Decompose rule just removes a goal that contains an atom of the form $(\mathrm{Def}\, t[\, p_j \leftarrow v_j \mid j \prec m\,])$ where t is a constructor term and we can assume $(\mathrm{Def}\, v_j)$ to be true (for all $j \prec m$) because it is listed in the condition of this goal. In former versions of this paper had a Def-Tautology Remove rule for this purpose.

Note that we also have an $\neq$-Solve rule[20] that in combination with the Variable Add and Remove rules[21] is more general than the $\neq$-Tautology Remove rule.

In former versions of this paper we also had a $\overline{\text{Def}}$-Tautology Remove rule, which we have removed now because of its low importance and since the $\overline{\text{Def}}$-Solve rule[22] is more general.

[20] cf. section 7.3
[21] cf. section 6.2
[22] cf. section 7.3

5.3 Removal of Redundant Literals

Let $m \in \mathbb{N}$; $\Gamma, \Delta, \Pi, \Lambda \in \mathrm{Form}$; $\aleph \in \mathrm{Weight}$; $\lambda \in \mathcal{GENLIT}(\mathrm{sig}, \mathrm{V})$; $s \in \mathbb{S}$; $t, t' \in \mathcal{T}_{\mathrm{SIG},s}$.
Let $p_0, \ldots, p_{m-1} \in \mathcal{POS}(t)$ be mutually parallel and $u_0, \ldots, u_{m-1}, v_0, \ldots, v_{m-1} \in \mathcal{T}$.
Let L be a finite subset of 'Form', and H, G be finite subsets of 'SynCons':

Multiple-Literal Remove:

$$\frac{(L \ , H \ , G \cup \{ \ (\Gamma\lambda\Delta \ , \aleph) \ \} \ )}{(L \ , H \ , G \cup \{ \ (\Gamma\Delta \ , \aleph) \ \} \ )}$$

if λ is contained in $\Gamma\Delta$.

$\neq$-Literal Remove:

$$\frac{(L \ , H \ , G \cup \{ \ (\Gamma(t[p_j \leftarrow u_j \mid j \prec m] \neq t[p_j \leftarrow v_j \mid j \prec m])\Delta \ , \aleph) \ \} \ )}{(L \ , H \ , G \cup \{ \ (\Gamma\Delta \ , \aleph) \ \} \ )}$$

if $\forall i \prec m$: $(u_i \neq v_i)$ is contained in $\Gamma\Delta$.

$\overline{\mathrm{Def}}$-Literal Remove:

$$\frac{(L \ , H \ , G \cup \{ \ (\Gamma\overline{(\mathrm{Def}\, t[p_j \leftarrow v_j \mid j \prec m])}\Delta \ , \aleph) \ \} \ )}{(L \ , H \ , G \cup \{ \ (\Gamma\Delta \ , \aleph) \ \} \ )}$$

if $t \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$ and $\forall i \prec m$: $\overline{(\mathrm{Def}\, v_i)}$ is contained in $\Gamma\Delta$.

Applicative Literal Remove:

$$\frac{(L \ , H \ , G \cup \{ \ (\Gamma(\lambda\mu)\Delta \ , \aleph) \ \} \ )}{(L \ , H \ , G \cup \{ \ (\Gamma\Delta \ , \aleph) \ \} \ )}$$

if there are $\Pi\overline{\lambda}\Lambda \in L \cup \mathrm{form}[H \cup G]$ and $\mu \in \mathcal{GENSUB}(\mathrm{V}, \mathcal{T})$
such that all the following holds:

- $(\Pi\Lambda)\mu$ is contained in $\Gamma\Delta$,
- $\forall x \in \mathrm{GENDOM}(\mu)$: $\left(\ \overline{(\mathrm{Def}\, x\mu)} \text{ is contained in } \Gamma\Delta \ \right)$,
- $\left(\begin{array}{ll} & \mathrm{GENDOM}(\mu) \cap \mathcal{V}_<(\Pi\overline{\lambda}\Lambda) = \emptyset \\ \vee & \text{the induction ordering is semantic} \end{array} \right)$.

=-Literal Remove:

$$\frac{(L \ , H \ , G \cup \{ \ (\Gamma(t{=}t')\Delta \ , \aleph) \ \} \ )}{(L \ , H \ , G \cup \{ \ (\Gamma\Delta \ , \aleph) \ \} \ )}$$

if $(t{=}t')$ is linear and $\mathrm{NARROW}(t, t', \mathcal{V}(t, t')) = \emptyset$.

Def-**Literal Remove:**

$$\frac{(L \quad , H \quad , G \cup \{\ (\Gamma(\mathrm{Def}\, t)\Delta \quad , \aleph)\ \}\ )}{(L \quad , H \quad , G \cup \{\ (\qquad\quad \Gamma\Delta \quad , \aleph)\ \}\ )}$$

if t is linear and for some $x \in \mathrm{V}_{\mathrm{CONS},s} \backslash \mathcal{V}(t)$ we have $\mathrm{NARROW}(x, t, \mathcal{V}(x,t)) = \emptyset$.

Corollary 87 *The Multiple-, $\neq$-, $\overline{\mathrm{Def}}$-, $=$-,* Def-*, and Applicative Literal Remove rules are sub-rules of the Transformation rule.*

Corollary 88 *The Multiple-, $\neq$-, and $\overline{\mathrm{Def}}$-Literal Remove rules are safe.*

Lemma 89 *The Applicative Literal Remove rule is safe.*

Lemma 90 *Assume either that the D- or E-case holds or otherwise that, for the literal $(t{=}t')$ of the $=$-Literal Remove rule, $t, t' \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$ and* K *is a sub-class of the class of constructor-minimal models of* R*. Now:*
The $=$-Literal Remove rule is safe.

Lemma 91 *Assume that the D- or E-case holds. Now:*
The Def-*Literal Remove rule is safe.*

The Literal Remove rules should be applied before one starts more complex contextual rewriting since they tend to reduce the search space.[23]

The Multiple-Literal Remove rule is in a certain sense complementary to the case of " $m{=}1$, $n{=}0$, $p_0{=}\emptyset$ " of the $=$- or Def-Decompose rules[24].

The $\neq$- and $\overline{\mathrm{Def}}$-Literal Remove rule are in a certain sense complementary to the special case of " $n{=}0$ " of the $=$- and Def-Decompose rule, resp..

The Applicative-Literal Remove rule also has a complementary rule that removes a goal, namely the Lemma Apply rule[25] for the special case of " $m{=}0$ ". Note that the Multiple-Literal Remove rule is an Applicative Literal Remove with the lemma $\overline{\lambda}\lambda$, that the $\neq$-Literal Remove rule is an Applicative Literal Remove with the lemma $(x_0{\neq}y_0)\ldots(x_{m-1}{\neq}y_{m-1})(t[\, p_j \leftarrow x_j \mid j \prec m\,] \neq t[\, p_j \leftarrow y_j \mid j \prec m\,])$ for $x_0, y_0, \ldots, x_{m-1}, y_{m-1} \in \mathrm{V}_{\mathrm{SIG}}$, and that the $\overline{\mathrm{Def}}$-Literal Remove rule is an Applicative Literal Remove with the lemma $(\mathrm{Def}\, t[\, p_i \leftarrow x_i \mid i \prec m\,])$ for $x_0, \ldots, x_{m-1} \in \mathrm{V}_{\mathrm{CONS}}$.

The $=$- and Def-Literal Remove rule are in a certain sense complementary to the $\neq$-Solve and $\overline{\mathrm{Def}}$-Solve rules[26], resp.. Note that in case of $t \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$ the Def-Literal Remove rule is never applicable (contrary to the Def-Decompose rule for " $m{=}n{=}0$ " which should be immediately applied in this case). This makes a second case of Lemma 91 according to Lemma 90 superfluous.

[23] In some situations, however, some redundant literals (typically $\overline{\mathrm{Def}}$-literals) will be generated again by a subsequent Lemma Apply etc.

[24] cf. section 5.2

[25] cf. section 5.5

[26] cf. section 7.3

5.4 Constant Rewriting

An essential feature of contextual rewriting is the following Constant Rewrite rule. The word "constant" was chosen to remind the reader that this rule does not allow to instantiate the variables of its "rewrite rule" for the rewrite step, i.e. the variables have to be treated like constants. While the Constant Rewrite rule can implement Knuth-Bendix completion for unconditional ground equations (cf. Zhang (1993)), in accordance with our design goal of section 1.3 to include "inference rules for most elementary proof steps" we present the Constant Rewrite rule in the following essential elementary form because long sequences of consecutive Constant Rewrite steps are hardly necessary in practice.

Let $m, n \in \mathbb{N}$ with $n \preceq m$; $\lambda_0, \ldots, \lambda_{m-1}, \lambda'_0, \ldots, \lambda'_{m-1}, \lambda'' \in \mathcal{GENLIT}(\mathrm{sig}, \mathrm{V})$; $\aleph, \aleph' \in$ Weight. Let L be a finite subset of 'Form', and H, G be finite subsets of 'SynCons':

Constant Rewrite:
$$\frac{(L \ , H \ , G \cup \{\ (\ \lambda_0 \ldots \lambda_{n-1} \lambda'' \lambda_n \ldots \lambda_{m-1} \ , \ \aleph \ )\ \}\ )}{(L \ , H \ , G \cup \{\ (\ \lambda'_0 \ldots \lambda'_{n-1} \lambda'' \lambda'_n \ldots \lambda'_{m-1} \ , \ \aleph' \ )\ \}\ )}$$

if there are some l, r such that all the following holds:

- λ'' equals $(l \neq r)$.
- $\left(\begin{array}{ll} & \aleph' = \aleph \\ \vee & \left(\begin{array}{ll} & \aleph \text{ rewrites to } \aleph' \text{ with } l = r \\ \wedge & \text{the induction ordering is semantic} \end{array} \right) \end{array} \right)$
- $\forall i \prec m$: $\left(\begin{array}{ll} & \lambda'_i = \lambda_i \\ \vee & \left(\begin{array}{ll} & \lambda_i \text{ rewrites to } \lambda'_i \text{ with } l = r \\ \wedge & \left(\begin{array}{ll} & \lambda_i, \lambda'_i \in \mathcal{LIT}(\mathrm{sig}, \mathrm{V}) \\ \vee & \text{the induction ordering is semantic} \end{array} \right) \end{array} \right) \end{array} \right)$

Lemma 92 *The Constant Rewrite rule is a safe sub-rule of the Transformation rule.*

The rewriting of literals and weights is defined in definitions 5 and 74.

Note that we could rewrite also λ'' with $l = r$. Then the Constant Rewrite rule would still be a sub-rule of the Transformation rule, but not safe anymore: Consider $m := 0$; $\lambda'' := (\mathsf{true} \neq \mathsf{false})$. Then $(\mathsf{true} \neq \mathsf{false})$ rewrites to $(\mathsf{false} \neq \mathsf{false})$ with the rule $\mathsf{true}{=}\mathsf{false}$, but, while $(\mathsf{true} \neq \mathsf{false})$ is likely to be inductively valid, $(\mathsf{false} \neq \mathsf{false})$ is not.

Note that rewriting the weight or a '<'-literal, which is only possible if the induction ordering is semantic, is of questionable importance: One usually can delay the rewriting of the weight until it has generated a '<'-literal and the rewriting of a '<'-literal until it has been transformed into an '='-literal like $\mathsf{less}(t, t') = \mathsf{true}$ by a special inference rule for '='. Therefore we have omitted this possibility in the Lemma and Hypothesis Rewrite rules for the sake of readability. Nevertheless, we consider the rewriting of the weight to be conceptually interesting, cf. Example 126.

5.5 Application of Rules and Lemmas

In this section we treat inference rules that allow us to apply a defining rule from R or a lemma Π (instantiated via a generalized substitution μ) to a goal Γ.

Let $m \in \mathbb{N}$; $\Gamma, \Pi, \Lambda_0, \ldots, \Lambda_{m-1} \in \mathrm{Form}$; $\lambda_0, \ldots, \lambda_{m-1} \in \mathcal{GENLIT}(\mathrm{sig}, \mathrm{V})$; $\aleph \in \mathrm{Weight}$.
Let L be a finite subset of 'Form', and H, G be finite subsets of 'SynCons':

Lemma Apply:
$$\frac{(L \cup \{\Pi\}, H, G \cup \{(\Gamma, \aleph)\})}{(L \cup \{\Pi\}, H, G \cup \{(\overline{\lambda_i}\Lambda_i\Gamma, \aleph) \mid i \prec m\})}$$

if there is some $\mu \in \mathcal{GENSUB}(\mathrm{V}, \mathcal{T})$ such that all the following holds:

- $\forall i \prec m$: $\Big($ Λ_i is contained in $\lambda_{i-1}\lambda_{i-2}\ldots\lambda_0$ $\Big)$,
- $\Pi\mu$ is contained in $\lambda_{m-1}\lambda_{m-2}\ldots\lambda_0\Gamma$,
- $\forall x \in \mathrm{GENDOM}(\mu)$: $\Big($ $\overline{(\mathrm{Def}\, x\mu)}$ is contained in $\lambda_{m-1}\lambda_{m-2}\ldots\lambda_0\Gamma$ $\Big)$,
- $\left(\begin{array}{ll} & \mathrm{GENDOM}(\mu) \cap \mathcal{V}_<(\Pi) = \emptyset \\ \vee & \text{the induction ordering is semantic} \end{array}\right)$.

Lemma 93 *The Lemma Apply rule is a safe sub-rule of the Transformation rule.*

The Lemma Apply is the more attractive the smaller the variable m is:

E.g. for " $m=0$ ", an application of the rule means just to remove the goal $(\Gamma, \aleph)$ by application of a lemma that subsumes it.

For " $m=1$ ", it means to add a new literal to Γ. This is attractive if this new literal is useful for the proof, which especially is the case if a rule becomes applicable that can remove the new goal,[27] or if the literal is of the form $(l \neq r)$ and l is a sub-term occurring in Γ, in which case we can use the literal for a Constant Rewrite. The latter case includes an application of a defining rule from R, since we can always suppose $\mathrm{R} \subseteq L$.[28] This rewriting case is so important in practice that, after some more motivation, we will combine a Lemma Apply with a succeeding Constant Rewrite into a single rule.

An " $m \succ 1$ " results from conditions of the applied defining rule or lemma (or else from definedness conditions for the generalized variables) that have to be added to the goal because they are not already present. Therefore in the Lemma Apply rule we intend

$$\forall i \prec m\colon \left(\begin{array}{ll} & \lambda_i \text{ is not contained in } \Gamma \\ \wedge & \left(\begin{array}{ll} & (\lambda_i \text{ is contained in } \Pi\mu) \\ \vee & \exists x \in \mathrm{GENDOM}(\mu)\colon\ \lambda_i = \overline{(\mathrm{Def}\, x\mu)} \end{array} \right) \end{array} \right).$$

Thus, the application of Π is lazy w.r.t. to the verification of the "$\overline{\lambda_i}$" which has to be done later, possibly by contextual rewriting again. In this sense the application of rules or lemmas is a recursive feature of contextual rewriting.

The laziness mentioned above is not a must. An eager tactic could, e.g., after applying the Lemma Apply rule, verify the sub-goals expressing the fulfilledness of the conditions of the rule or lemma first, and then, if this was not successful, undo the application of the Apply rule. Thus (in accordance with our design goal of a "homogeneous representation" of section 1.3), the homogeneous formulation of the satisfaction of the conditions of an Apply rule as sub-goals allows us to use the full power of our prover to verify these conditions as well as to be lazy or not, leaving this choice to our tactical consideration.

---

[27] like the Decompose or Tautology Remove rules of section 5.2 or Lemma (or Hypothesis, cf. section 7.2) Apply. Note, however, that if it is a Decompose rule that becomes applicable for the special case of " $m=1$, $n=0$, $p_0=\emptyset$ ", then it would have been possible to remove the goal at once by choosing " $m=0$ " for the original Lemma Apply instead of " $m=1$ ".

[28] according to Definition 13 by the Acquisition rule of Wirth & Becker (1995) as presented in section 3.4 of this paper

Example 94 *(continuing examples 9 and 10)*
Assume that we want to transform the goal[29]
(94.1) $\mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) = \mathsf{true}$
into a more concise form, using the rules of Example 9 as lemmas:
*(*mbp*1)* $\mathsf{mbp}(x, \mathsf{nil}) = \mathsf{false}$
*(*mbp*2)* $\mathsf{mbp}(x, \mathsf{cons}(y, l)) = \mathsf{true}, \quad x \neq y$
*(*mbp*3)* $\mathsf{mbp}(x, \mathsf{cons}(y, l)) = \mathsf{mbp}(x, l), \quad x = y$
*Now both (*mbp*2) and (*mbp*3) qualify for a Lemma Apply, since they both have equality literals matching the first term of our goal via* $\mu := \{ x \mapsto \mathsf{s}(x),\ y \mapsto \mathsf{s}(y),\ l \mapsto \mathsf{nil} \}$. *Since this match extends to the whole literal for the case of (*mbp*2), the application of (*mbp*2) is more attractive since it admits a smaller 'm' in the Lemma Apply. Therefore we apply (*mbp*2) first and then (*mbp*3) afterwards. Doing it the other way round produces the same result but with a bigger proof tree and the difficulty whether to instantiate* Λ_1 *to* $\emptyset$ *or* λ_0 *where only the latter choice brings us into the right direction.*
*The Lemma Apply of (*mbp*2) via* $m := 1; \quad \lambda_0 := \mathsf{s}(x) \neq \mathsf{s}(y)$ *results in the new goal*
(94.1.1) $\mathsf{s}(x) = \mathsf{s}(y), \quad \mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) = \mathsf{true}$
*which then is transformed with a Lemma Apply of (*mbp*3) via* $m := 1; \quad \lambda_0 := \mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) = \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil})$; *yielding*
(94.1.1.1)
$\mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) \neq \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}), \quad \mathsf{s}(x) = \mathsf{s}(y), \quad \mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) = \mathsf{true}$
which by Constant Rewrite can be transformed into
(94.1.1.1.1) $\mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) \neq \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}), \quad \mathsf{s}(x) = \mathsf{s}(y), \quad \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}) = \mathsf{true}$
*which in turn can be transformed by Lemma Apply of (*mbp*1) into*
(94.1^5*)* $\mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}) \neq \mathsf{false},$
$\mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) \neq \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}), \quad \mathsf{s}(x) = \mathsf{s}(y), \quad \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}) = \mathsf{true}$
then by Constant Rewrite into
(94.1^6*)* $\mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}) \neq \mathsf{false},$
$\mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) \neq \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}), \quad \mathsf{s}(x) = \mathsf{s}(y), \quad \mathsf{false} = \mathsf{true}$
then by =-Literal Remove into
(94.1^7*)* $\mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}) \neq \mathsf{false}, \quad \mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) \neq \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}), \quad \mathsf{s}(x) = \mathsf{s}(y)$
then by =-Decompose of $\mathsf{s}(x) = \mathsf{s}(y)$ *into*
(94.1^8*)* $x = y, \quad \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}) \neq \mathsf{false}, \quad \mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) \neq \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}), \quad \mathsf{s}(x) = \mathsf{s}(y)$
then by Applicative Literal Remove with (Inject s*) of Example 10 into*
(94.1^9*)* $x = y, \quad \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}) \neq \mathsf{false}, \quad \mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) \neq \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil})$
*and finally by Applicative Literal Remove with (*mbp*1) into*
(94.1^{10}*)* $x = y, \quad \mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) \neq \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil})$

[29] Note that we omit the weights of the goals here and in the other examples of this section

All in all, we have transformed

(94.1) $\mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) = \mathsf{true}$

into

(94.1[10]) $x = y, \;\; \mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) \neq \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil})$

By application of the Context Remove rule (cf. section 7.4), which, however, is *unsafe* in general, we can transform this into

$$x = y$$

which is much more concise.

Note that we cannot remove the literal

$$\mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) \neq \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil})$$

from (94.1[10]) by an Applicative Literal Remove with (mbp3) because this would require the presence of the literal $\mathsf{s}(x) = \mathsf{s}(y)$ which we have removed before. Of course, removing the literals in a different order could have led us to the desired concise goal $x = y$. But this all seems much too complicated for such a simple simplification problem. In general, throwing away the $\neq$-literals generated by Lemma Apply and then used for intermediate Constant Rewrite steps is unsafe, however. To see this, consider the following example:

Example 95 *(continuing examples 9 and 10)*
Let us transform the goal

(95.1) $\mathsf{s}(x) \neq \mathsf{s}(y), \;\; \Gamma$

by a Lemma Apply of (Inject s*) of Example 10 into*

(95.1.1) $x \neq y, \;\; \mathsf{s}(x) \neq \mathsf{s}(y), \;\; \Gamma$

and then by Constant Rewrite to

(95.1.1.1) $x \neq y, \;\; \mathsf{s}(y) \neq \mathsf{s}(y), \;\; \Gamma$

and finally by $\neq$-Literal Remove to

(95.1.1.1.1) $x \neq y, \;\; \Gamma$

If we now removed the literal generated by Lemma Apply for the intermediate Constant Rewrite (i.e. $x \neq y$), we would get the formula Γ. The transformation of (95.1) to Γ, however, is not safe in general.[30]

The reason why we are not allowed to safely remove the intermediate $\neq$-literal here is that we have applied it to the literal which was needed to invent it in the Lemma Apply step.

Instead of keeping book on which literals are used for intermediate Constant Rewrite, on which literals their invention depends, and to which literals they have been applied, and instead of searching for the right order in which to remove them, we prefer not to add these intermediate literals at all:

[30] To see this, consider $\Gamma := x = y$. Note, however, that in this case we had better remove the original goal directly with the lemma.

Let $m \in \mathbb{N}$; $\Gamma, \Delta, \Pi, \Lambda_0, \ldots, \Lambda_m \in \text{Form}$; $\lambda_0, \ldots, \lambda_m \in \mathcal{GENLIT}(\text{sig}, \text{V})$; $\aleph \in \text{Weight}$; $\lambda'' \in \mathcal{LIT}(\text{sig}, \text{V})$; $p \in \mathcal{POS}(\lambda'') \backslash \{\emptyset\}$.
Let L be a finite subset of 'Form', and H, G be finite subsets of 'SynCons':

Lemma Rewrite:

$$\frac{(L \cup \{\Pi\}, H, G \cup \{(\Gamma\lambda''\Delta, \aleph)\})}{\begin{array}{l}(L \cup \{\Pi\}, H, G \cup \{(\overline{\lambda_i}\,\Lambda_i\Gamma\lambda''\Delta, \aleph) \mid i \prec m\} \\ \qquad\qquad \cup \{(\Lambda_m\Gamma\lambda''[p \leftarrow r]\Delta, \aleph)\})\end{array}}$$

if there is some $\mu \in \mathcal{GENSUB}(\text{V}, \mathcal{T})$ such that all the following holds:

- λ_m equals $(l{=}r)$ for $l := \lambda''/p$,
- $\forall i \preceq m$: $\Big($ Λ_i is contained in $\lambda_{i-1}\lambda_{i-2}\ldots\lambda_0$ $\Big)$,
- $\Pi\mu$ is contained in $\lambda_m\Lambda_m\Gamma\Delta$,
- $\forall x \in \text{GENDOM}(\mu)$: $\Big($ $\overline{(\text{Def}\, x\mu)}$ is contained in $\lambda_{m-1}\lambda_{m-2}\ldots\lambda_0\Gamma\Delta$ $\Big)$,
- $\left(\begin{array}{ll} & \text{GENDOM}(\mu) \cap \mathcal{V}_<(\Pi) = \emptyset \\ \vee & \text{the induction ordering is semantic}\end{array}\right)$.

Lemma 96 *The Lemma Rewrite rule is a safe sub-rule of the Transformation rule.*

Note that in the Lemma Rewrite rule we intend that Λ_m equals $\lambda_{m-1}\lambda_{m-2}\ldots\lambda_0$.

Note that weakening the third condition of the Lemma Rewrite rule to

- $\Pi\mu$ is contained in $\lambda_m \Lambda_m \Gamma \lambda'' \Delta$,

makes the Lemma Rewrite rule unsafe. This can be easily seen in the following example:

Example 97 *(continuing Example 95)*
If we weakened the third condition of the Lemma Rewrite rule as indicated above, we could use a Lemma Rewrite with $m := 0$; $\lambda_0 := x = y$; $\lambda'' := \mathsf{s}(x) \neq \mathsf{s}(y)$; $p := 1\,1$; *to transform the goal*
(97.1) $\mathsf{s}(x) \neq \mathsf{s}(y),\ \Gamma$
with the lemma
(Inject s*)* $\mathsf{s}(x) \neq \mathsf{s}(y),\ x = y$
of Example 10 into the goal
(97.1.1) $\mathsf{s}(y) \neq \mathsf{s}(y),\ \Gamma$
which can safely be transformed by $\neq$*-Literal Remove into*
(97.1.1.1) Γ
The transformation of (97.1) into (97.1.1.1), however, is unsafe in general as already noted in Example 95.

Furthermore, note that instead of the result of the Lemma Rewrite
(G0) $\Lambda_m \Gamma \lambda''[\, p \leftarrow r \,]\Delta$
a Lemma Apply with subsequent Constant Rewrite can safely yield
(G1) $(l \neq r) \Gamma \lambda''[\, p \leftarrow r \,]\Delta$
A third possibility of a safe transformation is to yield the following instead:
(G2) $\lambda'' \Gamma \lambda''[\, p \leftarrow r \,]\Delta$
Thus, to achieve safety of the rewrite step in $\Gamma\lambda''\Delta$ we have to add to $\Gamma\lambda''[\, p \leftarrow r \,]\Delta$ one of Λ_m (as Lemma Rewrite does), $(l \neq r)$ (as Lemma Apply does), or λ''.
(G2) (in general strictly) implies
(G1') $(l \neq r) \lambda'' \Gamma \lambda''[\, p \leftarrow r \,]\Delta$
which is equivalent to (G1). (G1) again (in general strictly) implies
(G0') $\Lambda_m (l \neq r) \Gamma \lambda''[\, p \leftarrow r \,]\Delta$
which by $\Pi\mu$ is equivalent to (G0). Thus (G2) is possibly stronger and thus more difficult to prove than the result of the Lemma Apply, which again is possibly stronger and more difficult to prove than the result of Lemma Rewrite. Furthermore, (G2) is less concise and practically useful than the result of Lemma Apply, which again may be less concise (especially for the typical special case of $m = 0$ where $\Lambda_m = \emptyset$) and practically useful than the result of Lemma Rewrite.

Now we reconsider Example 94 to see whether the additional Lemma Rewrite rule admits a simpler and more appropriate treatment of the given simplification problem:

Example 98 *(continuing Example 94)*
Remember that we wanted to transform the goal
(98.1) $\mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) = \mathsf{true}$
into a more concise form, using the rules of Example 9 as lemmas.
*Again, both (*mbp*2) (with* $m := 1$*) and (*mbp*3) (with* $m := 2$*) qualify for a Lemma Apply since they both have equality literals matching the first term of our goal. Additionally, both (*mbp*2) and (*mbp*3) qualify for a Lemma Rewrite with* $m := 1$*. Again, the Lemma Apply of (*mbp*2) is the most attractive when we prefer an Apply to a Rewrite with equal 'm' because the Rewrite produces an additional goal.*
*The Lemma Apply of (*mbp*2) results (as before) in the goal*
(98.1.1) $\mathsf{s}(x) = \mathsf{s}(y),\ \ \mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) = \mathsf{true}$
*Now (*mbp*3) qualifies for a Lemma Apply with* $m := 1$ *and for a Lemma Rewrite with* $m := 0$*. Since we consider a Rewrite with a strictly smaller 'm' to be more attractive than an Apply, we choose the Rewrite here.*
*This Lemma Rewrite with (*mbp*3) via* $m := 0;\quad \lambda_0 := \mathsf{mbp}(\mathsf{s}(x), \mathsf{cons}(\mathsf{s}(y), \mathsf{nil})) = \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil})$*; transforms (98.1.1) into*
(98.1.1.1) $\mathsf{s}(x) = \mathsf{s}(y),\ \ \mathsf{mbp}(\mathsf{s}(x), \mathsf{nil}) = \mathsf{true}$
*which in turn can be transformed by Lemma Rewrite with (*mbp*1) into*
(98.1.1.1.1) $\mathsf{s}(x) = \mathsf{s}(y),\ \ \mathsf{false} = \mathsf{true}$
then by =-Literal Remove into
(98.1^5*)* $\mathsf{s}(x) = \mathsf{s}(y)$
then by =-Decompose into
(98.1^6*)* $x = y,\ \ \mathsf{s}(x) = \mathsf{s}(y)$
and finally by Applicative Literal Remove with (Inject s*) of Example 10 into*
(98.1^7*)* $x = y$

Thus we really have got rid of the intermediate $\neq$-literals without running the risk of an unsafe inference step, and obtained a more concise result. Moreover, the simplification took only six instead of nine inference steps and the intermediate goals are much simpler.

5.6 Completeness Result

Theorem 99 (Deductive Completeness)

Let F *denote the set of those formulas from* Form *that do not contain '*$<$*'-literals. For a finite set* $L \subseteq F$ *let '* $\widehat{L}$ *' denote a conjunction containing a universal quantifier closed version of each formula of* L*.*

The below subsystem of our inference system for contextual rewriting is deductively complete in the following sense: If some formula $\Gamma \in F$ *is valid in all those* sig/cons*-algebras in which all formulas from a finite set* $L \subseteq F$ *of lemmas are valid, i.e. if*

$$\widehat{L} \models \Gamma,$$

then we have

$$(L, \emptyset, \{(\Gamma, \aleph)\}) \vdash^* (L, \emptyset, \emptyset)$$

for any $\aleph \in$ Weight*.*

The complete subsystem contains the $=$*-Decompose (with the restriction of* $m = n = 0$*), and the* Def*-Decompose rule (with the restriction of* $n = 0$*). It also contains Constant Rewrite (with the restriction that* $\aleph = \aleph'$*, that, for all* $i \prec m$*,* $\lambda_i = \lambda'_i$ *or the literal* λ_i *is positive, and that only one occurrence of* l *is replaced with* r*). Finally, it also contains Lemma Apply (with the restrictions* $\Pi\mu = \lambda_{m-1} \ldots \lambda_0$*,* $\forall x \in \mathrm{GENDOM}(\mu)$*:* $(\overline{(\mathrm{Def}\, x\mu)}$ *is contained in* Γ*),* $\forall i \prec m$*:* $\Lambda_i = \emptyset$*, and that only Lemma Applys have been applied before).*

If each formula from L *contains a positive equation literal, then instead of the Lemma Apply we can also take the Lemma Rewrite (with the restrictions that the literal* λ'' *is positive,* $(\ m = 0 \ \vee$ $\Pi\mu$ *equals* $\lambda_m \Lambda_m\ )$*,* $\forall x \in \mathrm{GENDOM}(\mu)$*:* $(\overline{(\mathrm{Def}\, x\mu)}$ *is contained in* $\Gamma\Delta$*),* $\forall i \prec m$*:* $\Lambda_i = \emptyset$*,* $\Lambda_m = \lambda_{m-1} \ldots \lambda_0\ )$*.*

6 More Non-Inductive Rules

6.1 Adding Context

While the Apply and Rewrite rules are appropriate for automatic addition of context (i.e. of new literals) to a formula, a more intelligent tactic may want an explicit case analysis as a single rule instead of spreading the case analysis over several Apply or Rewrite rule applications driven by the application of defining rules, lemmas, or hypotheses. Since Gentzen's Cut Elimination is not possible for inductive proofs, such a rule may be really necessary for non-trivial examples. Furthermore, it is well-suited for user interaction in more difficult proofs.

Let $m \in \mathbb{N}$; $\Gamma, \Lambda_0, \ldots, \Lambda_{m+1} \in \text{Form}$; $\lambda_0, \ldots, \lambda_m \in \mathcal{GENLIT}(\text{sig}, \text{V})$; $\aleph \in \text{Weight}$.
Let L be a finite subset of 'Form', and H, G be finite subsets of 'SynCons':

Context Add:
$$\frac{(L \;, H \;, G \cup \{\, (\quad \Gamma,\ \aleph) \quad \})}{\begin{array}{l}(L \;, H \;, G \cup \{\, (\ \overline{\lambda_i}\,\Lambda_i\Gamma,\ \aleph) \mid i \preceq m \,\} \\ \qquad\qquad \cup \{\, (\ \Lambda_{m+1}\Gamma,\ \aleph) \quad \})\end{array}}$$

if $\forall i \preceq m{+}1$: $\Big(\ \Lambda_i$ is contained in $\lambda_{i-1}\lambda_{i-2}\ldots\lambda_0\ \Big)$.

Corollary 100 *The Context Add rule is a safe sub-rule of the Transformation rule.*

Note that our Context Add is similar to the complete case analysis used in mathematical (meta-level) proofs: There one puts up some cases λ_i and does the proof of Γ for each of these single cases, i.e., for each $i \preceq m$, one shows that λ_i implies Γ, which means to show $\overline{\lambda_i}\Gamma$. For that proof method to be sound one has to show that the case analysis is complete, i.e. that (under the assumptions of the state of the proof Γ) the logical disjunction of the cases holds, which means to show $\lambda_m\lambda_{m-1}\ldots\lambda_0\Gamma$, for which $\Lambda_{m+1}\Gamma$ is sufficient.

Indeed, in the Context Add rule we intend Λ_{m+1} to be $\lambda_m\lambda_{m-1}\ldots\lambda_0$. Especially the case of Λ_{m+1} being empty is not recommended because it just provides us with new goals that are useless since the logically stronger old goal is still present.

6.2 Removing and Adding Variables

Let $\Gamma, \Delta \in \mathrm{Form}$; $s \in \mathbb{S}$; $t \in \mathcal{T}_{\mathrm{SIG},s}$; $\aleph \in \mathrm{Weight}$;
$p_0, \ldots, p_{n-1} \in \mathcal{POS}(t)$ mutually parallel; $v_0, \ldots, v_{n-1} \in \mathcal{T} \backslash \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$.
Let L be a finite subset of 'Form', and H, G be finite subsets of 'SynCons':

SIG-**Variable Remove:**
$$\frac{(L \;, H \;, G \cup \{\, (\quad \Gamma(x{\neq}t)\Delta \;, \aleph) \,\} \;)}{(L \;, H \;, G \cup \{\, (\quad \Gamma\Delta \;, \aleph) \,\} \;)}$$

if $x \in \mathrm{V}_{\mathrm{SIG}} \backslash \mathcal{V}(t, \Gamma, \Delta)$.

CONS-**Variable Remove:**
$$\frac{(L \;, H \;, G \cup \{\, (\quad \Gamma(x{\neq}t)\Delta \;, \aleph) \,\} \;)}{(L \;, H \;, G \cup \{\, (\quad \Gamma\overline{(\mathrm{Def}\, t)}\Delta \;, \aleph) \,\} \;)}$$

if $x \in \mathrm{V}_{\mathrm{CONS}} \backslash \mathcal{V}(t, \Gamma, \Delta)$.

SIG-**Variable Add:**
$$\frac{(L \;, H \;, G \cup \{\, (\quad \Gamma \;, \aleph) \,\} \;)}{(L \;, H \;, G \cup \{\, (\quad (x{\neq}t)\Gamma \;, \aleph) \,\} \;)}$$

if $x \in \mathrm{V}_{\mathrm{SIG},s} \backslash \mathcal{V}(t, \Gamma, \aleph)$.

CONS-**Variable Add:**
$$\frac{(L \;, H \;, G \cup \{\, (\quad \Gamma \;, \aleph) \,\} \;)}{(L \;, H \;, G \cup \{\, ((x{\neq}t[\, p_i \leftarrow v_i \mid i \prec n \,])\Gamma \;, \aleph) \,\} \;)}$$

if $x \in \mathrm{V}_{\mathrm{CONS},s} \backslash \mathcal{V}(t[\, p_i \leftarrow v_i \mid i \prec n \,], \Gamma, \aleph)$ and
$t \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$ and $\forall i \prec n$: $\overline{(\mathrm{Def}\, v_i)}$ is contained in Γ.

Lemma 101
The SIG-*Variable Remove rule is a safe sub-rule of the Transformation rule.*

Lemma 102
The CONS-*Variable Remove rule is a safe sub-rule of the Transformation rule.*

Lemma 103
The SIG-*Variable Add rule is a safe sub-rule of the Transformation rule.*

Lemma 104
The CONS-*Variable Add rule is a safe sub-rule of the Transformation rule.*

Note that the Variable Add rules (choosing t to be a fresh variable) enable a subsequent linearization of literals as may be required before a Solve rule (cf. section 7.3) can be applied. Moreover, as shown in the following example, the CONS-Variable Add rule enables a subsequent case distinction on $t[\, p_i \leftarrow v_i \mid i \prec n \,]$ via application of a covering set of substitutions to x, cf. the Substitution Add rule of section 7.1.

Example 105 *(continuing examples 9 and 10)*

Let us show the lemma (Del swatch^3*) of Example 10, omitting the weights of the goals:*

(105.1) $\overline{\mathrm{Def}\,\mathsf{swatch}(0)},\ \mathsf{swatch}(\mathsf{swatch}(\mathsf{swatch}(x))) = \mathsf{swatch}(x)$

A Substitution Add yields:

(105.1.1) $\overline{\mathrm{Def}\,\mathsf{swatch}(0)},\ \mathsf{swatch}(\mathsf{swatch}(\mathsf{swatch}(0))) = \mathsf{swatch}(0)$

(105.1.2) $\overline{\mathrm{Def}\,\mathsf{swatch}(0)},\ \mathsf{swatch}(\mathsf{swatch}(\mathsf{swatch}(\mathsf{s}(x)))) = \mathsf{swatch}(\mathsf{s}(x))$

A CONS*-Variable Add with* $n := 1;\quad p_0 := \emptyset;\quad v_0 := \mathsf{swatch}(0);$ *transforms (105.1.1) into:*

(105.1.1.1) $y \neq \mathsf{swatch}(0),\ \overline{\mathrm{Def}\,\mathsf{swatch}(0)},\ \mathsf{swatch}(\mathsf{swatch}(\mathsf{swatch}(0))) = \mathsf{swatch}(0)$

A Substitution Add yields:

(105.1.1.1.1) $0 \neq \mathsf{swatch}(0),\ \overline{\mathrm{Def}\,\mathsf{swatch}(0)},\ \mathsf{swatch}(\mathsf{swatch}(\mathsf{swatch}(0))) = \mathsf{swatch}(0)$

(105.1.1.1.2) $\mathsf{s}(y) \neq \mathsf{swatch}(0),\ \overline{\mathrm{Def}\,\mathsf{swatch}(0)},\ \mathsf{swatch}(\mathsf{swatch}(\mathsf{swatch}(0))) = \mathsf{swatch}(0)$

Constant Rewrite with the first literal transforms (105.1.1.1.1) first into

(105.1^5*)* $0 \neq \mathsf{swatch}(0),\ \overline{\mathrm{Def}\,\mathsf{swatch}(0)},\ \mathsf{swatch}(\mathsf{swatch}(0)) = \mathsf{swatch}(0)$

and then into

(105.1^6*)* $0 \neq \mathsf{swatch}(0),\ \overline{\mathrm{Def}\,\mathsf{swatch}(0)},\ \mathsf{swatch}(0) = \mathsf{swatch}(0)$

which is then removed by an =-Decompose.

Moreover, Constant Rewrite with the first literal transforms (105.1.1.1.2) into

(105.1.1.1.2.1) $\mathsf{s}(y) \neq \mathsf{swatch}(0),\ \overline{\mathrm{Def}\,\mathsf{swatch}(0)},\ \mathsf{swatch}(\mathsf{swatch}(\mathsf{s}(y))) = \mathsf{s}(y)$

which is then removed by a Lemma Apply with the lemma (Del swatch swatch*) of Example 10.*

Finally, we complete the proof by showing (105.1.2). A Lemma Rewrite with the lemma (Del swatch swatch*) of Example 10 transforms (105.1.2) into*

(105.1.2.1) $\overline{\mathrm{Def}\,\mathsf{swatch}(0)},\ \mathsf{swatch}(\mathsf{s}(x)) = \mathsf{swatch}(\mathsf{s}(x))$

which is then removed by an =-Decompose.

Finally, it should be mentioned that a Lemma Apply with the lemma (Del swatch^3) (shown in the example above to be type-C, -D', etc., valid (cf. section 9)) and then a Lemma Apply with the type-D' valid lemma (for $X \in \mathrm{V}_{\mathrm{SIG},\mathsf{nat}}$)

$$\mathrm{Def}\,X$$

proves that

$$\mathsf{swatch}(\mathsf{swatch}(\mathsf{swatch}(x))) = \mathsf{swatch}(x)$$

is type-D' valid. It is not valid, however, for the types E, D, C, etc..

7 The Inductive Rules

7.1 Covering Sets of Substitutions

We are now going to explain why we need a further case analysis rule. Even our rather general form of contextual rewriting is not sufficient for inductive theorem proving. This will become obvious by the following, where x is assumed to be a constructor variable of the sort nat and where leq is assumed to be defined according to Example 9. The inductive instances that are described by the goal

$$\mathsf{leq}(x,\mathsf{0}) = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$$

can all be transformed by Lemma Rewrites with rules from R_9, but the goal itself does not have to be transformable. Therefore we first have to transform this goal into the following two goals:

$$\mathsf{leq}(\mathsf{0},\mathsf{0}) = \mathsf{true},\ \ \Gamma\sigma_0\ \ ;\ \ \aleph\sigma_0$$
$$\mathsf{leq}(\mathsf{s}(z),\mathsf{0}) = \mathsf{true}\,,\ \ \Gamma\sigma_1\ \ ;\ \ \aleph\sigma_1$$

by means of the *covering set of substitutions* $\sigma_0 := \{x \mapsto \mathsf{0}\},\quad \sigma_1 := \{x \mapsto \mathsf{s}(z)\},$ where z is a new constructor variable. Now Lemma Rewrite can transform these two goals into the following two new goals:

$$\mathsf{true} = \mathsf{true},\ \ \Gamma\sigma_0\ \ ;\ \ \aleph\sigma_0$$
$$\mathsf{false} = \mathsf{true},\ \ \Gamma\sigma_1\ \ ;\ \ \aleph\sigma_1$$

The former can be removed by =-Decompose, the latter can be transformed by an =-Literal Remove into

$$\Gamma\sigma_1\ \ ;\ \ \aleph\sigma_1$$

All in all, this is an appropriate transformation of the original goal, since the information of the literal $\mathsf{leq}(x,\mathsf{0}) = \mathsf{true}$ has been consumed in a way that tends to make the success of the proof attempt more likely than before: Namely the applied σ_1 carries the information that x can be assumed to be different from $\mathsf{0}$ now.

A question one could ask is the following: "The application of a covering set of substitutions is nothing but a complete case analysis; why don't you then use the above Context Add rule to generate this case analysis?"

It is true indeed that both the application of a covering set of substitutions and a Context Add mean a complete case analysis. However, there are two reasons for our distinction:

First reason: Our Context Add is rather poor regarding the introduction of new variables: Suppose we had introduced the variable z by Context Add to " $\mathsf{leq}(x,\mathsf{0}) = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$ " via $\lambda_0 := x{=}\mathsf{0};\quad \lambda_1 := x{=}\mathsf{s}(z);$ yielding the new goals

$$x \neq \mathsf{0},\ \ \mathsf{leq}(x,\mathsf{0}) = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$$
$$x \neq \mathsf{s}(z),\ \ \mathsf{leq}(x,\mathsf{0}) = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$$
$$x = \mathsf{0},\ \ x = \mathsf{s}(z),\ \ \mathsf{leq}(x,\mathsf{0}) = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$$

The first two are rather fine: By Constant Rewrite they can be transformed into

$$x \neq \mathsf{0},\ \ \mathsf{leq}(\mathsf{0},\mathsf{0}) = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$$
$$x \neq \mathsf{s}(z),\ \ \mathsf{leq}(\mathsf{s}(z),\mathsf{0}) = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$$

and then by Lemma Rewrite into

$$x \neq \mathsf{0},\ \ \mathsf{true} = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$$
$$x \neq \mathsf{s}(z),\ \ \mathsf{false} = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$$

The first can be removed by an =-Decompose, the second transformed by an =-Literal Remove into

$$x \neq \mathsf{s}(z),\ \ \Gamma\ \ ;\ \ \aleph$$

which is quite appropriate for the further proof. (But note that this semantic equality of x and $\mathsf{s}(z)$ is weaker than their syntactic equality expressed by the σ_1 from above.) Trouble, however, is caused by the goal

$$x = \mathsf{0},\ \ x = \mathsf{s}(z),\ \ \mathsf{leq}(x, \mathsf{0}) = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$$

we have left untransformed above: The context

$$x = \mathsf{0},\ \ x = \mathsf{s}(z)$$

is likely to be of no use for the proof of " $\mathsf{leq}(x, \mathsf{0}) = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$ " (since z does not occur in " $\mathsf{leq}(x, \mathsf{0}) = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$ "). Thus, if the original Context Add should bring our proof of " $\mathsf{leq}(x, \mathsf{0}) = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$ " further, then the above context itself should be valid. This is not the case, however. The reason is, that we actually should have made our Context Add via $\lambda_0 := x{=}\mathsf{0};\ \ \lambda_1 := \exists z\colon\ x{=}\mathsf{s}(z)$. Then, the above context would be

$$x = \mathsf{0},\ \ \exists z\colon\ x = \mathsf{s}(z)$$

Since this is valid, the situation would be better then. But still we would not know how to prove this without application of a covering set of substitutions to x. Moreover, as long as we do not admit existentially quantified variables in our formulas, the new variable z is implicitly universally quantified, which makes the above case analysis via Context Add useless for our proof.

Second reason for the introduction of a rule for applying a covering set of substitutions: If the weight $\aleph$ of the formula " $\mathsf{leq}(x, \mathsf{0}) = \mathsf{true},\ \ \Gamma\ \ ;\ \ \aleph$ " from above depends on the variable x and we apply the substitution σ_1 from above, then the new weight $\aleph\sigma_1$ depends on the variable z instead. This change of dependency may be necessary for successful application of induction hypotheses. It is also sound in this case. In general, however (unless the induction ordering is semantic), exchanging the goal " $x \neq \mathsf{s}(z),\ \ \Gamma\ \ ;\ \ \aleph$ " with " $x \neq \mathsf{s}(z),\ \ \Gamma\sigma_1\ \ ;\ \ \aleph\sigma_1$ " is not sound due to the instantiation of $\aleph$ and of possible '$<$'-literals in Γ, since the induction ordering may depend on the syntactic form of the terms represented by x while the context $x \neq \mathsf{s}(z)$ only implies semantic equality of x and $\mathsf{s}(z)$. Thus, we would need something like $x \not\equiv \mathsf{s}(z)$ instead of $x \neq \mathsf{s}(z)$, where '$\equiv$' denotes syntactic equality.

All in all, to have a successful case analysis in the above situation without using a covering set of substitutions we have to add some inference rule similar to the one saying that " $x \equiv \mathsf{0},\ \ \exists z\colon\ x \equiv \mathsf{s}(z)$ " is a lemma and then to use it for a Lemma Apply via $\lambda_0 := x \equiv \mathsf{0};\ \ \lambda_1 := \exists z\colon\ x \equiv \mathsf{s}(z)$. Since this requires a considerable extension of our syntax for formulas, we prefer to add another inference rule for adding covering sets of substitutions instead.

Definition 106 (Covering Sets of Substitutions)
Let $(\Gamma, \aleph) \in \mathrm{SynCons}$. A *covering set of substitutions for* $(\Gamma, \aleph)$ is a finite sequence $(n \in \mathbb{N})$

$$\sigma_0, \ldots, \sigma_n \ \in \mathcal{SUB}(\mathrm{V}, \mathcal{T})$$

such that

$$\forall (\tau, \mathcal{A}) \in \mathrm{Info}\colon\ \exists i \preceq n\colon\ \exists \varphi\colon \left(\begin{array}{ll} & (\varphi, \mathcal{A}) \in \mathrm{Info} \\ \wedge & \aleph\tau \gtrsim_{\mathcal{A}} \aleph(\sigma_i\varphi) \\ \wedge & (\tau\mathcal{A}_{\iota_{\mathcal{A}}})|_{\mathcal{V}(\Gamma)} = (\sigma_i\varphi\mathcal{A}_{\iota_{\mathcal{A}}})|_{\mathcal{V}(\Gamma)} \\ \wedge & \left(\begin{array}{ll} & \tau|_{\mathcal{V}_<(\Gamma)} = (\sigma_i\varphi)|_{\mathcal{V}_<(\Gamma)} \\ \vee & \text{the induction ordering is semantic} \end{array} \right) \end{array} \right).$$

Let $n \in \mathbb{N}$; $\Gamma \in \mathrm{Form}$; $\aleph \in \mathrm{Weight}$.
Let L be a finite subset of 'Form', and H, G be finite subsets of 'SynCons':

Substitution Add: $$\frac{(L \ , H \ , G \cup \{\ (\Gamma \ , \ \aleph \ )\ \})}{(L \ , H \ , G \cup \{\ (\Gamma\sigma_i, \ \aleph\sigma_i) \mid i \preceq n\ \})}$$

if $\sigma_0, \ldots, \sigma_n$ is a covering set of substitutions for $(\Gamma, \aleph)$.

Lemma 107 *The Substitution Add rule is a safe sub-rule of the Transformation rule.*

A first step towards operationalizing covering sets of substitutions is given by ...

Corollary 108
Let $(\Gamma, \aleph) \in \mathrm{SynCons}$. *A finite sequence* $(n \in \mathbb{N})$ $\sigma_0, \ldots, \sigma_n \in \mathcal{SUB}(\mathrm{V}, \mathcal{T})$ *with*
$\forall \tau \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{V_{SIG}}))\colon\ \exists i \preceq n\colon\ \exists \varphi \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{V_{SIG}}))\colon\ \tau|_{\mathcal{V}(\Gamma,\aleph)} = (\sigma_i\varphi)|_{\mathcal{V}(\Gamma,\aleph)}$
is a covering set of substitutions for $(\Gamma, \aleph)$.

The following corollary can be used to construct simple covering sets of substitutions of the form of Corollary 108.

Corollary 109 *Let* $(\Gamma, \aleph) \in \mathrm{SynCons}$.

1. $\mathrm{id}|_{\mathrm{V}}$ *is a covering set of substitutions for* $(\Gamma, \aleph)$.

2. *Assume that there are exactly* $m{+}1$ $(m \in \mathbb{N})$ *constructor symbols for some sort* $\hat{s} \in \mathbb{S}$, *namely* $c_0, \ldots, c_m \in \mathbb{C}$ *with* $\forall j \preceq m\colon\ \alpha(c_j) = s_{j,0} \ldots s_{j,l_j-1} \longrightarrow \hat{s}$. *Assume* $x \in \mathrm{V}_{\mathrm{CONS},\hat{s}}$ *and* $\forall j \preceq m\colon\ \forall k \prec l_j\colon\ y_{j,k} \in \mathrm{V}_{\mathrm{CONS},s_{j,k}}$ *with* $\forall j \preceq m\colon\ \forall k \prec l_j\colon\ \forall k' \prec k\colon\ y_{j,k'} \neq y_{j,k}$. *Define* $\sigma'_j \in \mathcal{SUB}(\mathrm{V}, \mathcal{T})$ *by* $\sigma'_j|_{\mathrm{V}\setminus\{x\}} \subseteq \mathrm{id}$ *and* $x\sigma'_j := c_j(y_{j,0}, \ldots, y_{j,l_j-1})$.
Now, if $\sigma_0, \ldots, \sigma_n$ *is a covering set of substitutions for* $(\Gamma, \aleph)$, $i \preceq n$, *and* $\forall j \preceq m\colon\ \forall k \prec l_j\colon\ y_{j,k} \notin \sigma_i[\mathcal{V}(\Gamma, \aleph)] \setminus \{x\}$, *then* $\sigma_0, \ldots, \sigma_{i-1}, \sigma_i\sigma'_0, \ldots, \sigma_i\sigma'_m, \sigma_{i+1}, \ldots, \sigma_n$ *is a covering set of substitutions for* $(\Gamma, \aleph)$, *too.*

Finally, we want to remark that our kind of covering sets of substitutions may not be sufficient for successful inductive theorem proving in case of non-free constructors:

Example 110
Let $\mathsf{0}$, s, *and* p *be constructor symbols for the sort* int *of integers. Let* $\mathsf{nonnegp}$ *be a non-constructor predicate with arity "*$\mathsf{int} \longrightarrow \mathsf{bool}$*". Let* x, y *be constructor variables of the sort* int. *Let* R_{110} *contain the following rules:*

$$\begin{array}{lcll}
\mathsf{s}(\mathsf{p}(x)) & = & x & \\
\mathsf{p}(\mathsf{s}(x)) & = & x & \\
\mathsf{nonnegp}(\mathsf{0}) & = & \mathsf{true} & \\
\mathsf{nonnegp}(\mathsf{s}(x)) & = & \mathsf{true} & \longleftarrow \ \mathsf{nonnegp}(x) = \mathsf{true} \\
\mathsf{nonnegp}(\mathsf{p}(\mathsf{0})) & = & \mathsf{false} & \\
\mathsf{nonnegp}(\mathsf{p}(\mathsf{p}(x))) & = & \mathsf{false} & \longleftarrow \ \mathsf{nonnegp}(\mathsf{p}(x)) = \mathsf{false}
\end{array}$$

Suppose that further non-constructor function symbols have the same definition scheme as $\mathsf{nonnegp}$. For this definition scheme one might consider covering sets of substitutions like

$$\begin{array}{ll}
\sigma_0 & := \{\ x \mapsto \mathsf{0}\ \ \ \ \ \ \ \ \}, \\
\sigma_1 & := \{\ x \mapsto \mathsf{s}(x)\ \ \ \ \}, \\
\sigma_2 & := \{\ x \mapsto \mathsf{p}(\mathsf{0})\ \ \ \}, \\
\sigma_3 & := \{\ x \mapsto \mathsf{p}(\mathsf{p}(x))\}
\end{array}$$

to be appropriate. The experienced reader, however, may recognize that adding the substitution σ_1, e.g., is likely to be of no use for most inductive proofs without also adding the context "$\mathsf{nonnegp}(x) \neq \mathsf{true}$". Such an adding of substitution and context has to be done in a single step because adding the context later via Context Add leaves the proof with the case $(\sigma_1, \mathsf{nonnegp}(x) = \mathsf{true})$ which does not fit into the definition scheme and thus is very unlikely to be easier to prove than the original goal. This observation, however, does not lead us to the augmentation of covering sets with context (as other authors do, cf. e.g. Definition 2 in Bronsard & Reddy (1991)) and the design of an inference rule for simultaneous adding of substitution and context. Instead, we advise the specifier, not to use definition schemes of the above kind for non-free constructors, but to proceed as exhibited by the following set of rules, which does not require context-augmented covering sets of substitutions and, beyond that, has no feasible proper critical pair anymore. Its confluence can easily be inferred via each of the (independent) theorems 68, 71, 75, 76, 77, and 78 of Wirth (1995).

Example 111
Let R_{111} *contain the following rules:*

$$\begin{array}{lcll}
\mathsf{s}(\mathsf{p}(x)) & = & x & \\
\mathsf{p}(\mathsf{s}(x)) & = & x & \\
\mathsf{nonnegp}(x) & = & \mathsf{true} & \longleftarrow \ x = \mathsf{0} \\
\mathsf{nonnegp}(x) & = & \mathsf{true} & \longleftarrow \ \mathsf{nonnegp}(\mathsf{p}(x)) = \mathsf{true} \\
\mathsf{nonnegp}(x) & = & \mathsf{false} & \longleftarrow \ x = \mathsf{p}(\mathsf{0}) \\
\mathsf{nonnegp}(x) & = & \mathsf{false} & \longleftarrow \ \mathsf{nonnegp}(\mathsf{s}(x)) = \mathsf{false}
\end{array}$$

This does not mean that we want to forbid syntactic case distinctions for sorts with non-free constructors in function definitions. We only want to avoid their combination with incomplete conditions in specifications of total functions. The following extension of the above definition, e.g., does not cause problems with covering sets of substitutions:

Example 112
Let R_{112} *contain the following rules:*

$$
\begin{array}{lcll}
x+\mathsf{0} & = & x & \\
x+\mathsf{s}(y) & = & \mathsf{s}(x+y) & \\
x+\mathsf{p}(y) & = & \mathsf{p}(x+y) & \\
x-\mathsf{0} & = & x & \\
x-\mathsf{s}(y) & = & \mathsf{p}(x-y) & \\
x-\mathsf{p}(y) & = & \mathsf{s}(x-y) & \\
\mathsf{invert}(\mathsf{0}) & = & \mathsf{0} & \\
\mathsf{invert}(\mathsf{s}(x)) & = & \mathsf{p}(\mathsf{invert}(x)) & \\
\mathsf{invert}(\mathsf{p}(x)) & = & \mathsf{s}(\mathsf{invert}(x)) & \\
\mathsf{abs}(x) & = & x & \longleftarrow \ \mathsf{nonnegp}(x)=\mathsf{true} \\
\mathsf{abs}(x) & = & \mathsf{invert}(x) & \longleftarrow \ \mathsf{nonnegp}(x)=\mathsf{false}
\end{array}
$$

Note, however, that now the only known theorem that allows to infer confluence of $\longrightarrow_{\mathrm{R}_{111}\cup\mathrm{R}_{112},\mathrm{V}_{\mathrm{SIG}}}$ conveniently is Theorem 77 of Wirth (1995). Moreover, we do not know how to prove the important lemma

$$\mathsf{nonnegp}(x)=\mathsf{true},\ \ \mathsf{nonnegp}(x)=\mathsf{false}$$

with our formulas because this needs terms like $\mathsf{s}^n(\mathsf{0})$ which are beyond our term language.

7.2 Application of Hypotheses

As illustrated in section 1.1, the Substitution Add rule of the previous section is usually not sufficient to provide us with finite proofs utilizing the term-generatedness ("no junk") (given by some notion of inductive validity) unless cyclic arguments can be used. These cyclic arguments become possible with the Hypothesis Apply and Hypothesis Rewrite rules we present in this section.

Let $m, n \in \mathbb{N}$; $\Gamma, \Pi, \Pi', \Lambda_0, \ldots, \Lambda_m, \Theta \in$ Form; $\lambda_0, \ldots, \lambda_{m-1} \in \mathcal{GENLIT}(\mathrm{sig}, \mathrm{V})$; $\aleph, \beth, \beth' \in$ Weight.
Let L be a finite subset of 'Form', and H, G be finite subsets of 'SynCons':

Hypothesis Apply:

$$\frac{(L \;, H \cup \{(\Pi, \beth)\} \;, G \cup \{\, (\quad \Gamma \quad , \aleph) \quad\})}{\begin{array}{l}(L \;, H \cup \{(\Pi, \beth)\} \;, G \cup \{\, (\quad \overline{\lambda_i}\, \Lambda_i \Gamma \quad , \aleph) \mid i \prec m \;\} \\ \qquad\qquad\qquad\qquad \cup \{\, (\quad (\beth' < \aleph) \Lambda_m \Gamma \quad , \aleph) \quad\})\end{array}}$$

if, for some $n \preceq m$, $(\Pi', \beth', \mathrm{Y}, \Theta)$ results from application of some $\mu \in \mathcal{GENSUB}(\mathrm{V}, \mathcal{T})$ to $(\Pi, \beth)$ w.r.t. $\mathcal{V}(\lambda_{n-1}\lambda_n \ldots \lambda_0 \Gamma, \aleph)$ and all the following holds:

- $\forall i \prec n$: $\big(\ \Lambda_i$ is contained in $\lambda_{i-1}\lambda_{i-2}\ldots\lambda_0\ \big)$,
- $\forall i \preceq m$: $\big(\ \Lambda_i$ is contained in $\lambda_{i-1}\lambda_{i-2}\ldots\lambda_0\Theta\ \big)$,
- Π' is contained in $\lambda_{m-1}\lambda_{m-2}\ldots\lambda_0\Theta\Gamma$,
- $\forall x \in \mathrm{Y}$: $\big(\ \overline{(\mathrm{Def}\, x\mu)}$ is contained in $\lambda_{n-1}\lambda_{n-2}\ldots\lambda_0\Gamma\ \big)$,
- $\forall x \in \mathrm{GENDOM}(\mu)$: $\big(\ \overline{(\mathrm{Def}\, x\mu)}$ is contained in $\lambda_{m-1}\lambda_{m-2}\ldots\lambda_0\Gamma\ \big)$.

Lemma 113 *The Hypothesis Apply rule is a safe sub-rule of the Transformation rule.*

The Hypothesis Apply rule differs from the Lemma Apply rule in adding a further subgoal saying that the instantiated weight of the hypothesis is smaller than the weight of the original goal in the context of the goal and the literals of the lazy application. For a first understanding assume $\mathrm{Y} = \emptyset$, i.e. $(\ \mathcal{V}_<(\Pi) \cup \mathcal{V}(\beth)\ ) \cap \mathrm{GENDOM}(\mu) = \emptyset$ or the induction ordering is semantic. Then we get $\Theta = \emptyset$, $\Pi' = \Pi\mu$, and $\beth' = \beth\mu$. The intuition behind the Hypothesis Apply rule in the complementary case is exhibited by the following Example 114.

Example 114 *(continuing examples 9 and 10)*
To prove (Irrefl 2 ack*) of Example 10 we start with the new goal*
(114.1) $\mathsf{less}(y,\mathsf{ack}(x,y)) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(x,y)$
First we use the Memorizing Rule of section 3.4 to copy (114.1) into the set of induction hypotheses. Then Substitution Add yields the two new goals:
(114.1.1) $\mathsf{less}(y,\mathsf{ack}(0,y)) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(0,y)$
(114.1.2) $\mathsf{less}(y,\mathsf{ack}(\mathsf{s}(x),y)) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x),y)$
*Lemma Rewrite with (*ack*1) yields:*
(114.1.1.1) $\mathsf{less}(y,\mathsf{s}(y)) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(0,y)$
which is removed by Lemma Apply with (True less*) of Example 10.*
Further Substitution Add to (114.1.2) yields:
(114.1.2.1) $\mathsf{less}(0,\mathsf{ack}(\mathsf{s}(x),0)) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x),0)$
(114.1.2.2) $\mathsf{less}(\mathsf{s}(y),\mathsf{ack}(\mathsf{s}(x),\mathsf{s}(y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y))$
Lemma Apply of (Pos ack*) removes (114.1.2.1).*
*Lemma Rewrite with (*ack*3) yields:*
(114.1.2.2.1) $\mathsf{less}(\mathsf{s}(y),\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y))$
A Lemma Apply of (Strict Trans less*) via* $m := 4$; $\lambda_0 := \overline{\mathrm{Def}\,\mathsf{ack}(\mathsf{s}(x),y)}$; $\lambda_1 := \overline{\mathrm{Def}\,\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))}$; $\lambda_2 := \mathsf{less}(y,\mathsf{ack}(\mathsf{s}(x),y)) \neq \mathsf{true}$; $\lambda_3 := \mathsf{less}(\mathsf{ack}(\mathsf{s}(x),y),\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))) \neq \mathsf{true}$ *; yields*
(114.1.2.2.1.1) $\mathrm{Def}\,\mathsf{ack}(\mathsf{s}(x),y),\;\; \mathsf{less}(\mathsf{s}(y),\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y))$
(114.1.2.2.1.2) $\mathrm{Def}\,\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y)),$
$\overline{\mathrm{Def}\,\mathsf{ack}(\mathsf{s}(x),y)},\;\; \mathsf{less}(\mathsf{s}(y),\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y))$
(114.1.2.2.1.3) $\mathsf{less}(y,\mathsf{ack}(\mathsf{s}(x),y)) = \mathsf{true},$
$\overline{\mathrm{Def}\,\mathsf{ack}(\mathsf{s}(x),y)},\;\; \mathsf{less}(\mathsf{s}(y),\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y))$
(114.1.2.2.1.4) $\mathsf{less}(\mathsf{ack}(\mathsf{s}(x),y),\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))) = \mathsf{true},$
$\overline{\mathrm{Def}\,\mathsf{ack}(\mathsf{s}(x),y)},\;\; \mathsf{less}(\mathsf{s}(y),\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y))$
*Lemma Apply of (*Def ack*) removes (114.1.2.2.1.1) and (114.1.2.2.1.2). Hypothesis Apply of (114.1) to (114.1.2.2.1.3) via* $m := 0$; $\mu_1 := \{\, x \mapsto \mathsf{s}(x) \,\}$; $\mathrm{Y} := \Theta := \emptyset$ *yields:*
(114.1.2.2.1.3.1) $\mathsf{w}_{114}(\mathsf{s}(x),y) < \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y)),\;\; \mathsf{less}(y,\mathsf{ack}(\mathsf{s}(x),y)) = \mathsf{true},$
$\overline{\mathrm{Def}\,\mathsf{ack}(\mathsf{s}(x),y)},\;\; \mathsf{less}(\mathsf{s}(y),\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y))$
Now we come to the Hypothesis Apply that explains the case of $\mathrm{Y} \neq \emptyset$*. Hence, we assume that the induction ordering is not semantic.*
We apply (114.1) to (114.1.2.2.1.4), choosing $m := 0$; $\mu_0 := \{\, y \mapsto \mathsf{ack}(\mathsf{s}(x),y) \,\}$; $\mathrm{Y} := \{y\}$; $\Lambda_m := \Theta := z \neq \mathsf{ack}(\mathsf{s}(x),y)$; $\urcorner := \mathsf{w}_{114}(x,y)$; *and* $\urcorner' := \mathsf{w}_{114}(x,z)$.
The resulting goal is:
(114.1.2.2.1.4.1) $\mathsf{w}_{114}(x,z) < \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y)),\;\; z \neq \mathsf{ack}(\mathsf{s}(x),y),$
$\mathsf{less}(\mathsf{ack}(\mathsf{s}(x),y),\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))) = \mathsf{true},$
$\overline{\mathrm{Def}\,\mathsf{ack}(\mathsf{s}(x),y)},\;\; \mathsf{less}(\mathsf{s}(y),\mathsf{ack}(x,\mathsf{ack}(\mathsf{s}(x),y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y))$
Finally, to complete the proof it suffices to solve the following parts of the ordering conditions of (114.1.2.2.1.3.1) and (114.1.2.2.1.4.1), resp.:
(114.1.2.2.1.3.1.1) $\mathsf{w}_{114}(\mathsf{s}(x),y) < \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y))$
(114.1.2.2.1.4.1.1) $\mathsf{w}_{114}(x,z) < \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y)),\;\; z \neq \mathsf{ack}(\mathsf{s}(x),y)$
We can simply solve this by using a lexicographic ordering on the arguments of w_{114}.[31]

[31] Note, however, that we cannot directly use the lexicographic path ordering because that would require
(114.1.2.2.1.4.1.1.1) $z < \mathsf{w}_{114}(\mathsf{s}(x),\mathsf{s}(y)),\;\; z \neq \mathsf{ack}(\mathsf{s}(x),y)$
which would be quite difficult to satisfy.

We will now use Example 114 for a discussion of the Lemma and Hypothesis Apply rules according to our design goals of section 1.3.

The careful reader may have noticed that the crucial step in the proof of Example 114 is the application of the lemma (Strict Trans less) of Example 10.

The substitution applied to the lemma is

$$\mu := \{\ x \mapsto y,\ y \mapsto \mathsf{ack}(\mathsf{s}(x), y),\ z \mapsto \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))\ \}.$$

While the values of μ for x and z are directly given by matching the literal $\mathsf{less}(\mathsf{s}(x), z)$ of the lemma to the literal $\mathsf{less}(\mathsf{s}(y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y)))$ of the goal (114.1.2.2.1), the value of μ for y is not obvious from the current state of the proof attempt. Nevertheless, it is crucial for a successful completion of this proof attempt that this value is exactly $\mathsf{ack}(\mathsf{s}(x), y)$. This violates the design goal called "natural flow of information" in section 1.3. More precisely, this violation corresponds[32] to one of those applications of the γ-rule that were indicated as problematic in section 1.3 because the instantiation of y has to be guessed before it can be recognized which instantiation will make the proof attempt successful. The design goal of "natural flow of information" requires "that a certain decision can be delayed until the state of the proof provides sufficient information to make a successful decision". How can we achieve this? The usual procedure is to replace y with a special kind of existential variable called "dummy variable" in Prawitz (1960) and Kanger (1963) or "free variable" in the framework of semantic tableaux[33], cf. Fitting (1990). We will try this in the following example:

Example 115 *(continuing Example 114)*
Assume that we have a state of a proof attempt that is given exactly by that of Example 114 before the Lemma Apply of (Strict Trans less*). We now continue this proof attempt with a free existential constructor variable* $\bar{y}$*:*

A Lemma Apply of (Strict Trans less*) via* $m := 3$*;* $\lambda_0 := \overline{\mathrm{Def}\,\mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))}$ *;* $\lambda_1 := \mathsf{less}(y, \bar{y}) \neq \mathsf{true}$ *;* $\lambda_2 := \mathsf{less}(\bar{y}, \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) \neq \mathsf{true}$ *; yields*

(115.1.2.2.1.2) $\mathrm{Def}\,\mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y)),$
$\mathsf{less}(\mathsf{s}(y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) = \mathsf{true}\ ;\ \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y))$

(115.1.2.2.1.3) $\mathsf{less}(y, \bar{y}) = \mathsf{true},\ \mathsf{less}(\mathsf{s}(y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) = \mathsf{true}\ ;\ \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y))$

(115.1.2.2.1.4) $\mathsf{less}(\bar{y}, \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) = \mathsf{true},$
$\mathsf{less}(\mathsf{s}(y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) = \mathsf{true}\ ;\ \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y))$

Now it becomes interesting to apply our hypothesis (114.1) to (115.1.2.2.1.4) because the first literals of the two are unifiable when we treat the usual variables of the goal as constants. This unifier now tells us that we should instantiate $\bar{y}$ *with* $\mathsf{ack}(\mathsf{s}(x), y)$*. We have overcome the problem of guessing the right instantiation of the variable* y *of the lemma by replacing it with the dummy variable* $\bar{y}$ *until it became obvious which instantiation is likely to make the proof attempt successful. When instantiating* $\bar{y}$ *with* $\mathsf{ack}(\mathsf{s}(x), y)$ *(which actually produces the additional goal* $\mathrm{Def}\,\mathsf{ack}(\mathsf{s}(x), y)$*), this Hypothesis Apply yields essentially the same goal as in Example 114:*

(115.1.2.2.1.4.1) $\mathsf{w}_{114}(x, z) < \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y)),\ z \neq \mathsf{ack}(\mathsf{s}(x), y),$
$\mathsf{less}(\mathsf{ack}(\mathsf{s}(x), y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) = \mathsf{true},$
$\mathsf{less}(\mathsf{s}(y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) = \mathsf{true}\ ;\ \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y))$

[32] when writing the universally quantified lemma into the antecedent and our goal into the succedent of a sequent

[33] where the variables are the dual of existential, i.e. universal.

It should be pointed out that the instantiation of the free existential variable $\bar{y}$ during the transformation of (115.1.2.2.1.4) to (115.1.2.2.1.4.1) also changes the goal (115.1.2.2.1.3) where the occurrence of $\bar{y}$ has to be instantiated with the same term, i.e. with $\mathsf{ack}(\mathsf{s}(x), y)$. A free variable is *global* in the sense that it denotes the same object in all prover formulas in which it occurs. This violates the "mutual independence of the proof sub-problems represented by the formulas at the open nodes in the proof graph", which is one of our design goals of section 1.3. We do not know which way is the best to overcome this violation. In the example, of course, we can make $\bar{y}$ local simply by extending our prover formulas to consist of super-literals (i.e. conjunctions of literals) instead of literals — resulting in a system with local free existential variables similar to those in the sequent calculus of Appendix B. In general, however, reasoning with these super-literals either needs an extension of the literals in the super-literals to disjunctions of literals, an extension of these literals to super-literals, etc.[34] or a premature instantiation of the free variable[35].

Note that the possibility of solving the instantiation problem by drawing connections between the literals containing unbound[36] variables (like the 'y' from above) and complementary literals of other lemmas or hypotheses (resulting in a single inference rule applying several lemmas and hypotheses) is rather limited: In general, we need the whole inference system and not only Lemma and Hypothesis Apply rules for finding the appropriate instantiation for the unbound variable, cf. the justification of our design goal of a "homogeneous representation" in section 1.3.

Thus, we do not know how to treat free existential variables according to our design goals without extending our framework considerably. Moreover, we have not calculated enough examples with free existential variables and therefore our ideas on their treatment are not as mature as the rest of our inference system. Furthermore, we have not clearly understood the advantages of the treatment of existential variables in the style of Padawitz (1992). Finally, the inclusion of existential variables would make our framework technically more complicated. For all those reasons, we decided not to include free existential variables in this paper.

[34] which together with our design goal of a "homogeneous representation" leads to a considerable extension of our language for prover formulas and a considerable complication of our inference rules

[35] which again may violate our design goal of "natural flow of information"

[36] i.e. not bound by the substitution matching the lemma (or hypotheses) to the goal

The proof of Example 114 does not obey our design goal of "natural flow of information" in even another aspect: The bridge lemma (Strict Trans less) is applied although only one of its three literals is found in the goal, which means that it is not obvious from the state of the proof that the decision to apply it, is likely to render the proof attempt successful. Moreover, even though only a few lemmas are listed in Example 10, there is quite a big number of similarly interesting lemmas. Even a very good heuristic for picking the right one does not help if the lemma is not known at all. Interestingly, NQTHM[37] proves the lemma (Irrefl 2 ack) automatically, even when the lemma (Strict Trans less) is not provided and the function 'less' is redefined such that the built-in features for treating arithmetics cannot help. Since NQTHM has no existential variables either, it may be a good idea to search for a proof of (Irrefl 2 ack) according to the heuristic knowledge built into the induction rule of explicit inductive theorem proving:

Example 116 *(continuing Example 114)*
Assume that we have a state of a proof attempt that is given exactly by that of Example 114 before the Lemma Apply of (Strict Trans less*). We now continue this proof attempt in a style very similar to explicit induction. By matching the subterm of the induction hypothesis (114.1) that has the expanded function 'ack' as head symbol to the subterms of the goal (114.1.2.2.1) we get the substitutions* $\mu_0 := \{ y \mapsto \mathsf{ack}(\mathsf{s}(x), y) \}$ *and* $\mu_1 := \{ x \mapsto \mathsf{s}(x) \}$*. Therefore, according to the heuristic knowledge built into the induction rule of explicit inductive theorem proving, the following two Hypothesis Applies of (114.1) are likely to be successful. The first uses* μ_0 *and yields the following new goals:*

(116.1.2.2.1.1) $\mathsf{Def}\,\mathsf{ack}(\mathsf{s}(x), y),\quad \mathsf{less}(\mathsf{s}(y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y))$

(116.1.2.2.1.2) $\mathsf{less}(\mathsf{ack}(\mathsf{s}(x), y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) \neq \mathsf{true},$
$\overline{\mathsf{Def}\,\mathsf{ack}(\mathsf{s}(x), y)},\quad \mathsf{less}(\mathsf{s}(y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y))$

(116.1.2.2.1.3) $\mathsf{w}_{114}(x, z) < \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y)),\quad z \neq \mathsf{ack}(\mathsf{s}(x), y),$
$\mathsf{less}(\mathsf{s}(y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y))$

*Lemma Apply of (*Def ack*) removes (116.1.2.2.1.1).*
The second Hypothesis Apply uses μ_1 *and transforms (116.1.2.2.1.2) into*

(116.1.2.2.1.2.1) $\mathsf{less}(y, \mathsf{ack}(\mathsf{s}(x), y)) \neq \mathsf{true},$
$\mathsf{less}(\mathsf{ack}(\mathsf{s}(x), y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) \neq \mathsf{true},$
$\overline{\mathsf{Def}\,\mathsf{ack}(\mathsf{s}(x), y)},\quad \mathsf{less}(\mathsf{s}(y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y))$

(116.1.2.2.1.2.2) $\mathsf{w}_{114}(\mathsf{s}(x), y) < \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y)),$
$\mathsf{less}(\mathsf{ack}(\mathsf{s}(x), y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) \neq \mathsf{true},$
$\overline{\mathsf{Def}\,\mathsf{ack}(\mathsf{s}(x), y)},\quad \mathsf{less}(\mathsf{s}(y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y))$

Now a Lemma Apply of (Strict Trans less*) can transform (116.1.2.2.1.2.1) into:*

(116.1.2.2.1.2.1.1) $\mathsf{Def}\,\mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y)),\quad \mathsf{less}(y, \mathsf{ack}(\mathsf{s}(x), y)) \neq \mathsf{true},$
$\mathsf{less}(\mathsf{ack}(\mathsf{s}(x), y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) \neq \mathsf{true},$
$\overline{\mathsf{Def}\,\mathsf{ack}(\mathsf{s}(x), y)},\quad \mathsf{less}(\mathsf{s}(y), \mathsf{ack}(x, \mathsf{ack}(\mathsf{s}(x), y))) = \mathsf{true} \;\;;\;\; \mathsf{w}_{114}(\mathsf{s}(x), \mathsf{s}(y))$

*which can be removed by Lemma Apply of (*Def ack*).*
Now the proof attempt can be completed just as in Example 114.
Note that now the application (Strict Trans less*) is straightforward (contrary to Example 114) because all its literals occur in the goal. Moreover, even if (Strict Trans* less*) is not known, it can be generated from (116.1.2.2.1.2.1) by good generalization heuristics, cf. e.g. Boyer & Moore (1979), Walther (1994).*

[37] cf. Boyer & Moore (1988)

Finally, we present the Hypothesis Rewrite rule that is for the Hypothesis Apply what the Lemma Rewrite is for the Lemma Apply as discussed in section 5.5. We could repeat the whole discussion of section 5.5 in a slightly adapted form here, but we think that the adaptation is obvious and the discussion can be omitted.

Let $m, n \in \mathbb{N}$; $\Gamma, \Pi, \Pi', \Lambda_0, \ldots, \Lambda_{m+1}, \Theta \in$ Form; $\lambda_0, \ldots, \lambda_m \in \mathcal{GENLIT}(\mathrm{sig}, \mathrm{V})$; $\aleph, \beth, \beth' \in$ Weight. $\lambda'' \in \mathcal{LIT}(\mathrm{sig}, \mathrm{V})$; $p \in \mathcal{POS}(\lambda'') \backslash \{\emptyset\}$.
Let L be a finite subset of 'Form', and H, G be finite subsets of 'SynCons':

Hypothesis Rewrite:

$$\frac{(L \;, H \cup \{(\Pi, \beth)\} \;, G \cup \{\, (\quad \Gamma\lambda''\Delta \quad , \aleph) \quad\})}{\begin{array}{l}(L \;, H \cup \{(\Pi, \beth)\} \;, G \cup \{\, (\quad \overline{\lambda_i}\,\Lambda_i\Gamma\lambda''\Delta \quad , \aleph) \mid i \prec m \;\} \\ \qquad\qquad\qquad\qquad \cup \{\, (\quad \Lambda_m\Gamma\lambda''[\,p \leftarrow r\,]\Delta \quad , \aleph) \quad\} \\ \qquad\qquad\qquad\qquad \cup \{\, (\quad (\beth'{<}\aleph)\Lambda_{m+1}\Gamma\lambda''\Delta \quad , \aleph) \quad\})\end{array}}$$

if, for some $n \preceq m$, $(\Pi', \beth', \mathrm{Y}, \Theta)$ results from application of some $\mu \in \mathcal{GENSUB}(\mathrm{V}, \mathcal{T})$ to $(\Pi, \beth)$ w.r.t. $\mathcal{V}(\lambda_{n-1}\lambda_n \ldots \lambda_0\Gamma\lambda''\Delta, \aleph)$ and all the following holds:

- λ_m equals $(l{=}r)$ for $l := \lambda''/p$,
- $\lambda_{m-1}\lambda_{m-2}\ldots\lambda_0\Theta$ is contained in Λ_m,
- $\forall i \prec n$: $\big(\ \Lambda_i$ is contained in $\lambda_{i-1}\lambda_{i-2}\ldots\lambda_0\ \big)$,
- $\forall i \preceq m{+}1$: $\big(\ \Lambda_i$ is contained in $\lambda_{i-1}\lambda_{i-2}\ldots\lambda_0\Theta\ \big)$,
- Π' is contained in $\lambda_m\Lambda_m\Gamma\Delta$,
- $\forall x \in \mathrm{Y}$: $\big(\ \overline{(\mathrm{Def}\, x\mu)}$ is contained in $\lambda_{n-1}\lambda_{n-2}\ldots\lambda_0\Gamma\Delta\ \big)$,
- $\forall x \in \mathrm{GENDOM}(\mu)$: $\big(\ \overline{(\mathrm{Def}\, x\mu)}$ is contained in $\lambda_{m-1}\lambda_{m-2}\ldots\lambda_0\Gamma\Delta\ \big)$.

Lemma 117 *The Hypothesis Rewrite rule is a safe sub-rule of the Transformation rule.*

7.3 Solving Negative Literals

If the atom corresponding to a negative literal of a goal is false for a certain inductive instantiation, then the goal is true for this instantiation. Thus, we only have to consider those instantiations of a goal which satisfy or *solve* the atoms corresponding to the negative literals. When we have a complete description of all those solutions by means of pairs (Λ, σ) consisting of a formula Λ and a substitution σ, such that for each inductive substitution τ that solves the atoms corresponding to the negative literals there is some (Λ, σ) of this description such that $\exists\varphi\colon\ \tau = \sigma\varphi$ and $\Lambda\tau$ is false, then we can transform the goal Γ into the sub-goals $(\Lambda\Gamma)\sigma$. An inference rule of this type will be said to *solve negative literals*.

In this section we present rules solving negative literals by incremental narrowing. These rules, namely the $\neq$- and $\overline{\text{Def}}$-Solve rules, utilize the initiality or freeness of the algebras that establish some specific notion of inductive validity, cf. section 1.1. In a certain sense, the $\neq$-Solve rule generalizes the $\neq$-Tautology Remove rule of section 5.2.

Rules for solving negative literals play an essential role in similar inference systems, cf. e.g. Bevers & Lewi (1990), Becker (1994), or Avenhaus & Madlener (1995). In our inference system they are only needed on the one hand for exploiting the incompleteness of the case analysis of the defining rules for a non-constructor function symbol (cf. Example 120) and on the other hand for showing the inequality of two pure constructor terms (cf. examples 122 and 123). The latter case is more generally applicable because it relies on the freeness of constructors only (no confusion in the constructor sub-universe) which is given in the C-, D'-, D-, and E-case, whereas the freeness requirements for the first case are only given in the D-, and E-case. The major role of the $\neq$-Solve rule in the above inference systems is played in our inference system by the more generally applicable Substitution Add instead, possibly followed by some Context Adds.

Note that our Solve rules below do not require termination of the rewrite relation given by the defining set of rules R. This is contrary to the inference systems cited above where termination is required — at least for effectiveness. The price we have to pay for this is that we have to require the negative literal to be linear. Even though our inference system can linearize non-linear literals (via Variable Add and Constant Rewrite), the linearity requirement makes it impossible to simulate some proofs that are possible without, cf. Example 124. Since we have left the induction ordering on an abstract level in the whole paper, we do not want to include the subject of practically establishing termination, especially not for the optimization of an inference rule that is of little importance for our inference system.

For the definition of NARROW(...,...,...) cf. Definition 80.

Let $\Gamma, \Delta \in \mathrm{Form}$; $s \in \mathbb{S}$; $t, t' \in \mathcal{T}_{\mathrm{SIG},s}$; $\aleph \in \mathrm{Weight}$.
Let L be a finite subset of 'Form', and H, G be finite subsets of 'SynCons'.

$\neq$-Solve:

$$\frac{(L \quad, H \quad, G \cup \{\ (\quad \Gamma(t{\neq}t')\Delta,\ \aleph) \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \}\ )}{(L \quad, H \quad, G \cup \{\ (\ \Lambda\Gamma(t{\neq}t')\Delta,\ \aleph)\,\sigma \mid (\Lambda,\sigma) \in \mathrm{NARROW}(t,t',\mathcal{V}(\Gamma(t{\neq}t')\Delta,\aleph)) \quad \}\ )}$$

if $(t{\neq}t')$ is linear and $\left(\begin{array}{ll} & \mathcal{V}(t{\neq}t') \cap (\mathcal{V}_<(\Gamma\Delta) \cup \mathcal{V}(\aleph)) = \emptyset \\ \vee & \text{the induction ordering is semantic} \end{array}\right)$.

$\overline{\mathrm{Def}}$-Solve:

$$\frac{(L \quad, H \quad, G \cup \{\ (\quad \Gamma\overline{(\mathrm{Def}\ t)}\Delta,\ \aleph) \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \}\ )}{(L \quad, H \quad, G \cup \{\ (\Lambda\Gamma\overline{(\mathrm{Def}\ t)}\Delta,\ \aleph)\,\sigma \mid (\Lambda,\sigma) \in \mathrm{NARROW}(x,t,\mathcal{V}(\Gamma(x{\neq}t)\Delta,\aleph)) \quad \}\ )}$$

if $x \in \mathrm{V}_{\mathrm{CONS},s} \backslash \mathcal{V}(\Gamma\overline{(\mathrm{Def}\ t)}\Delta,\aleph)$, t is linear, and $\left(\begin{array}{ll} & \mathcal{V}(t) \cap (\mathcal{V}_<(\Gamma\Delta) \cup \mathcal{V}(\aleph)) = \emptyset \\ \vee & \text{the induction ordering is semantic} \end{array}\right)$.

Lemma 118 *Assume either that the D- or E-case holds or otherwise that, for the literal $(t{\neq}t')$ of the $\neq$-Solve rule, $t, t' \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$ and* K *is a sub-class of the class of constructor-minimal models of* R. *Now:*
The $\neq$-Solve rule is a safe sub-rule of the Transformation rule.

Lemma 119 *Assume that the D- or E-case holds. Now:*
The Def*-Solve rule is a safe sub-rule of the Transformation rule.*

Note that the linearity requirement for the literal of $\neq$-Solve rule is necessary: Otherwise the following literals would be treated as valid (resulting in an unsound inference rule), which is not true, however: For R: $\mathsf{a}{=}\mathsf{f}(\mathsf{a})$ the literal $x \neq \mathsf{f}(x)$. For R: $\mathsf{a}{=}\mathsf{f}(\mathsf{a})$; $\mathsf{eq}(x,x){=}\mathsf{true}$ the literal $\mathsf{eq}(x,\mathsf{f}(x)) \neq \mathsf{true}$. For R: $\mathsf{a}{=}\mathsf{b}$; $\mathsf{b}{=}\mathsf{a}$; $\mathsf{q}(\mathsf{a},\mathsf{b}){=}\mathsf{true}$ the literal $\mathsf{q}(x,x) \neq \mathsf{true}$.

Moreover, note that in case that one of t, t', say t, is a variable $X \in \mathrm{V}_{\mathrm{SIG}}$, it is unwise to apply the $\neq$-Solve rule. Instead one should use Constant Rewrite to remove X from $\Gamma, \Delta, \aleph$ and then do a SIG-Variable Remove. Similarly, in case that one of t, t', say t, is a variable $x \in \mathrm{V}_{\mathrm{CONS}}$ and $t' \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$ it is also unwise to apply the $\neq$-Solve rule. Instead one should use Constant Rewrite to remove x from $\Gamma, \Delta, \aleph$ and then do a CONS-Variable Remove, followed by a $\overline{\mathrm{Def}}$-Literal Remove. For the same reason, in case of $t \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$ one should always apply the $\overline{\mathrm{Def}}$-Literal Remove instead of the $\overline{\mathrm{Def}}$-Solve rule.

Furthermore, note that if the literals are not linear (as required for the $\overline{\mathrm{Def}}$- and $\neq$-Solve rules), then this can always be achieved by introducing new aliases $x'_0, \ldots x'_{n-1} \in \mathrm{V}_{\varsigma,s}$ for each variable $x \in \mathrm{V}_{\varsigma,s}$ occurring $n{+}1$-times in the literal, then adding the literals $(x{\neq}x'_0) \ldots (x{\neq}x'_{n-1})$ by ς-Variable Add, and finally n Constant Rewrites for replacing the i^{th} occurrence of x in the original literal with x'_i.

Example 120 *(continuing Example 9)*
Let $X, Y \in \mathrm{V}_{\mathrm{SIG,nat}}$. *Let* $x, y \in \mathrm{V}_{\mathrm{CONS,nat}}$. *Assume that the induction ordering is semantic. Consider the formulas*

$$X - Y \neq 0, \quad X = Y$$

and

$$x - y \neq 0, \quad x = y$$

which are type-D and -E but not type-C or -D′ valid.
To prove the more general one, let us start with
(120.1) $X - Y \neq 0, \quad X = Y \quad ; \quad \mathsf{w}_{120}(X, Y).$
Now we can apply the $\neq$*-Solve rule to* $(X - Y \neq 0)$*, yielding:*
(120.1.1) $x - 0 \neq 0, \quad x = 0 \quad ; \quad \mathsf{w}_{120}(x, 0)$
(120.1.2) $\mathsf{s}(x) - \mathsf{s}(y) \neq 0, \quad \mathsf{s}(x) = \mathsf{s}(y) \quad ; \quad \mathsf{w}_{120}(\mathsf{s}(x), \mathsf{s}(y)).$
The first is transformed by a Lemma Rewrite into
(120.1.1.1) $x \neq 0, \quad x = 0 \quad ; \quad \mathsf{w}_{120}(x, 0)$
which is removed by $=$*-Decompose.*
The second is transformed by a Lemma Rewrite into
(120.1.2.1) $x - y \neq 0, \quad \mathsf{s}(x) = \mathsf{s}(y) \quad ; \quad \mathsf{w}_{120}(\mathsf{s}(x), \mathsf{s}(y)).$
Now a Hypothesis Apply with our original goal transforms this into
(120.1.2.1.1) $x \neq y, \quad x - y \neq 0, \quad \mathsf{s}(x) = \mathsf{s}(y) \quad ; \quad \mathsf{w}_{120}(\mathsf{s}(x), \mathsf{s}(y))$
(120.1.2.1.2) $\mathsf{w}_{120}(x, y) < \mathsf{w}_{120}(\mathsf{s}(x), \mathsf{s}(y)), \quad x - y \neq 0, \quad \mathsf{s}(x) = \mathsf{s}(y) \quad ; \quad \mathsf{w}_{120}(\mathsf{s}(x), \mathsf{s}(y))$
An $=$*-Decompose removes the goal (120.1.2.1.1) and the ordering literal of (120.1.2.1.2) can be easily solved.*

Note that the $\neq$-Solve is more powerful than a Substitution Add could have been. Even with the weaker version of the original formula that has constructor variables only, a Substitution Add would have produced a third goal, namely
(120.1.3) $0 - \mathsf{s}(y) \neq 0, \quad 0 = \mathsf{s}(y) \quad ; \quad \mathsf{w}_{120}(0, \mathsf{s}(y))$
which clearly exhibits why the original formula is only type-D and -E valid: The types C and D' must include the CONS:cons-term-generated constructor-minimal model with $0 - \mathsf{s}(y) = 0$ which falsifies this third goal.

Note that it is not the case that the proofs of lemmas involving functions with incomplete defining case distinctions usually include Solve rules. This would imply that theses lemmas were only known to be type-D and -E valid. Since we consider it to be important that the reader really becomes aware that type-C is a most appropriate notion of inductive validity we give another example:

Example 121 *(continuing Example 9)*
*Let us prove the lemma (*Def$-$*) of Example 10:*
(121.1) $\mathrm{Def}\,(x - y), \quad \mathsf{leq}(y, x) \neq \mathsf{true} \quad ; \quad \mathsf{w}_{121}(x, y)$
Substitution Add yields:
(121.1.1) $\mathrm{Def}\,(x - 0), \quad \mathsf{leq}(0, x) \neq \mathsf{true} \quad ; \quad \mathsf{w}_{121}(x, 0)$
(121.1.2) $\mathrm{Def}\,(x - \mathsf{s}(y)), \quad \mathsf{leq}(\mathsf{s}(y), x) \neq \mathsf{true} \quad ; \quad \mathsf{w}_{121}(x, \mathsf{s}(y))$
Lemma Rewrite with rule (−1) of Example 9 yields:
(121.1.1.1) $\mathrm{Def}\,x, \quad \mathsf{leq}(0, x) \neq \mathsf{true} \quad ; \quad \mathsf{w}_{121}(x, 0)$
which is removed by Def-*Decompose.*

Substitution Add to (121.1.2) yields:
(121.1.2.1) $\mathrm{Def}(0-\mathsf{s}(y)),\ \mathsf{leq}(\mathsf{s}(y),0)\neq\mathsf{true}\ ;\ \mathsf{w}_{121}(0,\mathsf{s}(y))$
(121.1.2.2) $\mathrm{Def}(\mathsf{s}(x)-\mathsf{s}(y)),\ \mathsf{leq}(\mathsf{s}(y),\mathsf{s}(x))\neq\mathsf{true}\ ;\ \mathsf{w}_{121}(\mathsf{s}(x),\mathsf{s}(y))$
Lemma Rewrite with the rule (leq*2) of Example 9 yields*
(121.1.2.1.1) $\mathrm{Def}(0-\mathsf{s}(y)),\ \mathsf{false}\neq\mathsf{true}\ ;\ \mathsf{w}_{121}(0,\mathsf{s}(y))$
which is removed as an $\neq$*-Tautology.*
Lemma Rewrite with the rule (leq*3) of Example 9 transforms (121.1.2.2) into*
(121.1.2.2.1) $\mathrm{Def}(\mathsf{s}(x)-\mathsf{s}(y)),\ \mathsf{leq}(y,x)\neq\mathsf{true}\ ;\ \mathsf{w}_{121}(\mathsf{s}(x),\mathsf{s}(y))$
Lemma Rewrite with the rule (–2) of Example 9 yields:
(121.1.2.2.1.1) $\mathrm{Def}(x-y),\ \mathsf{leq}(y,x)\neq\mathsf{true}\ ;\ \mathsf{w}_{121}(\mathsf{s}(x),\mathsf{s}(y))$
Finally, Hypothesis Apply of (121.1) yields
(121.1.2.2.1.1.1) $\mathsf{w}_{121}(x,y)<\mathsf{w}_{121}(\mathsf{s}(x),\mathsf{s}(y)),$
$\mathrm{Def}(x-y),\ \mathsf{leq}(y,x)\neq\mathsf{true}\ ;\ \mathsf{w}_{121}(\mathsf{s}(x),\mathsf{s}(y))$
whose ordering literal is trivially satisfiable.

Now let us see whether the $\neq$-Solve rule is powerful enough to prove those of the Peano Axioms that contain negative equations:

Example 122 *(continuing Example 9)*
Let us prove the lemma ($0\neq\mathsf{s}$*) of Example 10 assuming that the induction ordering is semantic:*
(122.1) $0\neq\mathsf{s}(x)\ ;\ \mathsf{w}_{122}(x)$
Since both 0 *and* $\mathsf{s}(x)$ *are pure constructor terms, we can apply the* $\neq$*-Solve rule without restricting to the D- or E-case. This removes the goal directly and finishes the proof.*

Example 123 *(continuing Example 9)*
Let us prove the lemma (Inject s*) of Example 10 assuming that the induction ordering is semantic:*
(123.1) $\mathsf{s}(x)\neq\mathsf{s}(y),\ x=y\ ;\ \mathsf{w}_{123}(x,y)$
Since both $\mathsf{s}(x)$ *and* $\mathsf{s}(y)$ *are pure constructor terms, we can apply the* $\neq$*-Solve rule without restricting to the D- or E-case. This results in the new goal*
(123.1.1) $\mathsf{s}(x)\neq\mathsf{s}(x),\ x=x\ ;\ \mathsf{w}_{123}(x,x)$
which is removed by an $=$*-Decompose.*

Now, while this was rather fine, the following example shows a shortcoming of our $\neq$-Solve rule:

Example 124 *(continuing Example 9)*

Let us try to prove the lemma (Irrefl s*) of Example 10 assuming that the induction ordering is semantic:*
(124.1) $x\neq\mathsf{s}(x)\ ;\ \mathsf{w}_{124}(x)$
Now since $x\neq\mathsf{s}(x)$ *is not linear, we cannot apply our* $\neq$*-Solve rule directly. We first have to do a* CONS*-Variable Add to yield*
(124.1.1) $y\neq x,\ x\neq\mathsf{s}(x)\ ;\ \mathsf{w}_{124}(x)$
and then a Constant Rewrite to yield

(124.1.1.1) $y \neq x,\ y \neq \mathsf{s}(x)\ \ ;\ \ \mathsf{w}_{124}(x)$

Now, since y is a constructor variable and $\mathsf{s}(x)$ is a pure constructor term we are in a situation of which we said above that it is unwise to apply an $\neq$-Solve. But let us be so silly as to do it:

(124.1.1.1.1) $\mathsf{s}(x) \neq x,\ \mathsf{s}(x) \neq \mathsf{s}(x)\ \ ;\ \ \mathsf{w}_{124}(x)$

Then an $\neq$-Literal Remove yields

(124.1[5]) $\mathsf{s}(x) \neq x\ \ ;\ \ \mathsf{w}_{124}(x)$

Now the goal (124.1[5]) equals the goal (124.1) which means that we have made no progress.

This means that our $\neq$-Solve rule is not more useful for proving the lemma (Irrefl s*) than the Constant Rewrite in Example 126 below. Nevertheless, we prove it in Example 125 and again in Example 126.*

Note that we could prove the lemma in a one step proof if the linearity restriction of the $\neq$-Solve rule were omitted. This can really be done when all instantiations of the terms of the $\neq$-literal are terminating, which is the case here with respect to R_9*. Cf. e.g. Avenhaus & Madlener (1995) for an $\neq$-Solve rule for the special case of termination. We will include such a rule when we come across a proof for which such a rule is necessary.*

Example 125 *(continuing Example 9)*

Let us again try to prove the lemma (Irrefl s*) of Example 10:*

(125.1) $x \neq \mathsf{s}(x)\ \ ;\ \ \mathsf{w}_{125}(x)$

A Substitution Add yields:

(125.1.1) $0 \neq \mathsf{s}(0)\ \ ;\ \ \mathsf{w}_{125}(0)$

(125.1.2) $\mathsf{s}(x) \neq \mathsf{s}(\mathsf{s}(x))\ \ ;\ \ \mathsf{w}_{125}(\mathsf{s}(x))$

An $\neq$-Tautology Remove deletes the goal (125.1.1).

A Lemma Apply of lemma (Inject s*) of Example 10 transforms (125.1.2) into*

(125.1.2.1) $x \neq \mathsf{s}(x),\ \mathsf{s}(x) \neq \mathsf{s}(\mathsf{s}(x))\ \ ;\ \ \mathsf{w}_{125}(\mathsf{s}(x))$

A Hypothesis Apply of (125.1) transforms this into the goal

(125.1.2.1.1) $\mathsf{w}_{125}(x) < \mathsf{w}_{125}(\mathsf{s}(x)),\ x \neq \mathsf{s}(x),\ \mathsf{s}(x) \neq \mathsf{s}(\mathsf{s}(x))\ \ ;\ \ \mathsf{w}_{125}(\mathsf{s}(x))$

whose ordering literal is trivially satisfiable.

Shorter than the proof above is the following proof for the same lemma:

Example 126 *(continuing Example 9)*

Let us again prove the lemma (Irrefl s*) of Example 10, now assuming the induction ordering to be semantic:*

(126.1) $x \neq \mathsf{s}(x)\ \ ;\ \ \mathsf{w}_{126}(x)$

A Constant Rewrite yields:

(126.1.1) $x \neq \mathsf{s}(x)\ \ ;\ \ \mathsf{w}_{126}(\mathsf{s}(x))$

Finally, Hypothesis Apply of (126.1) yields:

(126.1.1.1) $\mathsf{w}_{126}(x) < \mathsf{w}_{126}(\mathsf{s}(x)),\ x \neq \mathsf{s}(x)\ \ ;\ \ \mathsf{w}_{126}(\mathsf{s}(x))$

whose ordering literal is trivially satisfiable.

Finally we show how to use the $\overline{\text{Def}}$-Solve rule for showing the strictness of functions:

Example 127 *(continuing Example 9)*
Let $X, Y \in \mathrm{V}_{\mathrm{SIG},\mathsf{nat}}$. *Let* $x, y \in \mathrm{V}_{\mathrm{CONS},\mathsf{nat}}$. *Assuming the induction ordering to be semantic, let us prove the following goal, whose formula is type-D but not type-C valid.*

(127.1) $\overline{\text{Def}\,(X+Y)},\ \text{Def}\,X \ \ ;\ \ \mathsf{w}_{127}(X,Y)$

A $\overline{\text{Def}}$*-Solve yields:*

(127.1.1) $\overline{\text{Def}\,(x+\mathsf{0})},\ \text{Def}\,x \ \ ;\ \ \mathsf{w}_{127}(x,\mathsf{0})$

(127.1.2) $\overline{\text{Def}\,(x+\mathsf{s}(y))},\ \text{Def}\,x \ \ ;\ \ \mathsf{w}_{127}(x,\mathsf{s}(y))$

which are both removed by Def-*Decompose.*

7.4 Generalization

Contrary to deductive first order theorem proving, inductive theorem proving often is only successful when one tries to show stronger theorems than the ones one initially intended to show. This is because an inductive theorem is not only a task (as goal) but also a tool (as induction hypothesis) for the inductive argumentation. For this purpose we presented some generalizing rules in former versions of this paper. One of them was similar to the following:

Let $\Gamma, \Delta \in$ Form; $\lambda \in \mathcal{GENLIT}(\mathrm{sig}, \mathrm{V})$; $\aleph \in$ Weight.
Let L be a finite subset of 'Form', and H, G be finite subsets of 'SynCons':

Context Remove: $$\frac{(L \quad, H \quad, G \cup \{\ (\quad \Gamma\lambda\Delta,\ \aleph)\ \})}{(L \quad, H \quad, G \cup \{\ (\quad \Gamma\Delta,\ \aleph)\ \})}$$

Corollary 128 *The Context Remove rule is a sub-rule of the Transformation rule.*

In a certain sense the Context Remove rule is an antagonist of the Context Add rule. Similarly, we had an antagonist of the Substitution Add rule that replaced certain terms with new variables. Since it is the nature of these rules not to be safe, we did not give sub-cases in which they are safe. Due to this nature, they should be used rarely and only for preparation of new induction hypotheses. Since user supplied goals tend to be sufficiently general, most of the time these generalizing rules are only needed before starting a lemma proof or a mutual induction. Since an induction hypothesis is of no use without an appropriate weight, these generalizing rules also required a possibility to adjust the weight of the goal appropriately.

Before we are going to explain why we omit the generalizing rules in this version of the paper, we should try to make clear what we are talking about with the help of the following example:

Example 129 *(continuing Example 9)*
Let x and y be constructor variables of the sort nat*. Suppose we want to prove the lemma (True* p*) of Example 10:*
(129.1) $\mathsf{p}(x) = \mathsf{true} \ ; \ \mathsf{w}_1(x)$
First we use the Memorizing Rule of section 3.4 to copy (129.1) into the set of induction hypotheses. Then we replace (129.1) via Substitution Add with the following new goals:
(129.1.1) $\mathsf{p}(\mathsf{0}) = \mathsf{true} \ ; \ \mathsf{w}_1(\mathsf{0})$
(129.1.2) $\mathsf{p}(\mathsf{s}(x)) = \mathsf{true} \ ; \ \mathsf{w}_1(\mathsf{s}(x))$
*We remove (129.1.1) by Lemma Apply of (*p*1).*
*A Lemma Apply of (*p*2) (choosing* $m := 2$; $\lambda_0 := \mathsf{p}(x) \neq \mathsf{true}$; $\lambda_1 := \mathsf{q}(x, \mathsf{s}(x)) \neq \mathsf{true}$*) transforms (129.1.2) into the new goals:*
(129.1.2.1) $\mathsf{p}(x) = \mathsf{true},\ \mathsf{p}(\mathsf{s}(x)) = \mathsf{true} \ ; \ \mathsf{w}_1(\mathsf{s}(x))$
(129.1.2.2) $\mathsf{q}(x, \mathsf{s}(x)) = \mathsf{true},\ \mathsf{p}(\mathsf{s}(x)) = \mathsf{true} \ ; \ \mathsf{w}_1(\mathsf{s}(x))$
A Hypothesis Apply of (129.1) (choosing $m := 0$*) transforms (129.1.2.1) into*
(129.1.2.1.1) $\mathsf{w}_1(x) < \mathsf{w}_1(\mathsf{s}(x)),\ \mathsf{p}(x) = \mathsf{true},\ \mathsf{p}(\mathsf{s}(x)) = \mathsf{true} \ ; \ \mathsf{w}_1(\mathsf{s}(x))$
which we delay until the end of the proof where we will talk about the induction ordering.

In former versions of this paper we used a Context Remove (on $\mathsf{p}(\mathsf{s}(x))=\mathsf{true}$*) and a Substitution Remove (with* $\{y\mapsto\mathsf{s}(x)\}$*) including a weight manipulation to transform (129.1.2.2) into*

(129.2) $$\mathsf{q}(x,y)=\mathsf{true}\ ;\ \mathsf{w}_2(x,y)$$

Now, however, instead of transforming the goal (129.1.2.2) into the goal (129.2), we prefer to use the Expansion rule of section 3.4 to add (129.2) as a new goal and delay the treatment of (129.1.2.2).

After copying (129.2) into the set of induction hypotheses with the Memorizing Rule of section 3.4, a Substitution Add transforms (129.2) into the new goals:

(129.2.1) $$\mathsf{q}(x,0)=\mathsf{true}\ ;\ \mathsf{w}_2(x,0)$$

(129.2.2) $$\mathsf{q}(x,\mathsf{s}(y))=\mathsf{true}\ ;\ \mathsf{w}_2(x,\mathsf{s}(y))$$

A Lemma Apply of $(\mathsf{q}1)$ *removes (129.2.1).*

A Lemma Apply of $(\mathsf{q}2)$ *(choosing* $m:=2$; $\lambda_0:=\mathsf{q}(x,y)\neq\mathsf{true}$; $\lambda_1:=\mathsf{p}(x)\neq\mathsf{true}$*) transforms (129.2.2) into the new goals:*

(129.2.2.1) $$\mathsf{q}(x,y)=\mathsf{true},\ \mathsf{q}(x,\mathsf{s}(y))=\mathsf{true}\ ;\ \mathsf{w}_2(x,\mathsf{s}(y))$$

(129.2.2.2) $$\mathsf{p}(x)=\mathsf{true},\ \mathsf{q}(x,\mathsf{s}(y))=\mathsf{true}\ ;\ \mathsf{w}_2(x,\mathsf{s}(y))$$

A Hypothesis Apply of (129.2) (choosing $m:=0$*) transforms (129.2.2.1) into*

(129.2.2.1.1) $$\mathsf{w}_2(x,y)<\mathsf{w}_2(x,\mathsf{s}(y)),\ \mathsf{q}(x,y)=\mathsf{true},\ \mathsf{q}(x,\mathsf{s}(y))=\mathsf{true}\ ;\ \mathsf{w}_2(x,\mathsf{s}(y))$$

A Hypothesis apply of (129.1) (choosing $m:=0$*) transforms (129.2.2.2) into*

(129.2.2.2.1) $$\mathsf{w}_1(x)<\mathsf{w}_2(x,\mathsf{s}(y)),\ \mathsf{p}(x)=\mathsf{true},\ \mathsf{q}(x,\mathsf{s}(y))=\mathsf{true}\ ;\ \mathsf{w}_2(x,\mathsf{s}(y))$$

Now we have reached the first state of our proof attempt in which we know that the induction proof of (129.2) depends on the induction proof of (129.1). This means that now we know that a Lemma Apply of (129.2) to (129.1.2.2) will not lead to a successful completion of our proof attempt. Note that a Lemma Apply is usually the better choice since it does not require us to solve an additional ordering condition. Here, in our case, however, we now know that we have to use a Hypothesis Apply of (129.2) to transform (129.1.2.2) into the new goal

(129.1.2.2.1) $$\mathsf{w}_2(x,\mathsf{s}(x))<\mathsf{w}_1(\mathsf{s}(x)),\ \mathsf{q}(x,\mathsf{s}(x))=\mathsf{true},\ \mathsf{p}(\mathsf{s}(x))=\mathsf{true}\ ;\ \mathsf{w}_1(\mathsf{s}(x))$$

Gathering the ordering conditions from the remaining goals, i.e. (129.1.2.1.1), (129.1.2.2.1), (129.2.2.1.1), (129.2.2.2.1), we get

$$\mathsf{w}_1(x)<\mathsf{w}_1(\mathsf{s}(x))$$
$$\mathsf{w}_2(x,\mathsf{s}(x))<\mathsf{w}_1(\mathsf{s}(x))$$
$$\mathsf{w}_2(x,y)<\mathsf{w}_2(x,\mathsf{s}(y))$$
$$\mathsf{w}_1(x)<\mathsf{w}_2(x,\mathsf{s}(y))$$

which are all satisfied by the lexicographic path ordering where w_1 *and* w_2 *are equivalent in the precedence.*

This completes the proof. Note that if we had not adjusted the weight of (129.2) appropriately but would have gone on with $\mathsf{w}_1(x)$ *instead of* $\mathsf{w}_2(x,y)$*, then our proof attempt would not have been successful since the last two of the above four ordering conditions would be unsatisfiable.*

In comparison with the specialized generalizing rules of the former versions of this paper, the now recommended generalization by use of the Expansion rule instead, has the following advantages:

1. As exhibited in the above example, the choice whether to use a Lemma or a Hypothesis Apply to connect the intermediate sub-goal with the generalized new goal can be delayed until we know whether the cheaper and more widely applicable Lemma Apply will do or whether we have to use the Hypothesis Apply rule which requires us to solve an additional ordering condition.

   This means that our new solution (contrary to the one in the former versions) complies with our design goal of a "natural flow of information" of section 1.3.

2. A whole and always incomplete bunch of very complicated and difficult to apply generalizing sub-rules of the Transformation rule becomes superfluous.

3. We can restrict all Transformation steps to be safe. The only unsafe steps in our proofs will be the Expansion steps then. When we organize our proof attempts as a forest of trees where an Expansion adds a new root and a Transformation adds new sons, then we know the root to be invalid if one of its offspring is invalid (provided all Transformations are safe).

On the other hand, note that for using the Expansion rule for generalization in practice we really need something like a graph structure to organize our goals, including pointers for Lemma and Hypothesis Applies as well as for the intention to do such an Apply.

In former versions of this paper we also had some rules for *decomposition* similar to those called "inverse functionality" in section 6.3 of Walther (1994). These rules were generalizing (i.e. unsafe) in general. As indicated above, we do not want any unsafe rules besides the Expansion rule. Moreover, since most of our applications of these rules were safe actually, we have decided not to include the unsafe decomposition rules in the present version of this paper. Nevertheless, this former kind of decomposition is still possible:

In former versions of this paper we had a rule for =-decomposition that was unsafe, contrary to the one presented in this paper, cf. Lemma 84. The effect of this former rule can be achieved as follows:

Example 130
Suppose that we have a common top symbol 'f*' on both sides of an equation*
(130.1) $\mathsf{f}(t_0)=\mathsf{f}(t_1),\ \Gamma\ ;\ \aleph$
We can use an =-Decompose to yield
(130.1.1) $t_0=t_1,\ \mathsf{f}(t_0)=\mathsf{f}(t_1),\ \Gamma\ ;\ \aleph$
Finally, if really necessary, a Context Remove step could transform (130.1.1) into
(130.1.1.1) $t_0=t_1,\ \Gamma\ ;\ \aleph$
which is unsafe in general, so that we would prefer to do this step with the Expansion rule instead.
If we, however, have a lemma
$$x=y,\ \mathsf{f}(x)\neq\mathsf{f}(y)$$
saying that f *is injective, then we can do the step from (130.1.1) to (130.1.1.1) with an Applicative Literal Remove, which is safe.*

In former versions of this paper we also had a rule for Def-decomposition that was unsafe, contrary to the one presented in this paper, cf. Lemma 85. The effect of this former rule can be achieved as follows:

Example 131
Suppose that we have a constructor symbol 'c*' on top of a* Def*-literal*
(131.1) $\mathrm{Def}\,\mathsf{c}(t),\ \Gamma\ ;\ \aleph$
with[38] $t\notin\mathcal{T}(\mathrm{cons},\mathrm{V}_{\mathrm{CONS}})$. *Application of the* Def*-Decompose rule yields*
(131.1.1) $\mathrm{Def}\,t,\ \mathrm{Def}\,\mathsf{c}(t),\ \Gamma\ ;\ \aleph$
Finally, if really necessary, a Context Remove step could transform the (131.1.1) into
(131.1.1.1) $\mathrm{Def}\,t,\ \Gamma\ ;\ \aleph$
which is unsafe in general, so that we would prefer to do this step with the Expansion rule instead.
If we, however, have a lemma ($X\in\mathrm{V}_{\mathrm{SIG}}$)
(131.2) $\mathrm{Def}\,X,\ \overline{\mathrm{Def}\,\mathsf{c}(X)}$
then we can do the step from (131.1.1) to (131.1.1.1) with an Applicative Literal Remove, which is safe. Note, however, that (131.2) will never[39] *be type-C valid. Nevertheless, when all unifiers between* $\mathsf{c}(X)$ *and the left-hand sides of* R *instantiate* X *with a pure constructor term* c_i*, then a* $\overline{\mathrm{Def}}$*-Solve can transform (131.2) into*
(131.2.i) $\mathrm{Def}\,c_i,\ \overline{\mathrm{Def}\,\mathsf{c}(c_i)}$
which then can be removed by application of the Def*-Decompose rule.*

[38] otherwise we had better apply the Def-Decompose rule to remove the goal (131.1) immediately
[39] unless the sort of X is necessarily trivial

8 A Concrete Failure Predicate

In section 3.10 we have discussed the notions and requirements for an abstract failure predicate 'FAIL'. The simplest non-trivial sound failure predicate is given by defining $F \in \mathrm{FAIL}_0$ if $\emptyset \in F$, i.e. a prover state is a failure state if it contains the empty formula (or clause, sequent) (i.e. a formula with zero literals) as a lemma, hypothesis, or goal. Together with our Literal Remove rules from section 5.3 and the Variable Remove rules from section 6.2 this turns out to be surprisingly useful because the other rules in the inference system can help to find the failure.

Example 132 *(continuing examples 9 and 10)*

Suppose we have the goal (where $x, y \in \mathrm{V}_{\mathrm{CONS},\mathsf{nat}}$ *and* $\aleph \in \mathrm{Weight}$*)*
(132.1) $y \neq \mathsf{0},\ x = y\ ;\ \aleph$
An $\neq$*-Solve on the first literal transforms this into*
(132.1.1) $\mathsf{0} \neq \mathsf{0},\ x = \mathsf{0}\ ;\ \aleph\{y \mapsto \mathsf{0}\}$
An $\neq$*-Literal Remove yields*
(132.1.1.1) $x = \mathsf{0}\ ;\ \aleph\{y \mapsto \mathsf{0}\}$
A Substitution Add yields the two goals
(132.1.1.1.1) $\mathsf{0} = \mathsf{0}\ ;\ \aleph\{y \mapsto \mathsf{0},\ x \mapsto \mathsf{0}\}$
(132.1.1.1.2) $\mathsf{s}(x) = \mathsf{0}\ ;\ \aleph\{y \mapsto \mathsf{0},\ x \mapsto \mathsf{s}(x)\}$
An $=$*-Decompose removes the goal (132.1.1.1.1). An Applicative Literal Remove*[40] *with the lemma* $(\mathsf{0} \neq \mathsf{s})$ *of Example 10 transforms (132.1.1.1.2) into the empty goal*
(132.1.1.1.2.1) $;\ \aleph\{y \mapsto \mathsf{0},\ x \mapsto \mathsf{s}(x)\}$

Now (132.1.1.1.2.1) is a failure state w.r.t. FAIL_0*. Moreover, since* FAIL_0 *is sound and all applied inference rules are safe*[41] *we know that our original state (132.1) is invalid (cf. Corollary 61), i.e. it contains an invalid lemma, hypotheses, or goal. Since the lemmas can be restricted to the proved lemma* $(\mathsf{0} \neq \mathsf{s})$*, the Acquisition rule is safe,*[42] *and hypotheses are not involved, we know that the goal of (132.1) is invalid. Moreover, following the instantiations of the above proof, we can explain why this goal is invalid.*

Let us see what happens when we replace the constructor variables with non-constructor variables:

Example 133 *(continuing examples 9 and 10)*

Suppose we have the goal (where $X, Y \in \mathrm{V}_{\mathrm{SIG},\mathsf{nat}}$ *and* $\aleph \in \mathrm{Weight}$*)*
(133.1) $Y \neq \mathsf{0},\ X = Y\ ;\ \aleph$
Since an $\neq$*-Solve on the first literal is only possible in the D- or E-case (cf. Lemma 118), we now start with a Constant Rewrite*
(133.1.1) $Y \neq \mathsf{0},\ X = \mathsf{0}\ ;\ \aleph$
followed by a SIG*-Variable Remove yielding*
(133.1.1.1) $X = \mathsf{0}\ ;\ \aleph$

[40] Note that it is more straightforward to use the $=$-Literal Remove rule here, but below we want to discuss what to do when lemmas are involved.
[41] cf. Lemma 118, Corollary 88, Lemma 107, and Lemma 89
[42] cf. Corollary 58

Covering sets for a non-constructor variable are never[43] *useful for proving a goal because they have to contain a renaming substitution. Therefore they have not been discussed before. Nevertheless, they can help to discover a failure: A Substitution Add with the covering set of substitutions* $\{X \mapsto X\}$, $\{X \mapsto \mathsf{s}(\mathsf{0})\}$ *transforms (133.1.1.1) into*

(133.1.1.1.1) $X = \mathsf{0} \;\; ; \;\; \aleph$

(133.1.1.1.2) $\mathsf{s}(\mathsf{0}) = \mathsf{0} \;\; ; \;\; \aleph\{X \mapsto \mathsf{s}(\mathsf{0})\}$

Now an =-Literal Remove yields an empty goal again:

(133.1.1.1.2.1) $; \;\; \aleph\{X \mapsto \mathsf{s}(\mathsf{0})\}$

Now (133.1.1.1.2.1) is a failure state w.r.t. FAIL_0. *Moreover, since* FAIL_0 *is sound and all applied inference rules are safe*[44] *we know that our original state (133.1) is invalid. Since neither lemmas nor hypotheses have been involved, we know that the goal of (133.1) is invalid. Moreover, following the instantiations of the above proof, we can explain why this goal is invalid.*

In Avenhaus & Madlener (1995) one can find more sophisticated failure predicates which, however, are not more powerful[45] but only more convenient.

[43] provided that the sort of the non-constructor variable is non-trivial

[44] cf. Lemma 92, Lemma 101, Lemma 107, and Lemma 90

[45] unless the linearization required for our Solve rules makes them inferior to the analogous construction of Avenhaus & Madlener (1995) which is partly more powerful when the defining rules are known to be terminating. If we add special Solve rules for terminating R, however, this inferiority disappears because the linearity requirement can be dropped in case of a terminating R.

9 Main Results

In the following theorem we summarize our main results. The numbers on the right-hand side indicate the lemmas or corollaries that supply the corresponding result. These numbers are also useful for finding the definition of the relevant inference rule because this definition always directly precedes the indicated lemmas or corollaries.

Theorem 134

Assume the following:

- R *is a* Def-*MCRS over* sig/cons/V. $\longrightarrow_{R,\emptyset}$ *is confluent.* *62*
- *The B′-, C-, D′-, D-, or E-case of Definition 63 holds.*[46]
- *The global requirements 67, 68, and 70 of section 4.2 concerning the induction ordering are satisfied.*

Now, w.r.t. the definitions 64 and 65 of counterexamples and (inductive) validity, the following rules are safe sub-rules of the Transformation rule:

=- *and* Def-*Decompose* *84, 85*
Multiple-, ≠-, and $\overline{\text{Def}}$-*Literal Remove* *87, 88*
Applicative Literal Remove *87, 89*
Constant Rewrite *92*
Lemma Apply *93*
Lemma Rewrite *96*
Context Add *100*
SIG- *and* CONS-*Variable Remove* *101, 102*
SIG- *and* CONS-*Variable Add* *103, 104*
Substitution Add *107*
Hypothesis Apply *113*
Hypothesis Rewrite *117*

In addition, the following rules are safe sub-rules of the Transformation rule in the C-, D′-, D-, or E-case:

≠-Tautology Remove *86*

=-Literal Remove in case that the removed =-literal consists of pure constructor terms *87, 90*

≠-Solve in case in case that the solved ≠-literal consists of pure constructor terms *118*

Finally, the following are safe sub-rules of the Transformation rule in the D- or E-case:

=- *and* Def-*Literal Remove* *87, 90, 91*
≠- *and* $\overline{\text{Def}}$-*Solve* *118, 119*

[46] This implies that Global Requirement 66 is satisfied.

10 Outlook

Before coming to a conclusion we would like to point out the problems we have not sufficiently solved or covered yet but will study in the future:

Induction Ordering: An important aspect that is not included in this version of our inference system is a concrete set of possible types of orderings and inference rules for each of these types. So far our inference system only generates a set of ordering conditions in the form of a set of formulas (open goals).

Guidance: When designing our inference system we were not interested in a minimal calculus that admits an easy automatic control or guidance. Nevertheless, tactics and even whole strategies for finding proofs have to be developed for a useful prover based on our inference system.

Existential Variables: In the present version of our inference system we do not have to instantiate our hypotheses before they become useful for the proof. In this respect our approach is superior to the existential variables used in Walther (1992) to improve the hypotheses application of Boyer & Moore (1979). The unnatural flow of information in case of the γ-rule[47], however, (for which the premature instantiation of the hypotheses in Boyer & Moore (1979) is an example) may still occur in some proofs with our inference system when we apply a lemma or hypothesis and the matching substitution does not instantiate all of its variables (cf. e.g. the instantiation of the variable y in Example 114). We hope to remove this shortcoming in a future version of our inference system by adding free existential variables in a restricted form[48].

Universal Framework for Induction: Moreover, we hope that one day we will be able to represent the proofs of all the other known first-order-based inductive inference systems in terms of our prover formulas and our inference rules, in order to be able to represent the whole tactic and strategic knowledge on inductive theorem proving in a homogeneous unified framework. We are still far away from this goal, though. E.g. the possible treatment of existential properties or variables as described in Padawitz (1992) has not been included yet in the present version of our inference system.

[47] Cf. section 1.3

[48] such that the independence and self-descriptiveness of our prover formulas (cf. section 1.3) can be preserved

11 Conclusion

After describing why we consider explicit induction to be a special case of implicit induction (cf. also Example 116), we captured the idea of implicit inductive theorem proving on an abstract level. On this level we explained the advantages of explicit weights and of the "non-switched" framework for inference systems for inductive theorem proving. Moreover, this abstract level already provides us with the inference rules of our inference system: The unsafe Expansion rule for starting a proof attempt for a new lemma, the safe Memorizing rule which just copies a goal into the set of hypotheses, the safe Acquisition rule which makes lemmas and defining rules available for the proof, and the Transformation rule. In the rest of the paper we have described a concrete instance of our abstract frame, refined the Transformation rule by exhibiting two dozens of safe sub-rules of it, and described a simple failure predicate.

This concrete instance of the abstract frame is conceptually interesting: It does not require termination of the defining positive/negative-conditional equational rules, such that verification in necessarily non-terminating domains like program execution becomes possible. Beyond that, our constructor-based framework admits an adequate treatment of partially specified functions in the simple and clear framework of inductive validities of Wirth & Gramlich (1994b). Note that partial specification of functions is of practical importance because, even in the cases where an effective completion is possible, the over-specification resulting from a completion of the function definitions imposes many problems in practice.

Since, however abstract our treatment may appear, we have never lost sight of the practical aspect and always have given it preference to theoretical elegance, we have reason to expect that our inference system is well-suited as a kernel of a practically useful inductive theorem prover. We hope to give evidence of this with a system we are implementing according to the ideas elaborated in section 1, which we would like to recommend for a concluding second reading.

Appendix A Refutational Completeness

The definitions of section 3.10 are essential for this section.

For achieving refutational completeness we need a wellfounded ordering $>_{\text{refut}}$ on finite sets[49] of syntactic constructs.

To guarantee refutation of invalid initial states the following properties are appropriate.

Definition 135 (FAIL-**Completeness**)
The failure predicate FAIL is *complete w.r.t.* $\vdash$ *and* $>_{\text{refut}}$ if for all finite sets $L \subseteq$ Form; $H, G \subseteq$ SynCons; if form$[G]$ is invalid, but (L, H, G) is not a failure state, then there are finite sets L', H', G' with $(L, H, G) \vdash^{+} (L', H', G')$ and $G > _{\text{refut}} G'$.

By wellfoundedness of $>_{\text{refut}}$ we immediately get:

Corollary 136 (Non-Deterministic Refutational Completeness)
Let $L \subseteq$ Form*;* $H, G \subseteq$ SynCons *be finite sets. Assume either that* $\vdash$ *is sound or that*[50] L *is valid,* $H \subseteq G$*, and* $\vdash$ *is inductively sound. Furthermore assume* FAIL *to be complete w.r.t.* $\vdash$ *and* $>_{\text{refut}}$*. Now, if* form$[G]$ *is invalid, then there is some failure state* (L', H', G') *w.r.t.* FAIL *with* $(L, H, G) \vdash^{*} (L', H', G')$.

Definition 137 (Fairness)
Let β be an ordinal number with $\beta \preceq \omega$. Let $L_i \subseteq$ Form; $H_i, G_i \subseteq$ SynCons for all $i \prec 1+\beta$. Consider the derivation $(L_i, H_i, G_i) \vdash (L_{i+1}, H_{i+1}, G_{i+1})$ $(i \prec \beta)$. It is called *fair w.r.t.* FAIL if

$$\left(\begin{array}{ll} & \exists i \prec 1+\beta\text{: } G_i = \emptyset \\ \vee & \left(\begin{array}{ll} & \beta \prec \omega \\ \wedge & (L_\beta, H_\beta, G_\beta) \notin \text{dom}(\vdash)\ ^{51} \\ \wedge & \text{FAIL is complete w.r.t. } \vdash \text{ and } >_{\text{refut}} \end{array} \right) \\ \vee & \forall i \prec 1+\beta\text{: } \left(\left(\begin{array}{ll} & \text{form}[G_i] \text{ is invalid} \\ \wedge & (L_i, H_i, G_i) \text{ is no failure state} \end{array} \right) \Rightarrow \exists j \prec 1+\beta\text{: } G_i > _{\text{refut}} G_j \right) \end{array} \right)$$

Corollary 138 ([Deterministic] Refutational Completeness)
Let β *be an ordinal number with* $\beta \preceq \omega$ *. Let* $L_i \subseteq$ Form*;* $H_i, G_i \subseteq$ SynCons *be finite sets for all* $i \prec 1+\beta$*. Let* $(L_i, H_i, G_i) \vdash (L_{i+1}, H_{i+1}, G_{i+1})$ $(i \prec \beta)$ *be a fair derivation w.r.t.* FAIL*. Assume either that* $\vdash$ *is sound or that* L_0 *is valid,* $H_0 \subseteq G_0$*, and* $\vdash$ *is inductively sound. Now, if* form$[G]$ *is invalid, there must exist some* $i \prec 1+\beta$ *such that* (L_i, H_i, G_i) *is a failure state w.r.t.* FAIL*.*

---

[49] Note that for practical reasons it may be convenient to define $>_{\text{refut}}$ on lists of syntactic constructs or some other structural augmentation to a finite set of syntactic constructs in order to satisfy the fairness defined below more locally. Nevertheless, since we do not know how to describe this structural augmentation abstractly, we just suppose that $>_{\text{refut}}$ is defined on finite sets of syntactic constructs.

[50] In this second case we also know that the goals never become valid: Otherwise Corollary 32(5) implies inductiveness of the current state which implies inductiveness of the initial state, and then Corollary 51 implies validity of the original set of goals which contradicts the following assumption.

[51] i.e. no inference rule can be applied to $(L_\beta, H_\beta, G_\beta)$

Appendix B A Sequent Calculus for Deductive Validity

Similar to Gentzen (1935), we partition V into *bound* and *free* variables because this makes the distinction of bound and free occurrences of variables as well as the procedure of applying a substitution most simple. Beyond this, we then partition the free variables again into *universal* and *existential* variables: For each $(\varsigma, s) \in \{\mathrm{SIG}, \mathrm{CONS}\}\times\mathbb{S}$ we assume $\mathrm{V}_{\varsigma,s} = \mathrm{V}_{\mathrm{bound},\varsigma,s} \uplus \mathrm{V}_{\mathrm{free},\varsigma,s}$ and $\mathrm{V}_{\mathrm{free},\varsigma,s} = \mathrm{V}_{\forall,\varsigma,s} \uplus \mathrm{V}_{\exists,\varsigma,s}$ such that all three sets $\mathrm{V}_{\mathrm{bound},\varsigma,s}$, $\mathrm{V}_{\forall,\varsigma,s}$, and $\mathrm{V}_{\exists,\varsigma,s}$ are infinite. We define $(Q \in \{\mathrm{bound}, \mathrm{free}, \forall, \exists\};\ \varsigma \in \{\mathrm{SIG}, \mathrm{CONS}\})$: $\mathrm{V}_Q := (\mathrm{V}_{Q,\varsigma,s})_{(\varsigma,s)\in\{\mathrm{SIG},\mathrm{CONS}\}\times\mathbb{S}}$; $\mathrm{V}_{Q,\varsigma} := (\mathrm{V}_{Q,\varsigma,s})_{s\in\mathbb{S}}$; $\mathrm{V}_{Q,s} := (\mathrm{V}_{Q,\varsigma,s})_{\varsigma\in\{\mathrm{SIG},\mathrm{CONS}\}}$.

Let '$\mathcal{AT}$' be the set of *atoms* containing the set $\mathcal{AT}(\mathrm{sig}, \mathrm{V}_{\mathrm{free}})$ (cf. section 2.6) and possibly other predicate atoms with fixed arity over terms from $\mathcal{T}(\mathrm{sig}, \mathrm{V}_{\mathrm{free}})$. Let $\mathcal{LIT} := \{\ A,\ \overline{A} \mid A \in \mathcal{AT}\ \}$ be the set of *literals*.

Still proceeding similar to Gentzen (1935), let the set of formulas F be the smallest set satisfying: If $A \in \mathcal{AT}$ then $A \in \mathrm{F}$. If $A, B \in \mathrm{F}$, then $(A\vee B)$, $(A\wedge B)$, $(A\Rightarrow B)$, $\overline{A}$ are all formulas in F. Furthermore, if $A \in \mathrm{F}$, then $\forall x B$ and $\exists x B$ are formulas in F for $(\varsigma, s) \in \{\mathrm{SIG}, \mathrm{CONS}\}\times\mathbb{S}$, $x \in \mathrm{V}_{\mathrm{bound},\varsigma,s}\backslash\mathcal{V}(A)$, $y \in \mathrm{V}_{\mathrm{free},\varsigma,s}$, and $B = A\{y\mapsto x\}$. By "$A\{x\mapsto t\}$" we denote the result of substituting each occurrence of the variable x in A with the term t. Note that we never use $A\{x\mapsto t\}$ when A has a subsequence of the form "$\exists x$" or "$\forall x$".

A *sequent* is a list of formulas, i.e., an element of F^*. We define an equivalence relation on sequents of the same length: A sequent $A_0, \ldots, A_{m-1}$ is called a *variant* of the sequent $B_0, \ldots, B_{n-1}$ if $m = n$ and for all $i \prec m$ either $A_i = B_i$ or else A_i and B_i are both atoms of the same predicate symbol such that A_i and B_i are congruent (i.e. their top level terms are) w.r.t. $R := \{\ (s,t) \mid \exists j \prec n\colon (\overline{(s{=}t)} = A_j \wedge \mathcal{V}(s,t)\subseteq\mathrm{V}_\forall)\ \}$. Furthermore, for $A_i = (s{=}t)$ we may even always require $B_i = (s'{=}t)$ with s being congruent to s' w.r.t. to R. Note that it is important for our approach that $A_i \neq B_i$ implies that A_i and B_i are atoms and no negative =- or Def-literals which play a special role in our system.

The *axioms* of the sequent calculus we want to present are the variants of the sequents of the following form (where $\Gamma, \Delta, \Pi \in \mathrm{F}^*$; A is an atom with predicate symbol different from '=' and 'Def' and with $\mathcal{V}(A)\subseteq\mathrm{V}_\forall$; $t \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall)$; $u \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\forall,\mathrm{CONS}})$; $n \in \mathbb{N}$; $p_0, \ldots, p_{n-1} \in \mathcal{POS}(u)$ mutually parallel; $v_0, \ldots, v_{n-1} \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall)$):

$$\Gamma\ A\ \Delta\ \overline{A}\ \Pi$$

$$\Gamma\ \overline{A}\ \Delta\ A\ \Pi$$

$$\Gamma\ (t{=}t)\ \Pi$$

And if $\forall i \prec n$: $\overline{(\mathrm{Def}\, v_i)}$ is contained in $\Gamma\Pi$:

$$\Gamma\ (\mathrm{Def}\, u[\, p_i \leftarrow v_i \mid i \prec n\,])\ \Pi$$

The rules of our sequent calculus are the following ($\Gamma, \Delta, \Pi \in \mathrm{F}^*$; $A, B, C \in \mathrm{F}$):

α/β-rules:

$$\frac{\Gamma\ \overline{\overline{A}}\ \Pi}{\Gamma\ A\ \Pi}$$

α-rules:

$$\frac{\Gamma\ (A \vee B)\ \Pi}{\Gamma\ A\ B\ \Pi} \qquad \frac{\Gamma\ \overline{(A \wedge B)}\ \Pi}{\Gamma\ \overline{A}\ \overline{B}\ \Pi} \qquad \frac{\Gamma\ (A \Rightarrow B)\ \Pi}{\Gamma\ \overline{A}\ B\ \Pi}$$

β-rules:

$$\frac{\Gamma\ \overline{(A \vee B)}\ \Pi}{\Gamma\ \overline{A}\ \Pi \qquad \Gamma\ \overline{B}\ \Pi} \qquad \frac{\Gamma\ (A \wedge B)\ \Pi}{\Gamma\ A\ \Pi \qquad \Gamma\ B\ \Pi} \qquad \frac{\Gamma\ \overline{(A \Rightarrow B)}\ \Pi}{\Gamma\ A\ \Pi \qquad \Gamma\ \overline{B}\ \Pi}$$

if $\mathcal{V}_\exists(A) \cap \mathcal{V}_\exists(B) = \emptyset$.

γ-rules:

$$\frac{\Gamma\ \exists x B\ \Pi}{\Gamma\ B\{x \mapsto y\}\ \Pi} \qquad \frac{\Gamma\ \overline{\forall x B}\ \Pi}{\Gamma\ \overline{B\{x \mapsto y\}}\ \Pi}$$

where for some $(\varsigma, s) \in \{\mathrm{SIG}, \mathrm{CONS}\} \times \mathbb{S}$:
$x \in \mathrm{V}_{\mathrm{bound},\varsigma,s}$ and $y \in \mathrm{V}_{\exists,\varsigma,s} \backslash \mathcal{V}(B)$.

δ-rules:

$$\frac{\Gamma\ \overline{\exists x B}\ \Pi}{\Gamma\ \overline{B\{x \mapsto y\}}\ \Pi} \qquad \frac{\Gamma\ \forall x B\ \Pi}{\Gamma\ B\{x \mapsto y\}\ \Pi}$$

where for some $(\varsigma, s) \in \{\mathrm{SIG}, \mathrm{CONS}\} \times \mathbb{S}$:
$x \in \mathrm{V}_{\mathrm{bound},\varsigma,s}$ and $y \in \mathrm{V}_{\forall,\varsigma,s} \backslash \mathcal{V}(\Gamma B \Pi)$ and $\mathcal{V}_\exists(B) = \emptyset$.

Instantiation-rule:

$$\frac{\Gamma\ C\ \Pi}{\Gamma\ C\{x \mapsto t\}\ C\ \Pi}$$

where for some $(\varsigma, s) \in \{\mathrm{SIG}, \mathrm{CONS}\} \times \mathbb{S}$: $x \in \mathrm{V}_{\exists,\varsigma,s}$ and $t \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_{\mathrm{free}})_s$ and

$$\left(\begin{array}{ll} & x \in \mathrm{V}_{\mathrm{SIG}} \\ \vee & \exists \left\{ \begin{array}{l} u \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{free},\mathrm{CONS}}) \\ n \in \mathbb{N} \\ p_0, \ldots, p_{n-1} \in \mathcal{POS}(u) \\ v_0, \ldots, v_{n-1} \in \mathcal{T} \end{array} \right\} : \left(\begin{array}{ll} & t = u[\, p_i \leftarrow v_i \mid i \prec n \,] \\ \wedge & \forall i \preceq n\colon\ \Gamma C \Pi \text{ contains } \overline{(\mathrm{Def}\ v_i)} \end{array} \right) \end{array} \right).$$

The *Restricted Instantiation-rule* is like the Instantiation-rule but with the following restrictions: $t \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall)_s$; $u \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\forall,\mathrm{CONS}})$; $v_0, \ldots, v_{n-1} \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall)$; and either

- C is a literal and $x \in \mathcal{V}_\exists(C)$;
- C is of one of the forms $\overline{(A \vee B)}$, $(A \wedge B)$, $\overline{(A \Rightarrow B)}$, and $x \in \mathcal{V}_\exists(A) \cap \mathcal{V}_\exists(B)$; or
- C is of one of the forms $\overline{\exists x B}$, $\forall x B$, and $x \in \mathcal{V}_\exists(B)$.

Cut-rule:

$$\frac{\Gamma}{A\,\Gamma \qquad \overline{A}\,\Gamma}$$

This completes our sequent calculus. Note that we do not need the Cut rule and the non-restricted Instantiation-rule for deductive completeness, cf. Theorem 144. Thus, while we could omit the Cut-rule, we do not have inference rules for equality, which is similar to the systems of Wang (1960), Kanger (1963), and Lifschitz (1971).

A *proof tree* for a sequent Γ is a tree such that Γ is the label of the root and the labels of the sons of each node exactly result from applying an inference step of our sequent calculus to the label of this (father) node. A branch of a proof tree is called *closed* if it contains a node labeled with an axiom. A branch is called *fair* if it is closed or each α/β-, α-, β-, γ-, δ-, and Restricted Instantiation-rule application that is possible for some sequent is done for some descendent of this sequent in the branch below. Note that this fairness is easily established for α/β, α, β, γ, and δ. Since the Restricted Instantiation-rule has to be applied for each $t \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall)_s$ satisfying the condition of the rule, some bookkeeping using an enumeration of the terms of $\mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall)_s$ is required. Furthermore, note that, for the β- and Restricted[52] Instantiation-rule, in the branch below a sequent to which they are applicable, the condition keeps being satisfied and the principal literal keeps being present until the rule for the literal (and the term t) is applied. Moreover, the γ- and δ-rule keep being applicable too, possibly for different y. A proof tree is *fair* (or else *closed*) if all its branches that start from the root and do not end before a leaf is reached are fair (or else closed). If there is a closed proof tree for Γ, then it describes a proof for Γ in our sequent calculus and we say that Γ is *provable*.

[52] Note that the unrestricted Instantiation-rule may lose its principal literal C by an application of a β- or δ-rule.

Definition 139 (Validity of a Sequent in a sig/cons-Structure)

A sig/cons-structure $\mathcal{A}$ is an extension of a sig/cons-algebra such that for all predicate symbols P of 'sig' with arity $s_0 \dots s_{m-1}$ we have $P^{\mathcal{A}} \subseteq \mathcal{A}_{\mathrm{SIG},s_0} \times \dots \times \mathcal{A}_{\mathrm{SIG},s_{m-1}}$.

Let $\mathrm{X} \subseteq \mathrm{V}_{\mathrm{free}}$; $\mathcal{A}$ be a sig/cons-structure; and $\kappa \in \mathcal{SUB}(\mathrm{X}, \mathcal{A})$.

An atom $(u{=}v) \in \mathcal{AT}(\mathrm{sig}, \mathrm{V}_{\mathrm{free}})$ is *true w.r.t.* $\mathcal{A}_\kappa$ if $\mathcal{V}(u,v) \subseteq \mathrm{X}$ and $\mathcal{A}_\kappa(u) = \mathcal{A}_\kappa(v)$; and an atom $(\mathrm{Def}\, u) \in \mathcal{AT}(\mathrm{sig}, \mathrm{V}_{\mathrm{free}})$ (with $u \in \mathcal{T}_{\mathrm{SIG},s}$; $s \in \mathbb{S}$) is *true w.r.t.* $\mathcal{A}_\kappa$ if $\mathcal{V}(u) \subseteq \mathrm{X}$ and $\mathcal{A}_\kappa(u) \in \mathcal{A}_{\mathrm{CONS},s}$;
$Pt_0 \dots t_{m-1}$ is *true w.r.t.* $\mathcal{A}_\kappa$ if $\mathcal{V}(t_0, \dots, t_{m-1}) \subseteq \mathrm{X}$ and $(\mathcal{A}_\kappa(t_0), \dots, \mathcal{A}_\kappa(t_{m-1})) \in P^{\mathcal{A}}$.
An atom $(u{=}v) \in \mathcal{AT}(\mathrm{sig}, \mathrm{V})$ is *false w.r.t.* $\mathcal{A}_\kappa$ if $\mathcal{V}(u,v) \subseteq \mathrm{X}$ and $\mathcal{A}_\kappa(u) \neq \mathcal{A}_\kappa(v)$; and an atom $(\mathrm{Def}\, u) \in \mathcal{AT}(\mathrm{sig}, \mathrm{V})$ (with $u \in \mathcal{T}_{\mathrm{SIG},s}$; $s \in \mathbb{S}$) is *false w.r.t.* $\mathcal{A}_\kappa$ if $\mathcal{V}(u) \subseteq \mathrm{X}$ and $\mathcal{A}_\kappa(u) \notin \mathcal{A}_{\mathrm{CONS},s}$;
$Pt_0 \dots t_{m-1}$ is *false w.r.t.* $\mathcal{A}_\kappa$ if $\mathcal{V}(t_0, \dots, t_{m-1}) \subseteq \mathrm{X}$ and $(\mathcal{A}_\kappa(t_0), \dots, \mathcal{A}_\kappa(t_{m-1})) \notin P^{\mathcal{A}}$.

Let $A, B \in \mathrm{F}$. The following table defines being true or false w.r.t. $\mathcal{A}_\kappa$ by a complete case distinction:

A	B	$\overline{A}$	$(A \vee B)$	$(A \wedge B)$	$(A \Rightarrow B)$
false	false	true	false	false	true
false	true	true	true	false	true
true	false	false	true	false	false
true	true	false	true	true	true

Let $A \in \mathrm{F}$, $(\varsigma, s) \in \{\mathrm{SIG}, \mathrm{CONS}\} \times \mathbb{S}$, $x \in \mathrm{V}_{\mathrm{bound},\varsigma,s} \setminus \mathcal{V}(A)$, $y \in \mathrm{V}_{\mathrm{free},\varsigma,s}$, and $B = A\{y \mapsto x\}$. Define $\mathrm{Y} := \mathcal{V}_{\mathrm{free}}(B)$.

$\exists x B$ is *true* (or else *false*) *w.r.t.* $\mathcal{A}_\kappa$ if
A is true (or else false) w.r.t. $\mathcal{A}_{\kappa'}$ for some (or else all) $\kappa' \in \mathcal{SUB}(\mathrm{Y} \uplus \{y\}, \mathcal{A})$ with $\kappa'|_{\mathrm{Y}} \subseteq \kappa$.
$\forall x B$ is *true* (or else *false*) *w.r.t.* $\mathcal{A}_\kappa$ if
A is true (or else false) w.r.t. $\mathcal{A}_{\kappa'}$ for all (or else some) $\kappa' \in \mathcal{SUB}(\mathrm{Y} \uplus \{y\}, \mathcal{A})$ with $\kappa'|_{\mathrm{Y}} \subseteq \kappa$.

A sequent $\Gamma \in \mathrm{F}^*$ is *true w.r.t.* $\mathcal{A}_\kappa$ if
$\mathcal{V}_{\mathrm{free}}(\Gamma) \subseteq \mathrm{X}$ and there is some literal λ in Γ that is true w.r.t. $\mathcal{A}_\kappa$.
A sequent $\Gamma \in \mathrm{F}^*$ is *false w.r.t.* $\mathcal{A}_\kappa$ if all literals λ in Γ are false w.r.t. $\mathcal{A}_\kappa$.

A sequent $\Gamma \in \mathrm{F}^*$ is *valid in* $\mathcal{A}$ if
$\forall \kappa \in \mathcal{SUB}(\mathcal{V}_\forall(\Gamma), \mathcal{A})\colon\ \exists \varepsilon \in \mathcal{SUB}(\mathcal{V}_\exists(\Gamma), \mathcal{A})\colon\ (\Gamma \text{ is true w.r.t. } \mathcal{A}_{\kappa \cup \varepsilon})$.

A sequent is *deductively valid* if it is valid in all sig/cons-structures $\mathcal{A}$.

Corollary 140 *The inference rules of our sequent calculus are sound in the sense that all provable sequents are deductively valid. Moreover, they are safe in the sense that a sequent that is not deductively valid at a node in a proof tree implies that the sequent at the root of this tree is not deductively valid.*

Lemma 141 (Substitution-Lemma for Formulas)
Let $\mathrm{X} \subseteq \mathrm{V}_{\mathrm{free}}$. *Let* $\mathcal{A}$ *be a* sig/cons*-structure. Let* $\kappa \in \mathcal{SUB}(\mathrm{X}, \mathcal{A})$.
Let $\sigma \in \mathcal{GENSUB}(\mathrm{V}_{\mathrm{free}}, \mathcal{T}(\mathrm{V}_{\mathrm{free}}))$. *Let* $A \in \mathrm{F}$ *with* $\mathcal{V}_{\mathrm{free}}(A\sigma) \subseteq \mathrm{X}$.
Now the following two are logically equivalent:

- $A\sigma$ *is true w.r.t.* $\mathcal{A}_\kappa$.
- A *is true w.r.t.* $\mathcal{A}_{\sigma \mathcal{A}_\kappa}$.

Definition 142 (Dual Hintikka Sets)
A *dual Hintikka set* DHS is a subset of the set of formulas F such that the following conditions hold:

α/β**-Closed:** If $\overline{\overline{A}} \in \mathrm{DHS}$, then $A \in \mathrm{DHS}$.

α**-Closed:**
- If $(A \vee B) \in \mathrm{DHS}$, then $A \in \mathrm{DHS}$ and $B \in \mathrm{DHS}$.
- If $\overline{(A \wedge B)} \in \mathrm{DHS}$, then $\overline{A} \in \mathrm{DHS}$ and $\overline{B} \in \mathrm{DHS}$.
- If $(A \Rightarrow B) \in \mathrm{DHS}$, then $\overline{A} \in \mathrm{DHS}$ and $B \in \mathrm{DHS}$.

β**-Closed:** For all A and B with $\mathcal{V}_\exists(A) \cap \mathcal{V}_\exists(B) = \emptyset$:
- If $\overline{(A \vee B)} \in \mathrm{DHS}$, then $\overline{A} \in \mathrm{DHS}$ or $\overline{B} \in \mathrm{DHS}$.
- If $(A \wedge B) \in \mathrm{DHS}$, then $A \in \mathrm{DHS}$ or $B \in \mathrm{DHS}$.
- If $\overline{(A \Rightarrow B)} \in \mathrm{DHS}$, then $A \in \mathrm{DHS}$ or $\overline{B} \in \mathrm{DHS}$.

γ**-Closed:** For all B, for all $(\varsigma, s) \in \mathbb{S}$, and for all $x \in \mathrm{V}_{\mathrm{bound},\varsigma,s}$, there is some $y \in \mathrm{V}_{\exists,\varsigma,s} \backslash \mathcal{V}(B)$ such that:
- If $\exists x B \in \mathrm{DHS}$, then $B\{x \mapsto y\} \in \mathrm{DHS}$.
- If $\overline{\forall x B} \in \mathrm{DHS}$, then $\overline{B\{x \mapsto y\}} \in \mathrm{DHS}$.

δ**-Closed:** For all B with $\mathcal{V}_\exists(B) = \emptyset$, for all $(\varsigma, s) \in \mathbb{S}$, and for all $x \in \mathrm{V}_{\mathrm{bound},\varsigma,s}$, there is some $t \in \mathcal{T}(\mathrm{V}_{\mathrm{free}})_{\varsigma,s}$ such that:
- If $\overline{\exists x B} \in \mathrm{DHS}$, then $\overline{B\{x \mapsto t\}} \in \mathrm{DHS}$.
- If $\forall x B \in \mathrm{DHS}$, then $B\{x \mapsto t\} \in \mathrm{DHS}$.

Instantiation-Closed: For all $s \in \mathbb{S}$, for all $x \in \mathrm{V}_{\exists,s}$, for all $t \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall)_s$, for all $C \in \mathrm{DHS}$, we have $C\{x \mapsto t\} \in \mathrm{DHS}$

if $\left(\begin{array}{ll} & x \in \mathrm{V}_{\mathrm{SIG}} \\ \vee & \exists \left\{ \begin{array}{l} u \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\forall,\mathrm{CONS}}) \\ n \in \mathbb{N} \\ p_0, \ldots, p_{n-1} \in \mathcal{POS}(u) \\ v_0, \ldots, v_{n-1} \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall) \end{array} \right\} : \left(\begin{array}{ll} & t = u[\, p_i \leftarrow v_i \mid i \prec n \,] \\ \wedge & \forall i \preceq n\colon\ \overline{(\mathrm{Def}\, v_i)} \in \mathrm{DHS} \end{array} \right) \end{array} \right)$ and:
- C is a literal and $x \in \mathcal{V}_\exists(C)$;
- C is of one of the forms $\overline{(A \vee B)}$, $(A \wedge B)$, $\overline{(A \Rightarrow B)}$, and $x \in \mathcal{V}_\exists(A) \cap \mathcal{V}_\exists(B)$; or
- C is of one of the forms $\overline{\exists x B}$, $\forall x B$, and $x \in \mathcal{V}_\exists(B)$.

Leibniz-Substitutable: For all $l, r \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall)$ with $\overline{(l{=}r)} \in \mathrm{DHS}$ or $\overline{(r{=}l)} \in \mathrm{DHS}$:
- For each atom A with predicate symbol different from '=' and with $\mathcal{V}_\exists(A) = \emptyset$ and $A/q = l$: If $A \in \mathrm{DHS}$, then $A[\, q \leftarrow r \,] \in \mathrm{DHS}$.
- For each $s, t \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall)$ with $s/q = l$:
  If $(s = t) \in \mathrm{DHS}$, then $(s[\, q \leftarrow r \,] = t) \in \mathrm{DHS}$.

Reflexive-Consistent: For each $t \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall)$: $(t{=}t) \notin \mathrm{DHS}$.

Def**-Consistent:** For all $n \in \mathbb{N}$; $u \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\forall,\mathrm{CONS}})$; $p_0, \ldots, p_{n-1} \in \mathcal{POS}(u)$; $v_0, \ldots, v_{n-1} \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall)$; with $\forall i \prec n\colon\ \overline{(\mathrm{Def}\, v_i)} \in \mathrm{DHS}$: $(\mathrm{Def}\, u[\, p_i \leftarrow v_i \mid i \prec n \,]) \notin \mathrm{DHS}$.

Predicate-Consistent: There is no $A \in \mathcal{AT}$ with predicate symbol different from '=' and 'Def' and with $\mathcal{V}_\exists(A) = \emptyset$ such that $A \in \mathrm{DHS}$ and $\overline{A} \in \mathrm{DHS}$.

Lemma 143
For each dual Hintikka set DHS *there is some* sig/cons-*structure* $\mathcal{A}$ *and some* $\iota \in \mathcal{SUB}(\mathrm{V}_\forall, \mathcal{A})$ *such that* $\forall A \in$ DHS: $\forall \varepsilon \in \mathcal{SUB}(\mathrm{V}_\exists, \mathcal{A})$: ($A$ *is false w.r.t.* $\mathcal{A}_{\iota \cup \varepsilon}$).

Theorem 144
A sequent is provable in the above sequent calculus iff it is deductively valid.
Moreover, this still holds when we omit the Cut-rule and replace the Instantiation-rule with the Restricted Instantiation rule.

Appendix C The Proofs

Proof of Lemmas 2 and 3

For Lemma 2 one shows $\forall s \in \mathbb{S}$: $\forall t \in \mathcal{T}(\mathrm{X})_{\mathrm{SIG},s}$: $h_s(\mathcal{B}_\kappa(t)) = \mathcal{C}_{\kappa h}(t)$ by structural induction on t. Lemma 3 is just a corollary of Lemma 2 when one realizes that $t\mu = \mathcal{T}(\mathrm{X})_\mu(t)$. Then we get $\mathcal{A}_\kappa(t\mu) = \mathcal{A}_\kappa(\mathcal{T}(\mathrm{X})_\mu(t)) = \mathcal{A}_{\mu\mathcal{A}_\kappa}(t)$. **Q.e.d. (Lemmas 2 and 3)**

Proof of Lemma 12

It is sufficient to treat the case of λ being an atom.

We show the first equivalence first. For an equality atom the following are logically equivalent due to Lemma 3: $(u{=}v)\mu$ is true w.r.t. $\mathcal{A}_\kappa$; $\mathcal{A}_\kappa(u\mu) = \mathcal{A}_\kappa(v\mu)$; $\mathcal{A}_{\mu\mathcal{A}_\kappa}(u) = \mathcal{A}_{\mu\mathcal{A}_\kappa}(v)$; $(u{=}v)$ is true w.r.t. $\mathcal{A}_{\mu\mathcal{A}_\kappa}$. Similarly, for a definedness atom the following are logically equivalent for s being the sort of u: $(\mathrm{Def}\, u)\mu$ is true w.r.t. $\mathcal{A}_\kappa$; $\mathcal{A}_\kappa(u\mu) \in \mathcal{A}_{\mathrm{CONS},s}$; $\mathcal{A}_{\mu\mathcal{A}_\kappa}(u) \in \mathcal{A}_{\mathrm{CONS},s}$; $(\mathrm{Def}\, u)$ is true w.r.t. $\mathcal{A}_{\mu\mathcal{A}_\kappa}$.

Finally we show that, under the additional assumptions, $\lambda\mu$ is true w.r.t. $\mathcal{A}_\kappa$ iff $\lambda'\mu$ is true w.r.t. $\mathcal{A}_\kappa$. Since λ rewrites to λ' with $l{=}r$, there is some $P \subseteq \mathcal{POS}(\lambda)$ such that $\forall p \in P$: $\lambda/p = l$ and $\lambda' = \lambda[\, p \leftarrow r \mid p \in P\,]$. Choose an arbitrary $x \in \mathrm{V}_{\mathrm{SIG},s} \backslash \mathcal{V}(\lambda)$ for s being the sort of l. For $u \in \mathcal{T}_{\mathrm{SIG},s}$ define ν_u by $(y \in \mathrm{V})$: $y\nu_u := \left\{ \begin{array}{ll} u\mu & \text{if } y = x \\ y\mu & \text{otherwise} \end{array} \right\}$. Due to $\mathcal{A}_\kappa(l\mu) = \mathcal{A}_\kappa(r\mu)$ we get $\nu_l\mathcal{A}_\kappa = \nu_r\mathcal{A}_\kappa$. By our first equivalence shown above the following are logically equivalent: $\lambda[\, p \leftarrow x \mid p \in P\,]\nu_l$ is true w.r.t. $\mathcal{A}_\kappa$; $\lambda[\, p \leftarrow x \mid p \in P\,]$ is true w.r.t. $\mathcal{A}_{\nu_l\mathcal{A}_\kappa}$; $\lambda[\, p \leftarrow x \mid p \in P\,]$ is true w.r.t. $\mathcal{A}_{\nu_r\mathcal{A}_\kappa}$; $\lambda[\, p \leftarrow x \mid p \in P\,]\nu_r$ is true w.r.t. $\mathcal{A}_\kappa$. This actually finishes the proof due to $\lambda\mu = \lambda[\, p \leftarrow l \mid p \in P\,]\mu = \lambda\mu[\, p \leftarrow l\mu \mid p \in P\,] = \lambda\nu_l[\, p \leftarrow x\nu_l \mid p \in P\,] = \lambda[\, p \leftarrow x \mid p \in P\,]\nu_l$ and $\lambda[\, p \leftarrow r \mid p \in P\,]\mu = \lambda\mu[\, p \leftarrow r\mu \mid p \in P\,] = \lambda\nu_r[\, p \leftarrow x\nu_r \mid p \in P\,] = \lambda[\, p \leftarrow x \mid p \in P\,]\nu_r$.

Q.e.d. (Lemma 12)

Proof of Lemma 16
By the restriction of Definition 6 on the constructor rules, $\longrightarrow_{\mathrm{R,X},\omega}$ is just the standard closure over a finitary relation.

Proof of Lemma 17

We claim the following:

1. $\forall \mathrm{Y} \subseteq \mathrm{V}\colon\ \forall s \in \mathcal{T}(\mathrm{cons}, \mathrm{Y})\colon\ \forall t\colon\ (s \xrightarrow{*}_{\mathrm{R,X},\omega} t\ \Rightarrow\ t \in \mathcal{T}(\mathrm{cons}, \mathrm{Y}))$
2. $\forall i \in \mathbb{N}\colon\ \forall n \in \mathbb{N}\colon\ (\xrightarrow{n}_{\mathrm{R,X},\omega+i} \cap (\mathcal{T}(\mathrm{cons}, \mathrm{V_{SIG}} \uplus \mathrm{V_{CONS}}) \times \mathcal{T})) \subseteq \xrightarrow{n}_{\mathrm{R,X},\omega}$
3. $\forall i \in \mathbb{N}\colon\ \longrightarrow_{\mathrm{R,X},\omega} \subseteq \longrightarrow_{\mathrm{R,X},\omega+i} \subseteq \longrightarrow_{\mathrm{R,X},\omega+i+1}$
4. $\longrightarrow_{\mathrm{R,X}}$ satisfies the requirement of Lemma 17; i.e. $\longrightarrow_{\mathrm{R,X}} \in S$.
5. $\longrightarrow_{\mathrm{R,X},\omega} \subseteq \bigcap S$
6. $\forall \rightsquigarrow \in S\colon\ \forall n \in \mathbb{N}\colon\ (\ \overset{n}{\rightsquigarrow} \cap (\mathcal{GT}(\mathrm{cons}) \times \mathcal{T})\ ) \subseteq \xrightarrow{n}_{\mathrm{R,X},\omega}$
7. $\forall i \in \mathbb{N}\colon\ \longrightarrow_{\mathrm{R,X},\omega+i} \subseteq \bigcap S$
8. $\longrightarrow_{\mathrm{R,X}}$ is the minimum $\bigcap S \in S$.
9. For $\emptyset \neq S' \subseteq S$: $\bigcap S' \in S$.

To the proofs of these claims:

1. By the restriction of Definition 6 on the constructor rules.
2. By induction on i using Lemma 16, (1), and the restriction of Definition 6 on the constructor rules.
3. $\underline{i = 0\text{:}}$ By definition of $\longrightarrow_{\mathrm{R,X},\omega}$. $\underline{i \Rightarrow (i{+}1)\text{:}}$ By (2).
4. The first requirement follows directly from (2). The second follows from (3), taking $\longrightarrow_{\mathrm{R,X},\omega+i+1}$ on the left-hand side of the second requirement for the maximum i of all $\longrightarrow_{\mathrm{R,X},\omega+i}$ occurring positively (i.e. not in a $\nmid$-statement) in the fulfilledness condition (expanded via Definition 14) of the right-hand side of the second requirement.
5. By Lemma 16, $\longrightarrow_{\mathrm{R,X},\omega}$ is the intersection of a superset of S.
6. By induction on n using $(\ \rightsquigarrow \cap (\mathcal{GT}(\mathrm{cons}) \times \mathcal{T})\ ) \subseteq \longrightarrow_{\mathrm{R,X},\omega}$ and (1).
7. $\underline{i = 0\text{:}}$ By (5). $\underline{i \Rightarrow (i{+}1)\text{:}}$ By (6) and (3).
8. By (4) and (7).
9. By (6) and (5).

Q.e.d. (Lemma 17)

Proof of Lemma 20
Follows from items (2) and (1) of the Proof of Lemma 17.

Proof of Lemma 21
Follows from Lemma 20.

Proof of Lemma 22
Follows from item (3) of the Proof of Lemma 17 where monotonicity of fulfilledness additionally needs Lemma 21 in case of a negative condition.

Proof of Lemma 23
By the Axiom of Choice there is some $\tau \in \mathcal{SUB}(\mathrm{Y}, \mathcal{T}(\mathrm{X}))$ with $\tau|_{\mathrm{X}} \subseteq \mathrm{id}$. Using this τ in combination with Corollary 19 for getting rid of variables from $\mathrm{Y} \backslash \mathrm{X}$ (introduced by extra-variables), the first sentence can be shown by induction on β and the rest is trivial.

Proof of Theorem 25

Let $\mathcal{A} := \mathcal{T}(\mathrm{X})/\overset{*}{\longleftrightarrow}_{\mathrm{R},\mathrm{X}}$. Let $\mathcal{I} := \mathcal{GT}/\overset{*}{\longleftrightarrow}_{\mathrm{R},\emptyset}$.

Claim 1: If $\mathcal{C}$ is a sig/cons-model of R; $\mu \in \mathcal{SUB}(\mathrm{X}, \mathcal{C})$; then

$$\forall \beta \preceq \omega\colon\ \forall s \in \mathbb{S}\colon\quad \longrightarrow_{\mathrm{R},\mathrm{X},\beta} \cap\, (\mathcal{T}_{\mathrm{SIG},s} \times \mathcal{T}_{\mathrm{SIG},s}) \quad\subseteq\quad \ker(\mathcal{C}_\mu)_s\ .$$

The proof of Claim 1 is omitted because it reads just like the proof of Claim 2 until it comes to $\beta \succ \omega$.

Proof of Part *(1)* of the theorem:
By the Axiom of Choice, each element of $\mathcal{SUB}(\mathrm{V}, \mathcal{A})$ can be written $\sigma\mathcal{A}_\iota$ for some $\sigma \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}))$. Thus, by Lemma 12, for $\mathcal{A}$ being a sig/cons-model of R it is sufficient to note that for $((l, r), C) \in \mathrm{R}$; $\sigma \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}))$; we have:

$$\text{If } \forall u, v \in \mathcal{T}\colon \left(\begin{array}{llll} & (\ ((\ u{=}v\ ) \text{ in } C\sigma) & \Rightarrow & u \overset{*}{\longleftrightarrow}_{\mathrm{R},\mathrm{X}} v\ ) \\ \wedge & (\ ((\mathrm{Def}\, u) \text{ in } C\sigma) & \Rightarrow\ \exists \hat{u} \in \mathcal{T}(\mathrm{X})_{\mathrm{CONS},s}\colon & u \overset{*}{\longleftrightarrow}_{\mathrm{R},\mathrm{X}} \hat{u}\ ) \\ \wedge & (\ ((\ u{\neq}v\ ) \text{ in } C\sigma) & \Rightarrow & u \overset{*}{\not\longleftrightarrow}_{\mathrm{R},\mathrm{X}} v\ ) \end{array} \right), \text{ then } C\sigma$$

is fulfilled w.r.t. $\longrightarrow_{\mathrm{R},\mathrm{X}}$.
This is due to confluence of $\longrightarrow_{\mathrm{R},\mathrm{X}}$, $\mathrm{X} \subseteq \mathrm{V}_{\mathrm{SIG}}$ (i.e. $\mathcal{T}(\mathrm{X})_{\mathrm{CONS},s} = \mathcal{GT}_{\mathrm{CONS},s}$), Lemma 20, and the fact that R is a Def-MCRS.
Now, for the proof that $\mathcal{A}$ is a *constructor-minimum* model, suppose $\mathcal{C}$ to be a sig/cons-model of R. We have to find a cons-homomorphism from $\mathcal{A}|_{\mathbb{C}\cup(\{\mathrm{CONS}\}\times\mathbb{S})}$ to $\mathcal{C}|_{\mathbb{C}\cup(\{\mathrm{CONS}\}\times\mathbb{S})}$. Let $\mathcal{B}$ be the ground term algebra over cons. Due to $\mathrm{X} \subseteq \mathrm{V}_{\mathrm{SIG}}$ there is a cons-homomorphism $h::\mathcal{A}|_{\mathbb{C}\cup(\{\mathrm{CONS}\}\times\mathbb{S})} \to (\mathcal{B}/(\overset{*}{\longleftrightarrow}_{\mathrm{R},\mathrm{X}} \cap (\mathcal{GT}(\mathrm{cons}) \times \mathcal{GT}(\mathrm{cons}))))$ given by ($s \in \mathbb{S}$; $A \in \mathcal{A}_{\mathrm{CONS},s}$): $A \mapsto \mathcal{GT}(\mathrm{cons}) \cap A$. Thus we only have to find a cons-homomorphism from $\mathcal{B}/(\overset{*}{\longleftrightarrow}_{\mathrm{R},\mathrm{X}} \cap (\mathcal{GT}(\mathrm{cons}) \times \mathcal{GT}(\mathrm{cons})))$ to $\mathcal{C}|_{\mathbb{C}\cup(\{\mathrm{CONS}\}\times\mathbb{S})}$. Using the Homomorphism-Theorem (the usual one for cons-homomorphisms, not ours) all we have to show is $\forall s \in \mathbb{S}$: $\overset{*}{\longleftrightarrow}_{\mathrm{R},\mathrm{X}} \cap\, (\mathcal{GT}_{\mathrm{CONS},s} \times \mathcal{GT}_{\mathrm{CONS},s}) \subseteq \ker(\mathcal{C})_s$, which by confluence of $\longrightarrow_{\mathrm{R},\mathrm{X}}$ is the same as $\forall s \in \mathbb{S}$: $\downarrow_{\mathrm{R},\mathrm{X}} \cap\, (\mathcal{GT}_{\mathrm{CONS},s} \times \mathcal{GT}_{\mathrm{CONS},s}) \subseteq \ker(\mathcal{C})_s$. Because of Lemma 20, $\forall s \in \mathbb{S}$: $\longrightarrow_{\mathrm{R},\mathrm{X},\omega} \cap\, (\mathcal{GT}_{\mathrm{CONS},s} \times \mathcal{GT}_{\mathrm{CONS},s}) \subseteq \ker(\mathcal{C})_s$ is sufficient for this. But this is implied by Claim 1.
Q.e.d. (Part *(1)* of the theorem)

Claim 2: If $\mathcal{C} \in$ K; $\mu \in \mathcal{SUB}(\mathrm{X},\mathcal{C})$; $\longrightarrow_{\mathrm{R},\emptyset}$ is confluent; then
$$\forall \beta \preceq \omega{+}\omega\colon\ \forall s \in \mathbb{S}\colon\quad \longrightarrow_{\mathrm{R},\mathrm{X},\beta} \cap\, (\mathcal{T}_{\mathrm{SIG},s} \times \mathcal{T}_{\mathrm{SIG},s}) \quad\subseteq\quad \ker(\mathcal{C}_\mu)_s\ .$$
Proof of Claim 2: For the limit ordinals 0, ω, $\omega{+}\omega$ the induction step is trivial. For a non-limit ordinal $\beta{+}1$ the induction step is as follows: Suppose $s \longrightarrow_{\mathrm{R},\mathrm{X},\beta+1} t$. Then there are $((l,r),C) \in \mathrm{R}$; $\sigma \in \mathcal{SUB}(\mathrm{V},\mathcal{T}(\mathrm{X}))$; $p \in \mathcal{POS}(s)$; with $s/p = l\sigma$; $t = s[\,p \leftarrow r\sigma\,]$; and $C\sigma$ fulfilled w.r.t. $\longrightarrow_{\mathrm{R},\mathrm{X},\beta}$. As $\mathcal{C}$ is a sig/cons-model of R, for inferring $\mathcal{C}_\mu(s) = \mathcal{C}_\mu(t)$ (by Lemma 12) we only have to show that L is true w.r.t. $\mathcal{C}_\mu$ for each literal L in the condition $C\sigma$. Three cases:
If $L = (u{=}v)$, by $u{\downarrow}_{\mathrm{R},\mathrm{X},\beta} v$ the induction hypothesis implies $\mathcal{C}_\mu(u) = \mathcal{C}_\mu(v)$.
If $L = (\mathrm{Def}\, u)$ with $u \in \mathcal{T}_{\mathrm{SIG},s'}$, by the existence of some $\hat{u} \in \mathcal{GT}_{\mathrm{CONS},s'}$ with $u \overset{*}{\longrightarrow}_{\mathrm{R},\mathrm{X},\beta} \hat{u}$ the induction hypothesis implies $\mathcal{C}_\mu(u) = \mathcal{C}_\mu(\hat{u}) \in \mathcal{C}_{\mathrm{CONS},s'}$.
If $L = (u{\neq}v)$, by the existence of some $s \in \mathbb{S}$, $\hat{u},\hat{v} \in \mathcal{GT}_{\mathrm{CONS},s}$ with $u \overset{*}{\longrightarrow}_{\mathrm{R},\mathrm{X},\beta} \hat{u} {\not\downarrow}_{\mathrm{R},\mathrm{X},\beta} \hat{v} \overset{*}{\longleftarrow}_{\mathrm{R},\mathrm{X},\beta} v$ the induction hypothesis implies $\mathcal{C}_\mu(u) = \mathcal{C}_\mu(\hat{u}) = \mathcal{C}(\hat{u})$ and $\mathcal{C}(\hat{v}) = \mathcal{C}_\mu(\hat{v}) = \mathcal{C}_\mu(v)$. Thus, for $\mathcal{C}_\mu(u) \neq \mathcal{C}_\mu(v)$ it is sufficient to show $\mathcal{C}(\hat{u}) \neq \mathcal{C}(\hat{v})$. By $\omega \preceq \beta$ and Lemma 22 we know $\hat{u} {\not\downarrow}_{\mathrm{R},\mathrm{X},\omega} \hat{v}$. By Lemma 21 $\hat{u} {\not\downarrow}_{\mathrm{R},\mathrm{X}} \hat{v}$. Then by Lemma 23 $\hat{u} {\not\downarrow}_{\mathrm{R},\emptyset} \hat{v}$; and therefore (by confluence of $\longrightarrow_{\mathrm{R},\emptyset}$) $\hat{u} \overset{*}{\not\longleftrightarrow}_{\mathrm{R},\emptyset} \hat{v}$. Because of part *(1)* of the theorem (matching its X to $\emptyset$), $\mathcal{C}$ must be not only a constructor-minimal model but also a constructor-minimum model of R, just like $\mathcal{I}$. Thus there is some cons-homomorphism $h{::}\mathcal{C}|_{\mathbb{C}} \to \mathcal{I}|_{\mathbb{C}}$. Now, using the Homomorphism-Lemma for cons-homomorphisms (i.e. the analogue of Lemma 2), due to $\hat{u},\hat{v} \in \mathcal{GT}_{\mathrm{CONS},s}$ we get $h_s(\mathcal{C}|_{\mathbb{C}}(\hat{u})) = \mathcal{I}|_{\mathbb{C}}(\hat{u}) \neq \mathcal{I}|_{\mathbb{C}}(\hat{v}) = h_s(\mathcal{C}|_{\mathbb{C}}(\hat{v}))$, which implies $\mathcal{C}|_{\mathbb{C}}(\hat{u}) \neq \mathcal{C}|_{\mathbb{C}}(\hat{v})$.
Q.e.d. (Claim 2)

Claim 3: If $\longrightarrow_{\mathrm{R},\emptyset}$ is confluent, then $\mathcal{A}$ is free for K over X w.r.t. ι.
Proof of Claim 3: Suppose $\mathcal{C} \in$ K and μ to be a $\mathcal{C}$-valuation of X. The uniqueness of the required sig/cons-homomorphism $h{::}\mathcal{A} \to \mathcal{C}$ with $\mu = \iota h$ is trivial. For its existence (by the Homomorphism-Theorem(4)) we only have to show
$$\forall s \in \mathbb{S}\colon\quad \overset{*}{\longleftrightarrow}_{\mathrm{R},\mathrm{X}} \cap\, (\mathcal{T}_{\mathrm{SIG},s} \times \mathcal{T}_{\mathrm{SIG},s}) \quad\subseteq\quad \ker(\mathcal{C}_\mu)_s\ ,$$
which is implied by Claim 2. Q.e.d. (Claim 3)

Claim 4: If $\longrightarrow_{\mathrm{R},\mathrm{X}}$ is confluent, then $\longrightarrow_{\mathrm{R},\emptyset}$ is confluent, too.
Proof of Claim 4: Trivial by Lemma 23. Q.e.d. (Claim 4)

Proof of Part *(2)* of the theorem: Since $\mathcal{A} \in$ K by part *(1)* of the theorem, this is implied by the claims 4 and 3.

Proof of Part *(3)* of the theorem: Suppose $\mathcal{C}$ to be a sig/cons-model of R with $\mathcal{C} \lesssim_{\mathrm{H}} \mathcal{A}$. Then $\mathcal{C} \lesssim_{\mathrm{CONS}} \mathcal{A}$. By part *(1)* of the theorem we get $\mathcal{C} \in$ K, and then by part *(2)* of the theorem $\mathcal{A} \lesssim_{\mathrm{H}} \mathcal{C}$.

Q.e.d. (Theorem 25)

Proof of Lemma 45
The only rule whose soundness is non-trivial is the Deletion rule for form$[G]$ being valid. By Corollary 32(2) we get $G \searrow \emptyset$. Thus by Corollary 36(2) and the condition of the rule we get $\{S\} \searrow H$. By Corollary 44 we may assume that $(L, H, G \cup \{S\})$ is weakly correct. Thus (since $G \cup \{S\} \searrow H$ due to Corollary 35(1)) we can conclude that form$[H]$ must be valid. By Corollary 32(7) this means that form(S) is valid, i.e. that form$[G \cup \{S\}]$ is valid.
Q.e.d. (Lemma 45)

Proof of Lemma 47
Assume L to be valid. We show invariance of weak correctness first: Expansion: Assume $G \cup \{S\} \searrow H$. By Corollary 35(2) we get $G \searrow H$. Now form$[H]$ must be valid due to weak correctness of (L, H, G). Hypothesizing: Assume $G \searrow H \cup \{S\}$. By Lemma 53 we get $G \searrow H$. By weak correctness of (L, H, G), form$[H]$ must be valid, which by Corollary 32(7) means that form$[G \cup H]$ is valid. By the condition of the rule and Corollary 32(6), we know that form(S) is valid. Therefore form$[H \cup \{S\}]$ is valid, too. Acquisition: Trivial. Deletion: Assume $G \searrow H$. By Lemma 53 we get $G \cup \{S\} \searrow H$. By weak correctness of $(L, H, G \cup \{S\})$, form$[H]$ must be valid.

Finally we show invariance of correctness: Expansion: By Corollary 35(2). Hypothesizing: By Corollary 37 we get $G \curvearrowright G$. If we assume $H \curvearrowright G$, we get $H \cup G \curvearrowright G$ by Corollary 35(1). By the condition of the rule and Corollary 37 this means $\{S\} \curvearrowright G$. By our assumption we now get $H \cup \{S\} \curvearrowright G$ via Corollary 35(1). Acquisition: Trivial. Deletion: Assume $H \curvearrowright G \cup \{S\}$. By Corollary 36(1) from the condition of the rule we get $\{S\} \curvearrowright G$. By Corollary 37 we have $G \curvearrowright G$ and then by Corollary 35(1) $G \cup \{S\} \curvearrowright G$. By the assumption and Corollary 37 this means $H \curvearrowright G$. **Q.e.d. (Lemma 47)**

Proof of Lemma 53
Expansion: By Corollary 35(2). Hypothesizing: By Corollary 37 we get $H \curvearrowright H$ and then from the condition of the rule $H \cup \{S\} \curvearrowright H \cup G$ by Corollary 35(1). If we now assume $G \searrow H \cup \{S\}$, we get $G \searrow H \cup G$ by Corollary 34, and then by corollaries 35(3) and 35(2) $G \searrow H$. Acquisition: Trivial. Deletion: Assume $G \searrow H$. By the condition of the rule and Corollary 36(2) we get $\{S\} \searrow H$. Thus by our assumption and Corollary 35(1) we get $G \cup \{S\} \searrow H$. **Q.e.d. (Lemma 53)**

Proof of Lemma 76
It is sufficient to show for each literal λ in Π that $(\lambda\mu)\tau$ is true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$ iff $(\lambda\nu)\pi$ is true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. Additionally, we can restrict λ to be an atom. For a non-generalized literal $\lambda \in \mathcal{LIT}(\mathrm{sig}, \mathrm{V})$ the following are logically equivalent due to Lemma 12 and $(\mu\tau\mathcal{A}_{\iota_\mathcal{A}})|_{\mathcal{V}(\Pi)} = (\nu\pi\mathcal{A}_{\iota_\mathcal{A}})|_{\mathcal{V}(\Pi)}$: $\lambda\mu\tau$ is true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$; λ is true w.r.t. $\mathcal{A}_{\mu\tau\mathcal{A}_{\iota_\mathcal{A}}}$; λ is true w.r.t. $\mathcal{A}_{\nu\pi\mathcal{A}_{\iota_\mathcal{A}}}$; $\lambda\nu\pi$ is true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. For a '<'-atom the argumentation is different because semantic equality is not enough here. However, in case of $\mu\tau|_{\mathcal{V}_<(\Pi)} = \nu\pi|_{\mathcal{V}_<(\Pi)}$, for a literal $(\neg{<}\aleph)$ or $\overline{(\neg{<}\aleph)}$ in Π we have $\neg(\mu\tau) = \neg(\nu\pi)$ and $\aleph(\mu\tau) = \aleph(\nu\pi)$ by Global Requirement 67. By Global Requirement 70 we then get $(\neg\mu)\tau \approx_\mathcal{A} (\neg\nu)\pi$ and $(\aleph\mu)\tau \approx_\mathcal{A} (\aleph\nu)\pi$. This also holds in the case of a semantic induction ordering. Therefore the following are logically equivalent: $((\neg{<}\aleph)\mu)\tau$ is true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$; $(\neg\mu)\tau <_\mathcal{A} (\aleph\mu)\tau$; $(\neg\nu)\pi <_\mathcal{A} (\aleph\nu)\pi$; $((\neg{<}\aleph)\nu)\pi$ is true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. **Q.e.d. (Lemma 76)**

Proof of Lemma 79

For each $x \in \mathrm{Y}' \cap \mathrm{GENDOM}(\mu)$ (due to $(\overline{(\mathrm{Def}\, x\mu)}, \tau, \mathcal{A})$ being a counterexample) we have $\mathcal{A}_{\iota_\mathcal{A}}(x\mu\tau) \in \mathcal{A}_{\mathrm{CONS},s}$ for s being the sort of x. Since $\mathcal{A} \in \mathrm{K}$ we know that $\mathcal{A}$ is CONS:cons-term-generated by Global Requirement 66. Thus there is some $u_x \in \mathcal{GT}(\mathrm{cons})_s$ with $\mathcal{A}_{\iota_\mathcal{A}}(x\mu\tau) = \mathcal{A}(u_x)$. For $x \in \mathrm{GENDOM}(\mu) \setminus \mathrm{Y}'$ let u_x be some arbitrary constructor ground term of the sort of x. Using the Axiom of Choice, we now define $(x \in \mathrm{V})$: $x\pi := \left\{ \begin{array}{ll} u_x & \text{if } x \in \mathrm{GENDOM}(\mu) \\ x\mu\tau & \text{otherwise} \end{array} \right\}$. Now we immediately get the items 1, 2, and 3 of the lemma. Next we show item 4. In case of $\mathrm{GENDOM}(\mu) \cap \mathcal{V}_<(\Pi) = \emptyset$ by item 2 we get $\pi|_{\mathcal{V}_<(\Pi)} = (\mu\tau)|_{\mathcal{V}_<(\Pi)}$. Thus item 4 follows from $\mathcal{V}(\Pi) \subseteq \mathrm{Y}'$, item 3, and Lemma 76.

Now assume that $(\Pi', \neg', \mathrm{Y}, \Theta)$ results from application of μ to $(\Pi, \neg)$ w.r.t. Z. Let ξ be given as in Definition 78.

We define the substitution τ' by a little change of τ on the set of variables $\xi[\mathrm{Y}]$: $\tau'|_{\mathrm{V}\setminus\xi[\mathrm{Y}]} = \tau|_{\mathrm{V}\setminus\xi[\mathrm{Y}]}$; $\tau'|_{\xi[\mathrm{Y}]} = (\xi^{-1}\pi)|_{\xi[\mathrm{Y}]}$. By item 1 we get $(\tau', \mathcal{A}) \in \mathrm{Info}$. Moreover, since, by Definition 78, $\xi[\mathrm{Y}] \cap \mathrm{Z} = \emptyset$, we get $\tau'|_\mathrm{Z} = \tau|_\mathrm{Z}$. For $x \in \mathrm{Y}$ by item 3 (due to $\mathrm{Y} \subseteq \mathrm{Y}'$) we get $x\xi\tau'\mathcal{A}_{\iota_\mathcal{A}} = x\pi\mathcal{A}_{\iota_\mathcal{A}} = x\mu\tau\mathcal{A}_{\iota_\mathcal{A}} = x\mu\tau'\mathcal{A}_{\iota_\mathcal{A}}$ because of $\xi[\mathrm{Y}] \cap \mathcal{V}(\mu[\mathrm{Y}]) = \emptyset$ by Definition 78. Therefore, $(\Theta, \tau', \mathcal{A})$ is a counterexample, i.e. item 5 holds.

Finally we are going to show item 6. Define $(x \in \mathrm{V})$: $x\varrho := \left\{ \begin{array}{ll} x\xi\tau & \text{if } x \in \mathrm{Y} \\ x\pi & \text{otherwise} \end{array} \right\}$. Due to item 1 we get $(\varrho, \mathcal{A}) \in \mathrm{Info}$.

Claim 0: For $x \in \mathcal{V}(\Pi, \neg)$ we have $x\varrho\mathcal{A}_{\iota_\mathcal{A}} = x(\mu|_{\mathrm{V}\setminus\mathrm{Y}} \cup \xi|_\mathrm{Y})\tau\mathcal{A}_{\iota_\mathcal{A}}$. When the induction ordering is not semantic, for $x \in \mathcal{V}_<(\Pi) \cup \mathcal{V}(\neg)$ we have $x\varrho = x(\mu|_{\mathrm{V}\setminus\mathrm{Y}} \cup \xi|_\mathrm{Y})\tau$.
Proof of Claim 0: In case of $x \in \mathrm{Y}$ we have $x\varrho = x\xi\tau = x(\mu|_{\mathrm{V}\setminus\mathrm{Y}} \cup \xi|_\mathrm{Y})\tau$. Otherwise, in case of $x \in \mathcal{V}(\Pi, \neg) \setminus \mathrm{Y}$, by item 3 (due to $\mathcal{V}(\Pi, \neg) \subseteq \mathrm{Y}'$) we have $x\varrho\mathcal{A}_{\iota_\mathcal{A}} = x\pi\mathcal{A}_{\iota_\mathcal{A}} = x\mu\tau\mathcal{A}_{\iota_\mathcal{A}} = x(\mu|_{\mathrm{V}\setminus\mathrm{Y}} \cup \xi|_\mathrm{Y})\tau\mathcal{A}_{\iota_\mathcal{A}}$. Now assume the induction ordering not to be semantic. In case of $x \in (\mathcal{V}_<(\Pi) \cup \mathcal{V}(\neg)) \setminus \mathrm{Y}$, since we are in the second case of the definition of Y in Definition 78, we have $x \notin \mathrm{GENDOM}(\mu)$ and thus get by item 2: $x\varrho = x\pi = x\mu\tau = x(\mu|_{\mathrm{V}\setminus\mathrm{Y}} \cup \xi|_\mathrm{Y})\tau$.
Q.e.d. (Claim 0)

Claim 1: $(\Pi, \varrho, \mathcal{A})$ is a counterexample.
Proof of Claim 1: Let λ be an arbitrary literal from Π. We have to show that $\lambda\varrho$ is false w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$.
We first treat the case that λ is a non-generalized literal. Then (according to the definition of Π') $\lambda\mu$ occurs in Π'. Since $(\Pi', \tau, \mathcal{A})$ is a counterexample, $(\lambda\mu, \tau, \mathcal{A})$ is a counterexample, too. Thus, by Lemma 76, we only have to show $x\varrho\mathcal{A}_{\iota_\mathcal{A}} = x\mu\tau\mathcal{A}_{\iota_\mathcal{A}}$ for all $x \in \mathcal{V}(\lambda)$. In case of $x \in \mathrm{Y}$, by the assumption that $(\Theta, \tau, \mathcal{A})$ is a counterexample, $x\varrho\mathcal{A}_{\iota_\mathcal{A}} = x\xi\tau\mathcal{A}_{\iota_\mathcal{A}} = x\mu\tau\mathcal{A}_{\iota_\mathcal{A}}$. Otherwise, by item 3 (due to $\mathcal{V}(\Pi, \neg) \subseteq \mathrm{Y}'$) we get $x\varrho\mathcal{A}_{\iota_\mathcal{A}} = x\pi\mathcal{A}_{\iota_\mathcal{A}} = x\mu\tau\mathcal{A}_{\iota_\mathcal{A}}$.
Now, to the contrary, we assume that λ is a '<'-literal. Then (according to the definition of Π') $\lambda(\mu|_{\mathrm{V}\setminus\mathrm{Y}} \cup \xi|_\mathrm{Y})$ occurs in Π'. Since $(\Pi', \tau, \mathcal{A})$ is a counterexample, $(\lambda(\mu|_{\mathrm{V}\setminus\mathrm{Y}} \cup \xi|_\mathrm{Y}), \tau, \mathcal{A})$ is a counterexample, too. By Claim 0 we can apply Lemma 76 to infer that $(\lambda, \varrho, \mathcal{A})$ is a counterexample. Q.e.d. (Claim 1)

Claim 2: $\neg\varrho \lesssim_\mathcal{A} \neg'\tau$.
Proof of Claim 2: By definition we have $\neg' = \neg(\mu|_{\mathrm{V}\setminus\mathrm{Y}} \cup \xi|_\mathrm{Y})$. Thus by Global Requirement 70 and Claim 0 we have $\neg'\tau = (\neg(\mu|_{\mathrm{V}\setminus\mathrm{Y}} \cup \xi|_\mathrm{Y}))\tau \approx_\mathcal{A} \neg((\mu|_{\mathrm{V}\setminus\mathrm{Y}} \cup \xi|_\mathrm{Y})\tau) \approx_\mathcal{A} \neg\varrho$. Q.e.d. (Claim 2)

Q.e.d. (Lemma 79)

Proof of Lemma 81

If no step in the reduction $t\tau \xrightarrow{*}_{\mathrm{R,X}} t''$ takes place at a non-variable position of t, then due to the linearity of t there is some $\tau' \in \mathcal{GENSUB}(\mathrm{V}, \mathcal{T}(\mathrm{X} \cup \mathrm{X}'))$ such that $t'' = t\tau'$; $\forall x \in \mathcal{V}(t)$: $x\tau \xrightarrow{*}_{\mathrm{R,X}} x\tau'$; and $\forall x \in \mathrm{V} \setminus \mathcal{V}(t)$: $x\tau = x\tau'$. By $\mathrm{X} \subseteq \mathrm{X}'$ and Lemma 20: $\tau' \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}'))$. Since this was excluded by the assumption of the lemma, we know that there must be some first reduction step at a non-variable position $p \in \mathbb{FPOS}(t)$. More precisely, there must be some $\tau' \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}'))$ as above and some $\mu \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}))$ and some rule $(l{=}r \longleftarrow \lambda_0 \ldots \lambda_{n-1}) \in \mathrm{R}$ such that $(t/p)\tau' = l\mu$ and $(\lambda_0 \ldots \lambda_{n-1})\mu$ is fulfilled w.r.t. $\longrightarrow_{\mathrm{R,X}}$. Define $\mathrm{Z} := \mathcal{V}(l{=}r \longleftarrow \lambda_0 \ldots \lambda_{n-1})$. Let ξ be as in the definition of $\mathrm{SUPERPOSE}(t, \mathrm{Y})$. Define $(x \in \mathrm{V})$ $x\varrho := \left\{ \begin{array}{ll} x\xi^{-1}\mu & \text{if } x \in \xi[\mathrm{Z}] \\ x\tau' & \text{otherwise} \end{array} \right\}$. Due to $l\xi\varrho = l\xi\xi^{-1}\mu = l\mu = (t/p)\tau' = (t/p)\varrho$ (last step due to $\xi[\mathrm{Z}] \cap \mathrm{Y} = \emptyset$ and $\mathcal{V}(t) \subseteq \mathrm{Y}$) we can define $\sigma := \mathrm{mgu}(\{(l\xi, t/p)\}, \mathrm{Y} \cup \xi[\mathrm{Z}])$ and then get some substitution π with $(\sigma\pi)|_{\mathrm{Y} \cup \xi[\mathrm{Z}]} = \varrho|_{\mathrm{Y} \cup \xi[\mathrm{Z}]}$. Since $\mu \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}))$, $\tau' \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}'))$, and $\mathrm{X} \subseteq \mathrm{X}'$; and since $\xi \in \mathcal{SUB}(\mathrm{V}, \mathrm{V})$ is a bijection; we get $\varrho \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}'))$. Thus we can assume $\pi \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}'))$ w.l.o.g., i.e. item 1 is satisfied. Moreover, using the fact that $\xi[\mathrm{Z}] \cap \mathrm{Y} = \emptyset$, we get $(\sigma\pi)|_{\mathrm{Y}} = \varrho|_{\mathrm{Y}} = \tau'|_{\mathrm{Y}}$. Thus, from $\tau'|_{\mathrm{V} \setminus \mathcal{V}(t)} = \tau|_{\mathrm{V} \setminus \mathcal{V}(t)}$ we know that item 2 is satisfied, and from $(\tau'\mathcal{I}_\iota)|_{\mathcal{V}(t)} = (\tau\mathcal{I}_\iota)|_{\mathcal{V}(t)}$ and $\mathcal{V}(t) \subseteq \mathrm{Y}$ we know that item 3 is satisfied. When we now define $\Lambda := (\overline{\lambda_0} \ldots \overline{\lambda_{n-1}})\xi$, then we know that $(\Lambda, \sigma) \in \mathrm{SUPERPOSE}(t, \mathrm{Y})$. Item 4 holds, because, in case of $t \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$, we have $t\tau \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$ and then by Lemma 20 $(t/p)\tau' \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$, so that the matching rule $l{=}r \longleftarrow \lambda_0 \ldots \lambda_{n-1}$ must be a constructor rule and thus the literals λ_i are all positive, cf. Definition 6. For item 5 it now suffices to show that $\lambda_i\xi\sigma\pi$ is true w.r.t. $\mathcal{I}_\iota$ for each $i \prec n$. Due to $\lambda_i\xi\sigma\pi = \lambda_i\xi\varrho = \lambda_i\xi\xi^{-1}\mu = \lambda_i\mu$ it suffices to show $\lambda_i\mu$ to be true w.r.t. $\mathcal{I}_\iota$. Note that we already know that $\lambda_i\mu$ is fulfilled w.r.t. $\longrightarrow_{\mathrm{R,X}}$. Thus, in case of $\lambda_i = (u{=}v)$ for some $u, v \in \mathcal{T}$, we get $u\mu{\downarrow}_{\mathrm{R,X}} v\mu$, i.e. $\mathcal{I}_\iota(u\mu) = \mathcal{I}_\iota(v\mu)$. Otherwise, in case of $\lambda_i = (\mathrm{Def}\, u)$, we get $u\mu \xrightarrow{*}_{\mathrm{R,X}} \hat{u}$ for some $\hat{u} \in \mathcal{GT}(\mathrm{cons})$, i.e. $\mathcal{I}_\iota(u\mu) = \mathcal{I}_\iota(\hat{u}) \in \mathcal{I}_{\mathrm{CONS},s}$ for s being the sort of $\hat{u}$. Finally, in case of $\lambda_i = (u{\neq}v)$, we get $u\mu \xrightarrow{*}_{\mathrm{R,X}} \hat{u}{\not\downarrow}_{\mathrm{R,X}} \hat{v} \xleftarrow{*}_{\mathrm{R,X}} v\mu$ for some $\hat{u}, \hat{v} \in \mathcal{GT}(\mathrm{cons})$. Since by item 4 we may assume $\longrightarrow_{\mathrm{R,X}}$ to be confluent in this case, we get $\hat{u} \not\leftrightarrow_{\mathrm{R,X}} \hat{v}$ and therefore $\mathcal{I}_\iota(u\mu) = \mathcal{I}_\iota(\hat{u}) \neq \mathcal{I}_\iota(\hat{v}) = \mathcal{I}_\iota(v\mu)$. **Q.e.d. (Lemma 81)**

Proof of Lemma 82

Due to $t\tau \xleftrightarrow{*}_{\mathrm{R,X}} t'\tau$ and confluence of $\longrightarrow_{\mathrm{R,X}}$ we get $t\tau \xrightarrow{*}_{\mathrm{R,X}} t'' \xleftarrow{*}_{\mathrm{R,X}} t'\tau$ for some t''. First we consider the case that there is some $\tau' \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}'))$ such that $t\tau' = t'' = t'\tau'$; $\forall x \in \mathcal{V}(t, t')$: $x\tau \xrightarrow{*}_{\mathrm{R,X}} x\tau'$; $\forall x \in \mathrm{V} \setminus \mathcal{V}(t, t')$: $x\tau = x\tau'$. In this case we can define $\Lambda := \emptyset$ and $\sigma := \mathrm{mgu}(\{(t, t')\}, \mathrm{Y})$ and get some π with $(\sigma\pi)|_{\mathrm{Y}} = \tau'|_{\mathrm{Y}}$. W.l.o.g. we may assume $\pi \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{X}'))$. Then we have $(\Lambda, \sigma) \in \mathrm{NARROW}(t, t', \mathrm{Y})$ and all items of the lemma hold due to $\mathcal{V}(t, t') \subseteq \mathrm{Y}$. Otherwise, if such a τ' does not exist, due to t and t' being linear and $\mathcal{V}(t) \cap \mathcal{V}(t') = \emptyset$, the conditions of Lemma 81 are satisfied, matching its t either to our t or to our t'. Thus, in any case, we have $(\Lambda, \sigma) \in \mathrm{NARROW}(t, t', \mathrm{Y})$ and all items of the lemma hold due to $\mathcal{V}(t, t') \subseteq \mathrm{Y}$. **Q.e.d. (Lemma 82)**

Proof of Lemma 83
In the D-case $\longrightarrow_{\mathrm{R,X}}$ is confluent due to Definition 63. Otherwise we have $\mathrm{X}=\emptyset$ and thus $\longrightarrow_{\mathrm{R,X}}$ is confluent due to Global Requirement 62. Now assume $\mathcal{A}_{\iota_{\mathcal{A}}}(t\tau)=\mathcal{A}_{\iota_{\mathcal{A}}}(t'\tau)$. If the D- or E-case holds we get $\mathcal{A}=\mathcal{I}$ and therefore $\mathcal{I}_{\iota_{\mathcal{I}}}(t\tau)=\mathcal{I}_{\iota_{\mathcal{I}}}(t'\tau)$. Otherwise the lemma says that $t,t'\in\mathcal{T}(\mathrm{cons},\mathrm{V_{CONS}})$, $\mathrm{X}=\emptyset$, and K is a sub-class of the class of constructor-minimal models of R; and we define $\mathcal{I}$ as in the E-case. Then we have $t\tau,t'\tau\in\mathcal{GT}(\mathrm{cons})$ and, since R is a Def-MCRS and $\longrightarrow_{\mathrm{R},\emptyset}$ is confluent (cf. Global Requirement 62), by Theorem 25(1), $\mathcal{I}$ is a constructor-minimum model of R. By Definition 63 we have $\mathcal{A}\in\mathrm{K}$. Thus, $\mathcal{A}$ is a constructor-minimal model of R. Since $\mathcal{I}$ is a constructor-minimum model, $\mathcal{A}$ must be a constructor-minimum model of R, too. Thus there must be a cons-homomorphism $c::\mathcal{A}|_{\mathbb{CU}(\{\mathrm{CONS}\}\times\$)}\to\mathcal{I}|_{\mathbb{CU}(\{\mathrm{CONS}\}\times\$)}$. Now Lemma 2 says that $\mathcal{A}|_{\mathbb{CU}(\{\mathrm{CONS}\}\times\$)}\,c=\mathcal{I}|_{\mathbb{CU}(\{\mathrm{CONS}\}\times\$)}$ (where $\mathcal{A}|_{\mathbb{CU}(\{\mathrm{CONS}\}\times\$)}$ is meant to denote the evaluation cons-homomorphism of the cons-algebra $\mathcal{A}|_{\mathbb{CU}(\{\mathrm{CONS}\}\times\$)}$). Thus we get $\mathcal{I}(t\tau)=\mathcal{I}(t'\tau)$ from $c_s(\mathcal{A}(t\tau))=c_s(\mathcal{A}(t'\tau))$ due to $t\tau,t'\tau\in\mathcal{GT}(\mathrm{cons})$. This means that we get $\mathcal{I}_{\iota_{\mathcal{I}}}(t\tau)=\mathcal{I}_{\iota_{\mathcal{I}}}(t'\tau)$ in any case. In other words $t\tau\overset{*}{\longleftrightarrow}_{\mathrm{R,X}}t'\tau$. Finally, if t,t' are a linear terms with $\mathcal{V}(t)\cap\mathcal{V}(t')=\emptyset$, $\mathcal{V}(t,t')\subseteq\mathrm{Y}$, and Y is a finite subset of V, then by Lemma 82 we get $\mathrm{NARROW}(t,t',\mathrm{Y})\neq\emptyset$. **Q.e.d. (Lemma 83)**

Proof of Lemma 84
Since $(u_0{\neq}v_0)\ldots(u_{m-2}{\neq}v_{m-2})(u_{m-1}{\neq}v_{m-1})(t[\,p_j\leftarrow u_j\mid j\prec m\,]=t[\,p_j\leftarrow v_j\mid j\prec m\,])$ is valid (even deductively), we can assume this formula to be in L w.l.o.g.. Then the $=$-Decompose rule is a sub-rule of the Lemma Apply rule which is a safe sub-rule of the Transformation rule according to Lemma 93. **Q.e.d. (Lemma 84)**

Proof of Lemma 85
For $j\prec m$, let x_j be a constructor variable of the sort of v_j that not already occurs in t. Moreover assume the x_j to be mutually distinct. Let $\mu\in\mathcal{GENSUB}(\mathrm{V},\mathcal{T})$ be given in such a way that $\mathrm{GENDOM}(\mu)\subseteq\{x_0,\ldots,x_{m-1}\}$ and $\forall j\prec m\colon\ x_j\mu=v_j$. Now since $(\mathrm{Def}\,t[\,p_j\leftarrow x_j\mid j\prec m\,])$ is valid (even deductively) due to $t[\,p_j\leftarrow x_j\mid j\prec m\,]\in\mathcal{T}(\mathrm{cons},\mathrm{V_{CONS}})$, we can assume $(\mathrm{Def}\,t[\,p_j\leftarrow x_j\mid j\prec m\,])\in L$ w.l.o.g.. Then the Def-Decompose rule is a sub-rule of the Lemma Apply rule which is a safe sub-rule of the Transformation rule according to Lemma 93. **Q.e.d. (Lemma 85)**

Proof of Lemma 86
The $\neq$-Tautology Remove rule is trivially safe. For showing it to be a sub-rule of the Transformation rule it is sufficient to show that $(u_0{\neq}u_1)$ is valid under the assumption that the conditions of the $\neq$-Tautology Remove rule are satisfied. Then there are some terms $v_0,v_1\in\mathcal{T}(\mathrm{cons},\mathrm{V_{CONS}})$ and some $\zeta\in\mathcal{SUB}(\mathrm{V},\mathrm{V})$ such that $(v_0{\neq}v_1)$ is linear and $(v_0{\neq}v_1)\zeta=(u_0{\neq}u_1)$. Furthermore: v_0,v_1 are clashing and $\forall i\prec 2\colon\ \forall p\in\mathbb{FPOS}(v_i)\colon\ \forall((l,r),C)\in\mathrm{R}\colon$ (v_i/p and l are clashing). Therefore we have $\mathrm{NARROW}(v_0,v_1,\mathcal{V}(v_0,v_1))=\emptyset$. Now for each $(\tau,\mathcal{A})\in\mathrm{Info}$ we get $(\zeta\tau,\mathcal{A})\in\mathrm{Info}$ and thus $\mathcal{A}_{\iota_{\mathcal{A}}}(v_0\zeta\tau)\neq\mathcal{A}_{\iota_{\mathcal{A}}}(v_1\zeta\tau)$ by Lemma 83 and the assumed constructor-minimality. This means that $(u_0{\neq}u_1)$ is valid.
Q.e.d. (Lemma 86)

Proof of Lemma 89
If the Applicative Literal Remove rule were not safe, then there would be some $(\tau, \mathcal{A}) \in$ Info such that $(\Gamma\Delta)\tau$ is false w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$, but $(\Gamma(\lambda\mu)\Delta)\tau$ is true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$, and $\Pi\overline{\lambda}\Lambda$ is valid. By the condition that $\Gamma\Delta$ contains $(\Pi\Lambda)\mu$, we conclude that $(\Pi\overline{\lambda}\Lambda)\mu\tau$ is false w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. By the condition that $\forall x \in \mathrm{GENDOM}(\mu)$: $\left(\ \overline{(\mathrm{Def}\, x\mu)} \text{ is contained in } \Gamma\Delta\ \right)$, we conclude $\forall x \in \mathrm{GENDOM}(\mu)$: ($(\overline{(\mathrm{Def}\, x\mu)}, \tau, \mathcal{A})$ is a counterexample). Thus, by the condition that $\left(\begin{array}{ll} & \mathrm{GENDOM}(\mu) \cap \mathcal{V}_<(\Pi\overline{\lambda}\Lambda) = \emptyset \\ \vee & \text{the induction ordering is semantic} \end{array}\right)$, item 1 and 4 of Lemma 79 imply the existence of some π such that $(\Pi\overline{\lambda}\Lambda, \pi, \mathcal{A})$ is a counterexample, which contradicts the validity of $\Pi\overline{\lambda}\Lambda$. **Q.e.d. (Lemma 89)**

Proof of Lemma 90
When the conditions of the =-Literal Remove rule are satisfied, then we get $\mathcal{A}_{\iota_\mathcal{A}}(t\tau) \neq \mathcal{A}_{\iota_\mathcal{A}}(t'\tau)$ for each $(\tau, \mathcal{A}) \in$ Info by Lemma 83. **Q.e.d. (Lemma 90)**

Proof of Lemma 91
Assume that the conditions of the Def-Literal Remove rule are satisfied. Let $(\tau, \mathcal{A}) \in$ Info. It suffices to refute $\mathcal{A}_{\iota_\mathcal{A}}(t\tau) \in \mathcal{A}_{\mathrm{CONS},s}$. If this were the case, then (since $\mathcal{A}$ is CONS:cons-term-generated by Definition 63 and Global Requirement 66) there would be some $u \in \mathcal{GT}(\mathrm{cons})$ with $\mathcal{A}_{\iota_\mathcal{A}}(t\tau) = \mathcal{A}(u)$. When we define $(y \in \mathrm{V})$ $y\tau' := \left\{\begin{array}{ll} u & \text{if } y = x \\ y\tau & \text{otherwise} \end{array}\right\}$ we get $(\tau', \mathcal{A}) \in$ Info and $\mathcal{A}_{\iota_\mathcal{A}}(x\tau') = \mathcal{A}_{\iota_\mathcal{A}}(u) = \mathcal{A}_{\iota_\mathcal{A}}(t\tau) = \mathcal{A}_{\iota_\mathcal{A}}(t\tau')$. Then, by the assumed D- or E-case, Lemma 83 implies that $\mathrm{NARROW}(x, t, \mathcal{V}(x, t)) \neq \emptyset$ which contradicts the conditions of the Def-Literal Remove rule. **Q.e.d. (Lemma 91)**

Proof of Lemma 92

Claim 1: If $(\lambda_0 \ldots \lambda_{n-1}\lambda''\lambda_n \ldots \lambda_{m-1}, \tau, \mathcal{A})$ (or else $(\lambda'_0 \ldots \lambda'_{n-1}\lambda''\lambda'_n \ldots \lambda'_{m-1}, \tau, \mathcal{A})$) is a counterexample, then $(\lambda'_0 \ldots \lambda'_{n-1}\lambda''\lambda'_n \ldots \lambda'_{m-1}, \tau, \mathcal{A})$
(or else $(\lambda_0 \ldots \lambda_{n-1}\lambda''\lambda_n \ldots \lambda_{m-1}, \tau, \mathcal{A})$) is a counterexample, too.
Proof of Claim 1: Since λ'' equals $(l{=}r)$, $((l{=}r), \tau, \mathcal{A})$ is a counterexample, too. We have only to show for each $i \preceq m$ that $(\lambda_i, \tau, \mathcal{A})$ (or else $(\lambda'_i, \tau, \mathcal{A})$) being a counterexample implies that $(\lambda'_i, \tau, \mathcal{A})$ (or else $(\lambda_i, \tau, \mathcal{A})$) is a counterexample, too. In case of $\lambda'_i = \lambda_i$ this is trivial. Otherwise λ_i rewrites to λ'_i with $l{=}r$. In case of $\lambda_i, \lambda'_i \in \mathcal{LIT}(\mathrm{sig}, \mathrm{V})$ this follows from Lemma 12. Otherwise the induction ordering must be semantic and we are finished due to Corollary 75. Q.e.d. (Claim 1)

That the Constant Rewrite rule is safe follows directly from Claim 1. To show it to be a sub-rule of the Transformation rule suppose $((\lambda_0 \ldots \lambda_{n-1}\lambda''\lambda_n \ldots \lambda_{m-1}, \aleph), \tau, \mathcal{A})$ to be a counterexample. By Claim 1, $((\lambda'_0 \ldots \lambda'_{n-1}\lambda''\lambda'_n \ldots \lambda'_{m-1}, \aleph'), \tau, \mathcal{A})$ is a counterexample, too. In case that $\aleph$ rewrites to $\aleph'$ with $l{=}r$ and the induction ordering is semantic, Corollary 75 implies $\aleph'\tau \approx_\mathcal{A} \aleph\tau$ because $((l{=}r), \tau, \mathcal{A})$ is a counterexample. Otherwise, we have $\aleph' = \aleph$. Thus, in any case, $((\lambda'_0 \ldots \lambda'_{n-1}\lambda''\lambda'_n \ldots \lambda'_{m-1}, \aleph'), \tau, \mathcal{A}) \precsim ((\lambda_0 \ldots \lambda_{n-1}\lambda''\lambda_n \ldots \lambda_{m-1}, \aleph), \tau, \mathcal{A})$.

Q.e.d. (Lemma 92)

Proof of Lemma 93

Since Γ is contained in all newly introduced formulas, the Lemma Apply rule is safe. To show it to be sub-rule of the Transformation rule assume $((\Gamma,\aleph),(\tau,\mathcal{A}))$ to be a counterexample. We make a complete case analysis as follows.

There is some $i \prec m$ with $\lambda_i\tau$ being true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$:

Taking the smallest such i, we know that $(\overline{\lambda_i}\lambda_{i-1}\ldots\lambda_0)\tau$ is false w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. Then by the condition that all literals of Λ_i are contained in $\lambda_{i-1}\ldots\lambda_0$, we conclude that $((\overline{\lambda_i}\Lambda_i\Gamma,\aleph),(\tau,\mathcal{A}))$ is a counterexample.

$\forall i \prec m$: ($\lambda_i\tau$ is false w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$):

In this case, $(\lambda_{m-1}\ldots\lambda_0\Gamma,\tau,\mathcal{A})$ is a counterexample. From the condition that all literals $\overline{(\mathrm{Def}\,x\mu)}$ with $x\in\mathrm{GENDOM}(\mu)$ are contained in $\lambda_{m-1}\ldots\lambda_0\Gamma$, we get $\forall x\in\mathrm{GENDOM}(\mu)$: $(\overline{(\mathrm{Def}\,x\mu)},\tau,\mathcal{A})$ is a counterexample. Let π be given according to Lemma 79. Now, since $\lambda_{m-1}\ldots\lambda_0\Gamma$ contains $\Pi\mu$, we know that $\Pi\mu\tau$ is not true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. Since $\begin{pmatrix}\mathrm{GENDOM}(\mu)\cap\mathcal{V}_<(\Pi)=\emptyset\\ \vee\quad\text{the induction ordering is semantic}\end{pmatrix}$, Lemma 79(4) implies that also $\Pi\pi$ is not true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. By Lemma 79(1) $(\Pi,\pi,\mathcal{A})$ is a counterexample. This means that the proof is complete because we have an invalid lemma.

Q.e.d. (Lemma 93)

Proof of Lemma 96

The result of the Lemma Rewrite rule can be obtained by a Lemma Apply (matching m to $m{+}1$ and Γ to $\Gamma\lambda''\Delta$) followed by a Constant Rewrite followed by a Context Remove (cf. section 7.4) that removes the literal $\overline{\lambda_m}$. Since all these rules are sub-rules of the Transformation rule (cf. Lemma 93, Lemma 92, and Corollary 128), by applying Corollary 39 twice we can conclude that the Lemma Rewrite rule is a sub-rule of the Transformation rule, too.

More interesting is to find out, why it is safe. If it were not, this would have to be due to the only newly introduced formula that does not contain $\Gamma\lambda''\Delta$ as a sub-formula. More precisely, there would have to be some counterexample $(\Lambda_m\Gamma\lambda''[\,p\leftarrow r\,]\Delta,\tau,\mathcal{A})$ such that $(\Gamma\lambda''\Delta)\tau$ is true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. This means that $\lambda''\tau$ is true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$ but $\lambda''[\,p\leftarrow r\,]\tau$ is not.

Claim 1: $\lambda_m\tau$ is false w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$.
Proof of Claim 1: Otherwise we have $\mathcal{A}_{\iota_\mathcal{A}}(l\tau)=\mathcal{A}_{\iota_\mathcal{A}}(r\tau)$ and then due to $\lambda''\in\mathcal{LIT}(\mathrm{sig},\mathrm{V})$ we get by Lemma 12 that $\lambda''\tau$ is true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$ iff $\lambda''[\,p\leftarrow r\,]\tau$ is true w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. This contradicts the situation described above. Q.e.d. (Claim 1)

By Claim 1 we now know that $(\lambda_m\Lambda_m\Gamma\Delta)\tau$ is false w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. Since we also have the condition that $\overline{(\mathrm{Def}\,x\mu)}$ is contained in $\Lambda_m\Gamma\Delta$ for all $x\in\mathrm{GENDOM}(\mu)$, we conclude that $(\overline{(\mathrm{Def}\,x\mu)},\tau,\mathcal{A})$ is a counterexample. Let π be given according to Lemma 79. Since the literals of $\Pi\mu$ are contained in $\lambda_m\Lambda_m\Gamma\Delta$, it follows that $\Pi\mu\tau$ is false w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. Since $\begin{pmatrix}\mathrm{GENDOM}(\mu)\cap\mathcal{V}_<(\Pi)=\emptyset\\ \vee\quad\text{the induction ordering is semantic}\end{pmatrix}$, Lemma 79(4) implies that $\Pi\pi$ is false w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$, too. By Lemma 79(1) this means that Π is not valid.

Q.e.d. (Lemma 96)

Proof of Theorem 99

If $\widehat{L} \models \Gamma$, then $\models \overline{\widehat{L}}, \Gamma$. Thus, considering all variables of Γ to be from $V_\forall$, by Theorem 144 there is a proof tree (without Cut rule and with the restricted version of the Instantiation-rule only) for $\overline{\widehat{L}}, \Gamma$ in our sequent calculus of Appendix B such that all leaf nodes are labeled with axioms. The axioms of this sequent calculus can be removed by Constant Rewrite on atoms and subsequent =- and Def-Decompose with the restrictions given in the theorem. The only possible rule applications on $\overline{\widehat{L}}, \Gamma$ are those that put $\overline{\widehat{L}}$ to pieces. α-rules break $\overline{\widehat{L}}$ into $\overline{\forall x_{0,0} \ldots \forall x_{0,m_0-1} C_0}, \ldots, \overline{\forall x_{n-1,0} \ldots \forall x_{n-1,m_{n-1}} C_{n-1}}$ where $C_0, \ldots, C_{n-1}$ lists the elements of L. Subsequent applications of γ-rules remove the universal quantifiers and finally β- and Restricted Instantiation-rule applications split the tree and set new literals free. The only relevant part of the results of this process are the literals λ with $\mathcal{V}(\lambda) \subseteq V_\forall$, since no other formulas influence applicability of inference rules or the property of being an axiom. Regarding these literals the result of a completed sequence of steps of the above kind can be achieved by a sequence of Lemma Applies with the restrictions given in the theorem. Thus, the subsystem is deductively complete.

If each formula from L contains a positive equation literal and our subsystem contains the Lemma Rewrite instead of the Lemma Apply then we may have to do some proof transformation steps starting from a proof tree consisting solely of Lemma Applies of the restricted kind specified in the theorem as inner nodes and Appendix B axioms as leaf nodes:

1. Reorder the tree such that each left son of a Lemma Apply adds an inequality literal.

2. Associate with each leaf a sequence of Constant Rewrite steps on atoms that transform the formula at its label into a formula that can be removed by a =- or a Def-Decomposition with the restrictions given in the theorem.

3. We will now do a transformation according to two proof transformation rules. For being able to show their termination, we assume that they are always applied at the deepest applicable position in the branch from the root to its leftmost leaf, or, if there is no application possible at this branch, recursively to all subtrees (in any order) that result when one cuts off this branch. Note that this is not leftmost innermost, but more like "innermost leftmost".

   The first transformation rule is to transform the tree in such a way that no Lemma Apply introduces an inequality literal to its left son that is not needed for the Constant Rewrite proof associated with its leftmost leaf. Note that also such literals are not needed which have been introduced above or already occur in the original root formula. This transformation can be described by associating with each Lemma Apply a new function symbol which has the sons as arguments. Suppose f to be a Lemma Apply that introduces an inequality literal $(l \neq r)$ to its left son that is not needed for the Constant Rewrite proof of its leftmost leaf. If this left son is a leaf, then we rewrite according to

   $$f(((l \neq r)\Delta), (y_1), \ldots, (y_m)) \qquad \Longrightarrow \qquad (\Delta)$$

   where the y_i denote the other sons of f which are all discarded here. Otherwise, if the left son of f (introducing $(l \neq r)$, $\lambda_1, \ldots, \lambda_m$) is another Lemma Apply g (introducing $\lambda'_0, \ldots, \lambda'_n$). Then we rewrite according to

$$f(\ \ g(\ (\lambda'_0(l{\neq}r)x_0)\ ,\ \ldots\ ,\ (\lambda'_n(l{\neq}r)x_n)\ ),\ (\lambda_1 y_1),\ \vdots\ (\lambda_m y_m)\ ) \quad \Longrightarrow \quad g(\ t_0\ ,\ \ldots\ ,\ t_n\ )$$

where

$$t_i := \left\{ \begin{array}{ll} (\lambda'_i x_i) & \text{if } (l{\neq}r) \text{ is not needed in the Constant Rewrite proofs of the leaves of } (\lambda'_i(l{\neq}r)x_i) \\ f(\ \ ((l{\neq}r)\lambda'_i x_i),\ (\lambda_1\lambda'_i y_1),\ \vdots\ (\lambda_n\lambda'_i y_m)) & \text{otherwise} \end{array} \right\}.$$

The second transformation rule is to transform the tree in such a way that Lemma Applies in each leftmost branch ending in a leaf are in the order of their first usage for rewriting steps with their invented inequality literals in the Constant Rewrite proof associated with its leaf axiom. Suppose f to be a Lemma Apply being father of its left son being the Lemma Apply g which should come before f. Then we rewrite according to the same rule as given above.

Note that the termination of the whole transformation can be shown using a lexicographic path ordering where the function symbol f is strictly bigger than g in the quasi-ordering on function symbols. The whole quasi-ordering can be constructed a priori by saying a function symbol to be bigger than another if in the original tree it occurs above the other: Due to our strategy for applying our transformation rules, if we have finished our bottom-up cleaning of the branch from the root to its leftmost leaf, then, in all subtrees that result when one cuts off this branch, each node's function symbol is still bigger than the function symbols of its sons.

Furthermore, note that we really can do the Lemma Apply g before f because f does not add a negated Def-literal (on the presence of which the applicability of g could depend) to its left son (but a negated equality literal only).

4. To each left son of a Lemma Apply add the list of the negated literals added to its other sons. This list will play the role of the Λ_m in the Lemma Rewrite rule.

5. Rearrange the tree such that Constant Rewrite steps and Context Remove (cf. section 7.4) steps are inserted into each leftmost branch ending in a leaf in such a way that the inequality literals introduced for each left son of a Lemma Apply are at once used for a Constant Rewrite step and then removed by a Context Remove step. Note that here we also may have to include some Constant Rewrite steps with inequality literals of the original root formula Γ and also to include some Lemma Rewrite steps with $m = 0$ for nevertheless rewriting with inequalities removed by the Context Remove steps (either for using the inequality again or for undoing the steps for keeping the Constant Rewrite proofs of the non-leftmost leaves possible). The latter is possible due to step 4.

6. Finally, we replace all Lemma Applies together with their following Constant Rewrite and Context Remove steps with a Lemma Rewrite where $\Pi\mu$ equals $\lambda_m\Lambda_m$.

Q.e.d. (Theorem 99)

Proof of Lemma 101
The SIG-Variable Remove rule is a sub-rule of the Transformation rule since $\Gamma(x{\neq}t)\Delta$ contains $\Gamma\Delta$. To show the SIG-Variable Remove rule to be safe, suppose $(\Gamma\Delta, \tau, \mathcal{A})$ to be a counterexample. Define $\pi \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{V_{SIG}}^{\mathcal{A}}))$ via $\pi|_{\mathrm{V}\setminus\{x\}} \subseteq \tau$ and $x\pi := t\tau$. Since $x \notin \mathcal{V}(t, \Gamma, \Delta)$ we know that $(\Gamma\Delta)\pi = (\Gamma\Delta)\tau$ and $t\pi = t\tau = x\pi$. Thus $(\Gamma(x{\neq}t)\Delta, \pi, \mathcal{A})$ is a counterexample, too. **Q.e.d. (Lemma 101)**

Proof of Lemma 102
To show that the CONS-Variable Remove rule is a sub-rule of the Transformation rule, assume that $((\Gamma(x{\neq}t)\Delta, \aleph), \tau, \mathcal{A})$ is a counterexample. Then for the sort s of x (due to $x \in \mathrm{V_{CONS}}$) we have $\mathcal{A}_{\mathrm{CONS},s} \ni \mathcal{A}_{\iota_{\mathcal{A}}}(x\tau) = \mathcal{A}_{\iota_{\mathcal{A}}}(t\tau)$. Thus $((\Gamma\overline{(\mathrm{Def}\, t)}\Delta, \aleph), \tau, \mathcal{A})$ is a counterexample, too.
To show that the CONS-Variable Remove rule is safe, assume that $(\Gamma\overline{(\mathrm{Def}\, t)}\Delta, \tau, \mathcal{A})$ is a counterexample. Then $\mathcal{A}_{\iota_{\mathcal{A}}}(t\tau) \in \mathcal{A}_{\mathrm{CONS},s}$ for s being the sort of x. By Global Requirement 66, $\mathcal{A}$ is CONS:cons-term-generated. Thus there is some $u \in \mathcal{GT}(\mathrm{cons})_s$ with $\mathcal{A}_{\iota_{\mathcal{A}}}(t\tau) = \mathcal{A}(u)$. Define $\pi \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{V_{SIG}}^{\mathcal{A}}))$ via $\pi|_{\mathrm{V}\setminus\{x\}} \subseteq \tau$ and $x\pi := u$. Since $x \notin \mathcal{V}(t, \Gamma, \Delta)$ we know that $(\Gamma\Delta)\pi = (\Gamma\Delta)\tau$ and $t\pi = t\tau$. Moreover, $\mathcal{A}_{\iota_{\mathcal{A}}}(x\pi) = \mathcal{A}_{\iota_{\mathcal{A}}}(u) = \mathcal{A}_{\iota_{\mathcal{A}}}(t\tau) = \mathcal{A}_{\iota_{\mathcal{A}}}(t\pi)$. Thus, $(\Gamma(x{\neq}t)\Delta, \pi, \mathcal{A})$ is a counterexample, too.
Q.e.d. (Lemma 102)

Proof of Lemma 103
The SIG-Variable Add rule is safe because $(x{\neq}t)\Gamma$ contains Γ. To show that the SIG-Variable Add rule is a sub-rule of the Transformation rule, assume that $((\Gamma, \aleph), (\tau, \mathcal{A}))$ is a counterexample. Define $\pi \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{V_{SIG}}^{\mathcal{A}}))$ via $\pi|_{\mathrm{V}\setminus\{x\}} \subseteq \tau$ and $x\pi := t\tau$. Since $x \notin \mathcal{V}(t, \Gamma, \aleph)$ we know that $\Gamma\pi = \Gamma\tau$, $x\pi = t\tau = t\pi$, and (by Global Requirement 67) $\aleph\pi = \aleph\tau$. Thus we get the counterexample $(((x{\neq}t)\Gamma, \aleph), (\pi, \mathcal{A})) \lesssim ((\Gamma, \aleph), (\tau, \mathcal{A}))$.
Q.e.d. (Lemma 103)

Proof of Lemma 104
The CONS-Variable Add rule is safe because $(x{\neq}t\tau[\, p_i \leftarrow v_i \mid i \prec n\,])\Gamma$ contains Γ. To show that the CONS-Variable Add rule is a sub-rule of the Transformation rule, assume that $((\Gamma, \aleph), (\tau, \mathcal{A}))$ is a counterexample. For all $i \prec n$ the condition of the rule says that Γ contains $\overline{(\mathrm{Def}\, v_i)}$. Thus, $\mathcal{A}_{\iota_{\mathcal{A}}}(v_i\tau) \in \mathcal{A}_{\mathrm{CONS},s_i}$ for s_i being the sort of v_i. By Global Requirement 66, $\mathcal{A}$ is CONS:cons-term-generated. Thus there is some $u_i \in \mathcal{GT}(\mathrm{cons})_{s_i}$ with $\mathcal{A}_{\iota_{\mathcal{A}}}(v_i\tau) = \mathcal{A}(u_i)$. We define $\pi \in \mathcal{SUB}(\mathrm{V}, \mathcal{T}(\mathrm{V_{SIG}}^{\mathcal{A}}))$ via $\pi|_{\mathrm{V}\setminus\{x\}} \subseteq \tau$ and $x\pi := t\tau[\, p_i \leftarrow u_i \mid i \prec n\,]$. Since $x \notin \mathcal{V}(t[\, p_i \leftarrow v_i \mid i \prec n\,], \Gamma, \aleph)$ we know that $\Gamma\pi = \Gamma\tau$, $\mathcal{A}_{\iota_{\mathcal{A}}}(x\pi) = \mathcal{A}_{\iota_{\mathcal{A}}}(t\tau[\, p_i \leftarrow u_i \mid i \prec n\,]) = \mathcal{A}_{\iota_{\mathcal{A}}}(t\tau[\, p_i \leftarrow v_i\tau \mid i \prec n\,]) = \mathcal{A}_{\iota_{\mathcal{A}}}(t[\, p_i \leftarrow v_i \mid i \prec n\,]\tau) = \mathcal{A}_{\iota_{\mathcal{A}}}(t[\, p_i \leftarrow v_i \mid i \prec n\,]\pi)$, and (by Global Requirement 67) $\aleph\pi = \aleph\tau$. Thus we get the counterexample $(((x{\neq}t\tau[\, p_i \leftarrow v_i \mid i \prec n\,])\Gamma, \aleph), (\pi, \mathcal{A})) \lesssim ((\Gamma, \aleph), (\tau, \mathcal{A}))$. **Q.e.d. (Lemma 104)**

Proof of Lemma 107

The Substitution Add rule is safe, since if $(\Gamma\sigma_i, \tau, \mathcal{A})$ is a counterexample then $(\Gamma, \sigma_i\tau, \mathcal{A})$ is a counterexample, too. To show it to be a sub-rule of the Transformation rule assume that $((\Gamma, \aleph), (\tau, \mathcal{A}))$ is a counterexample. Then there are some $i \preceq n$ and some φ with $(\varphi, \mathcal{A}) \in \mathrm{Info}$; $\aleph\tau \gtrsim_{\mathcal{A}} \aleph(\sigma_i\varphi)$; $(\tau\mathcal{A}_{\iota_{\mathcal{A}}})|_{\mathcal{V}(\Gamma)} = (\sigma_i\varphi\mathcal{A}_{\iota_{\mathcal{A}}})|_{\mathcal{V}(\Gamma)}$; and either $\tau|_{\mathcal{V}_<(\Gamma)} = (\sigma_i\varphi)|_{\mathcal{V}_<(\Gamma)}$ or the induction ordering is semantic. By Lemma 76 we infer that $(\Gamma\sigma_i, \varphi, \mathcal{A})$ is a counterexample. By Global Requirement 70 we get $\aleph\tau \gtrsim_{\mathcal{A}} \aleph(\sigma_i\varphi) \approx_{\mathcal{A}} (\aleph\sigma_i)\varphi$. Thus, we get a counterexample $((\Gamma\sigma_i, \aleph\sigma_i), (\varphi, \mathcal{A})) \precsim ((\Gamma, \aleph), (\tau, \mathcal{A}))$. **Q.e.d. (Lemma 107)**

Proof of Lemma 113

Since Γ is contained in all newly introduced formulas, the Hypothesis Apply rule is safe. To show it to be sub-rule of the Transformation rule assume $((\Gamma, \aleph), (\tau, \mathcal{A}))$ to be a counterexample. We make a complete case analysis as follows.

There is some $i \prec n$ with $\lambda_i\tau$ being true w.r.t. $\mathcal{A}_{\iota_{\mathcal{A}}}$:

Taking the smallest such i, we know that $(\overline{\lambda_i}\lambda_{i-1}\ldots\lambda_0)\tau$ is false w.r.t. $\mathcal{A}_{\iota_{\mathcal{A}}}$. Then by the condition that all literals of Λ_i are contained in $\lambda_{i-1}\ldots\lambda_0$, we conclude that $((\overline{\lambda_i}\Lambda_i\Gamma, \aleph), (\tau, \mathcal{A}))$ is a counterexample.

$\forall i \prec n$: ($\lambda_i\tau$ is false w.r.t. $\mathcal{A}_{\iota_{\mathcal{A}}}$):

In this case, $(\lambda_{n-1}\ldots\lambda_0\Gamma, \tau, \mathcal{A})$ is a counterexample. From the condition that all literals $\overline{(\mathrm{Def}\, x\mu)}$ with $x \in \mathrm{Y}$ are contained in $\lambda_{n-1}\ldots\lambda_0\Gamma$, we get $\forall x \in \mathrm{Y}$: $(\overline{(\mathrm{Def}\, x\mu)}, \tau, \mathcal{A})$ is a counterexample. Define $\mathrm{Z} := \mathcal{V}(\lambda_{n-1}\ldots\lambda_0\Gamma, \aleph)$. The condition of the rule says that $(\Pi', \daleth', \mathrm{Y}, \Theta)$ results from application of μ to $(\Pi, \daleth)$ w.r.t. Z. By Lemma 79(5) (matching its Y′ to our Y) there is some τ' such that $(\tau', \mathcal{A}) \in \mathrm{Info}$, $\tau'|_{\mathrm{Z}} = \tau|_{\mathrm{Z}}$, and $(\Theta, \tau', \mathcal{A})$ is a counterexample. Thus, we have the counterexample $((\lambda_{n-1}\ldots\lambda_0\Theta\Gamma, \aleph), (\tau', \mathcal{A})) \precsim ((\Gamma, \aleph), (\tau, \mathcal{A}))$.

There is some i with $n \preceq i \prec m$ and $\lambda_i\tau'$ being true w.r.t. $\mathcal{A}_{\iota_{\mathcal{A}}}$:

Taking the smallest such i, we know that $(\overline{\lambda_i}\lambda_{i-1}\ldots\lambda_0\Theta\Gamma, \tau', \mathcal{A})$ is a counterexample. Then by the condition that all literals of Λ_i are contained in $\lambda_{i-1}\ldots\lambda_0\Theta$, we conclude that $((\overline{\lambda_i}\Lambda_i\Gamma, \aleph), (\tau', \mathcal{A})) \precsim ((\Gamma, \aleph), (\tau, \mathcal{A}))$ is a counterexample.

$\forall i \prec m$: ($\lambda_i\tau'$ is false w.r.t. $\mathcal{A}_{\iota_{\mathcal{A}}}$):

In this case, $(\lambda_{m-1}\ldots\lambda_0\Theta\Gamma, \tau', \mathcal{A})$ is a counterexample. If we now do not have $\daleth'\tau' <_{\mathcal{A}} \aleph\tau'$, then we conclude with the counterexample $((\daleth'{<}\aleph)\Lambda_m\Gamma, \aleph), (\tau', \mathcal{A})) \precsim ((\Gamma, \aleph), (\tau, \mathcal{A}))$ because all literals of Λ_m are contained in $\lambda_{m-1}\ldots\lambda_0\Theta$. Otherwise, since the condition of the rule says that $\lambda_{m-1}\ldots\lambda_0\Theta\Gamma$ contains Π', we know that we have a counterexample $((\Pi', \daleth'), (\tau', \mathcal{A})) < ((\Gamma, \aleph), (\tau', \mathcal{A})) \precsim ((\Gamma, \aleph), (\tau, \mathcal{A}))$. Thus, it is now sufficient to find a counterexample $((\Pi, \daleth), (\varrho, \mathcal{A})) \precsim ((\Pi', \daleth'), (\tau', \mathcal{A}))$ because then we have a counterexample for a hypothesis which is strictly smaller than our original counterexample for the goal. That counterexample is given by an application of Lemma 79(6) (matching its τ to our τ' and its Y′ to V) because from the condition that all literals $\overline{(\mathrm{Def}\, x\mu)}$ with $x \in \mathrm{GENDOM}(\mu)$ are contained in $\lambda_{m-1}\ldots\lambda_0\Gamma$, we get $\forall x \in \mathrm{GENDOM}(\mu)$: $(\overline{(\mathrm{Def}\, x\mu)}, \tau', \mathcal{A})$ is a counterexample.

Q.e.d. (Lemma 113)

Proof of Lemma 117

The proof is like that of Lemma 96 but uses Lemma 113 instead of Lemma 93, Hypothesis Apply instead of Lemma Apply, and Hypothesis Rewrite instead of Lemma Rewrite. Moreover, the last paragraph of the proof of Lemma 96 must be replaced with:

By Claim 1 and the condition that Λ_m contains all literals of $\lambda_{m-1}\lambda_{m-2}\ldots\lambda_0\Theta$, we now know that $(\lambda_m\ldots\lambda_0\Theta\Gamma\Delta)\tau$ is false w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. Since we also have the condition that $\overline{(\mathrm{Def}\,x\mu)}$ is contained in $\lambda_{m-1}\ldots\lambda_0\Gamma\Delta$ for all $x \in \mathrm{GENDOM}(\mu)$, we conclude that $(\overline{(\mathrm{Def}\,x\mu)},\tau,\mathcal{A})$ is a counterexample. $(\Pi',\tau,\mathcal{A})$ and $(\Theta,\tau,\mathcal{A})$ are counterexamples because Π' is contained in $\lambda_m\Lambda_m\Gamma\Delta$ by the condition of the Hypothesis Rewrite rule. Thus, by Lemma 79(6) there is some counterexample $(\Pi,\varrho,\mathcal{A})$. This means that Π is an invalid hypothesis.

Q.e.d. (Lemma 117)

Proof of Lemma 118

Define $\mathrm{Y} := \mathcal{V}(\Gamma(t{\neq}t')\Delta, \aleph)$. For showing the $\neq$-Solve rule to be a sub-rule of the Transformation rule assume that $((\Gamma(t{\neq}t')\Delta, \aleph), (\tau, \mathcal{A}))$ is a counterexample. Then $\mathcal{A}_{\iota_\mathcal{A}}(t\tau) = \mathcal{A}_{\iota_\mathcal{A}}(t'\tau)$.

If the *D*- or *E*-case holds, let X and $\mathcal{I}$ be defined as usual in these cases, cf. Definition 63, and $\mathrm{X}' := \mathrm{X}$. Otherwise the lemma says that $t, t' \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\mathrm{CONS}})$, and we define X and $\mathcal{I}$ as in the *E*-case and $\mathrm{X}' := \mathrm{V}_{\mathrm{SIG}}{}^{\mathcal{A}}$.

Now Lemma 83 says that $\longrightarrow_{\mathrm{R,X}}$ is confluent and $t\tau \overset{*}{\longleftrightarrow}_{\mathrm{R,X}} t'\tau$. Thus by Lemma 82 we have some π and some $(\Lambda, \sigma) \in \mathrm{NARROW}(t, t', \mathrm{Y})$ such that the items of Lemma 82 hold.

Item 5 of Lemma 82 says that $\Lambda\sigma\pi$ is false w.r.t. $\mathcal{I}_\iota$. In case of the *D*- or *E*-case we then immediately know that $\Lambda\sigma\pi$ is false w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$. Otherwise, since R is a Def-MCRS and $\longrightarrow_{\mathrm{R},\emptyset}$ is confluent (cf. Global Requirement 62) and K is a sub-class of the class of constructor-minimal models, by Corollary 26 $\mathcal{I}$ is initial in K. Thus, we get some $h{::}\mathcal{I}{\to}\mathcal{A}$ and then by Lemma 2 $\mathcal{I}\,h = \mathcal{A}$. By item 4 of Lemma 82, Λ contains only negative literals, which also must be from $\mathcal{LIT}(\mathrm{sig}, \mathrm{V})$ because of $(\Lambda, \sigma) \in \mathrm{NARROW}(t, t', \mathrm{Y})$. In case of a literal $(u{\neq}v)$ in Λ (where s' is the sort of u) we get $\mathcal{A}(u\sigma\pi) = h_{s'}(\mathcal{I}(u\sigma\pi)) = h_{s'}(\mathcal{I}(v\sigma\pi)) = \mathcal{A}(v\sigma\pi)$. In case of a literal $\overline{(\mathrm{Def}\, u)}$ in Λ we get $\mathcal{A}(u\sigma\pi) = h_{s'}(\mathcal{I}(u\sigma\pi)) \in h_{s'}[\mathcal{I}_{\mathrm{CONS},s'}] \subseteq \mathcal{A}_{\mathrm{CONS},s'}$. Thus, in any case $\Lambda\sigma\pi$ is false w.r.t. $\mathcal{A}_{\iota_\mathcal{A}}$.

By item 2 of Lemma 82 and the condition of the $\neq$-Solve rule we get $(\sigma\pi)|_{\mathcal{V}_<(\Gamma(t\neq t')\Delta)\cup\mathcal{V}(\aleph)} = \tau|_{\mathcal{V}_<(\Gamma(t\neq t')\Delta)\cup\mathcal{V}(\aleph)}$ or the induction ordering is semantic. From item 3 of Lemma 82 we get $(\sigma\pi\mathcal{A}_{\iota_\mathcal{A}})|_{\mathcal{V}(t,t')} = (\tau\mathcal{A}_{\iota_\mathcal{A}})|_{\mathcal{V}(t,t')}$; in the *D*- or *E*-case directly, otherwise after application of the above h. Together with item 2 of Lemma 82 we have $(\sigma\pi\mathcal{A}_{\iota_\mathcal{A}})|_{\mathcal{V}(\Gamma(t\neq t')\Delta)\cup\mathcal{V}(\aleph)} = (\tau\mathcal{A}_{\iota_\mathcal{A}})|_{\mathcal{V}(\Gamma(t\neq t')\Delta)\cup\mathcal{V}(\aleph)}$. Thus, since $(\Gamma(t{\neq}t')\Delta, \tau, \mathcal{A})$ is a counterexample, by Lemma 76 $((\Gamma(t{\neq}t')\Delta)\sigma, \pi, \mathcal{A})$ is a counterexample, too. Moreover, by the global requirements 67 and 70 and Definition 73 we get $(\aleph\sigma)\pi \approx_\mathcal{A} \aleph\tau$.

All in all we get the new counterexample $((\Lambda\Gamma(t{\neq}t')\Delta, \aleph)\sigma, (\pi, \mathcal{A})) \lesssim ((\Gamma(t{\neq}t')\Delta, \aleph), (\tau, \mathcal{A}))$, which completes the proof that the $\neq$-Solve rule is a sub-rule of the Transformation rule.

Finally, the $\neq$-Solve rule is safe because if a new goal has a counterexample $((\Lambda\Gamma(t{\neq}t')\Delta)\sigma, \tau, \mathcal{A})$ then $(\Gamma(t{\neq}t')\Delta, \sigma\tau, \mathcal{A})$ is a counterexample, too.

Q.e.d. (Lemma 118)

Proof of Lemma 119

The result of an application of a $\overline{\mathrm{Def}}$-Solve rule can be achieved in the following way: First a CONS-Variable Add to introduce the literal $(x{\neq}t)$, then a $\neq$-Solve of this literal, and finally a CONS-Variable Remove or (in case of σ being the unifier of x and t) an $\neq$-Literal Remove on $(x{\neq}t)\sigma$. Since (under the assumptions of the lemma) all these inference steps are safe applications of the Transformation rule (cf. 104, 118, 102, 87, 88) by Corollary 39 this also holds for the $\overline{\mathrm{Def}}$-Solve rule. **Q.e.d. (Lemma 119)**

Proof of Lemma 141

By induction on A, using Lemma 3, similar to the proof of Lemma 12. The only non-trivial step is that for the quantifiers: We consider the induction steps for $\forall x B$ and $\exists x B$ by denoting one them with $Qx B$. Assume $(\varsigma, s) \in \{\mathrm{SIG}, \mathrm{CONS}\}\times\mathbb{S}$, $x \in \mathrm{V}_{\mathrm{bound},\varsigma,s}\backslash\mathcal{V}(A)$, $y \in \mathrm{V}_{\mathrm{free},\varsigma,s}$, and $B = A\{y\mapsto x\}$. We may w.l.o.g. assume $y \notin \mathcal{V}(\sigma[\mathcal{V}(B)])$. Define $\sigma' \in \mathcal{GENSUB}(\mathrm{V}_{\mathrm{free}}, \mathcal{T}(\mathrm{V}_{\mathrm{free}}))$ by $y\sigma' = y$ and $\sigma'|_{\mathrm{V}_{\mathrm{free}}\backslash\{y\}} = \sigma|_{\mathrm{V}_{\mathrm{free}}\backslash\{y\}}$. Note that $(Qx B)\sigma = Qx(B\sigma)$ and $B\sigma = B\sigma' = A\{y\mapsto x\}\sigma' = A\sigma'\{y\mapsto x\}$. Let $a \in \mathcal{A}_{\varsigma,s}$ be (meta-) Q-quantified. Define $\kappa' \in \mathcal{SUB}(\mathcal{V}_{\mathrm{free}}(B\sigma)\uplus\{y\}, \mathcal{A})$ by $y\kappa' = a$, $\kappa'|_{\mathcal{V}_{\mathrm{free}}(B\sigma)} \subseteq \kappa$; and $\kappa'' \in \mathcal{SUB}(\mathcal{V}_{\mathrm{free}}(B)\uplus\{y\}, \mathcal{A})$ by $y\kappa'' = a$, $\kappa''|_{\mathcal{V}_{\mathrm{free}}(B)} \subseteq (\sigma\mathcal{A}_\kappa)$. Note that $\kappa'' = (\sigma'\mathcal{A}_{\kappa'})|_{\mathcal{V}_{\mathrm{free}}(B)\uplus\{y\}}$. Now the following are logically equivalent by our induction hypothesis: $Qx(B\sigma)$ is true w.r.t. $\mathcal{A}_\kappa$; $A\sigma'$ is true w.r.t. $\mathcal{A}_{\kappa'}$; A is true w.r.t. $\mathcal{A}_{\sigma'\mathcal{A}_{\kappa'}}$; A is true w.r.t. $\mathcal{A}_{\kappa''}$; $Qx B$ is true w.r.t. $\mathcal{A}_{\sigma\mathcal{A}_\kappa}$. **Q.e.d. (Lemma 141)**

Proof of Lemma 143

Assume DHS to be a dual Hintikka set. We then define $R_{\mathrm{DHS}} := \{\ (l,r) \mid \overline{(l{=}r)} \in \mathrm{DHS} \wedge \mathcal{V}(l,r)\subseteq\mathrm{V}_\forall\ \}$. Let $\overset{*}{\longleftrightarrow}_{R_{\mathrm{DHS}}}$ be the congruence closure of R_{DHS} on $\mathcal{T}(\mathrm{sig},\mathrm{V}_\forall)$ and on atoms with top level terms from $\mathcal{T}(\mathrm{sig},\mathrm{V}_\forall)$. Furthermore we define the sig/cons-structure $\mathcal{B}$ in the following way: $\mathcal{B}_{\mathrm{SIG},s} := \mathcal{T}(\mathrm{sig},\mathrm{V}_\forall)_s$;

$$\mathcal{B}_{\mathrm{CONS},s} := \left\{ u[\,p_i \leftarrow v_i \mid i \prec n\,] \in \mathcal{T}(\mathrm{sig},\mathrm{V}_\forall)_s \;\middle|\; \begin{pmatrix} n \in \mathbb{N} \\ \wedge\ p_0,\ldots,p_{n-1} \in \mathcal{POS}(u) \\ \wedge\ v_0,\ldots,v_{n-1} \in \mathcal{T}(\mathrm{sig},\mathrm{V}_\forall) \\ \wedge\ u \in \mathcal{T}(\mathrm{cons},\mathrm{V}_{\forall,\mathrm{CONS}}) \\ \wedge\ \forall i \prec n\colon\ \overline{(\mathrm{Def}\, v_i)} \in \mathrm{DHS} \end{pmatrix} \right\};$$

$$P^{\mathcal{B}} := \{\ (t_0,\ldots,t_{m-1}) \mid \overline{(Pt_0\ldots t_{m-1})} \in \mathrm{DHS} \wedge \forall i \prec m\colon\ t_i \in \mathcal{T}(\mathrm{sig},\mathrm{V}_\forall)\ \};$$

$f^{\mathcal{B}}(t_0,\ldots,t_{m-1}) := ft_0\ldots t_{m-1}$. Note that the definition of $\mathcal{B}_{\mathrm{CONS},s}$ must be so complicated in order to make the constructor sub-universes of $\mathcal{B}$ closed under constructor function applications. Moreover, we define $\mathcal{A} := \mathcal{B}/\overset{*}{\longleftrightarrow}_{R_{\mathrm{DHS}}}$. Finally we define $\iota \in \mathcal{SUB}(\mathrm{V}_\forall, \mathcal{A})$ by $(x \in \mathrm{V}_\forall)\ \ x \mapsto \overset{*}{\longleftrightarrow}_{R_{\mathrm{DHS}}}[\{x\}]$.

Claim 1: For all $A \in \mathcal{AT}$ with $\overline{A} \in \mathrm{DHS}$ we have

$$\forall \varepsilon \in \mathcal{SUB}(\mathrm{V}_\exists, \mathcal{A})\colon\ (A \text{ is true w.r.t. } \mathcal{A}_{\iota\cup\varepsilon}).$$

Proof of Claim 1: For each $\varepsilon \in \mathcal{SUB}(\mathcal{V}_\exists(A), \mathcal{A})$ there is some $\epsilon \in \mathcal{SUB}(\mathcal{V}_\exists(A), \mathcal{B})$ with $\varepsilon = \epsilon\mathcal{A}_\iota$ since $\mathcal{V}_\exists(A)$ is finite. Now assume $A \in \mathcal{AT}$ with $\overline{A} \in \mathrm{DHS}$. By the Instantiation-Closure property we know that $\overline{A}(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon) \in \mathrm{DHS}$. Directly from the definition of $\mathcal{A}_\iota$ we conclude that $A(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon)$ is true w.r.t. $\mathcal{A}_\iota$. By Lemma 141, A is true w.r.t. $\mathcal{A}_{(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon)\mathcal{A}_\iota}$, which finishes the proof due to $(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon)\mathcal{A}_\iota = \iota\cup\varepsilon$. Q.e.d. (Claim 1)

Claim 2: For all $A \in \mathcal{AT}$ with $A \in \mathrm{DHS}$ we have

$$\forall \varepsilon \in \mathcal{SUB}(\mathrm{V}_\exists, \mathcal{A}):\ (A \text{ is false w.r.t. } \mathcal{A}_{\iota\cup\varepsilon}).$$

Proof of Claim 2: For each $\varepsilon \in \mathcal{SUB}(\mathcal{V}_\exists(A), \mathcal{A})$ there is some $\epsilon \in \mathcal{SUB}(\mathcal{V}_\exists(A), \mathcal{B})$ with $\varepsilon = \epsilon\mathcal{A}_\iota$ since $\mathcal{V}_\exists(A)$ is finite. Now assume $A \in \mathcal{AT}$ with $A \in \mathrm{DHS}$. By the Instantiation-Closure property we know that $A(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon) \in \mathrm{DHS}$. Now suppose that A would be true w.r.t. $\mathcal{A}_{\iota\cup\varepsilon}$. Due to $\iota\cup\varepsilon = (\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon)\mathcal{A}_\iota$, by Lemma 141 we know that $A(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon)$ is true w.r.t. $\mathcal{A}_\iota$. First assume that the predicate symbol of A is different from '=' and 'Def'. Now, when $A(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon)$ is true w.r.t. $\mathcal{A}_\iota$, then there is some $B \in \mathcal{AT}$ with $A(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon) \overset{*}{\longleftrightarrow}_{R_{\mathrm{DHS}}} B$, $\overline{B} \in \mathrm{DHS}$, and $\mathcal{V}_\exists(B) = \emptyset$. By $A(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon) \in \mathrm{DHS}$ and the Leibniz-Substitutability we get $B \in \mathrm{DHS}$. This, however, contradicts the Predicate-Consistency. Second, assume that A is is of the form $(\mathrm{Def}\, t)$. Now, when $A(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon)$ is true w.r.t. $\mathcal{A}_\iota$, then there are some $n \in \mathbb{N}$; $u \in \mathcal{T}(\mathrm{cons}, \mathrm{V}_{\forall,\,\mathrm{CONS}})$; $p_0, \ldots, p_{n-1} \in \mathcal{POS}(u)$; $v_0, \ldots, v_{n-1} \in \mathcal{T}(\mathrm{sig}, \mathrm{V}_\forall)$; with $\forall i \prec n$: $\overline{(\mathrm{Def}\, v_i)} \in \mathrm{DHS}$ and $t(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon) \overset{*}{\longleftrightarrow}_{R_{\mathrm{DHS}}} u[\, p_i \leftarrow v_i \mid i \prec n\,]$. By $(\mathrm{Def}\, t)(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon) \in \mathrm{DHS}$ and the Leibniz-Substitutability we get $(\mathrm{Def}\, u[\, p_i \leftarrow v_i \mid i \prec n\,]) \in \mathrm{DHS}$. This, however, contradicts the Def-Consistency. Finally assume that $A = (s{=}t)$. Now, when $A(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon)$ is true w.r.t. $\mathcal{A}_\iota$, then $s(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon) \overset{*}{\longleftrightarrow}_{R_{\mathrm{DHS}}} t(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon)$. By the Leibniz-Substitutability from $(s{=}t)(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon) \in \mathrm{DHS}$ we get $(t(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon){=}t(\mathrm{id}|_{\mathrm{V}_\forall}\cup\epsilon)) \in \mathrm{DHS}$, which contradicts the Reflexive-Consistency. Q.e.d. (Claim 2)

Now that we have shown that for each literal $A \in \mathrm{DHS}$ we have $\forall \varepsilon \in \mathcal{SUB}(\mathrm{V}_\exists, \mathcal{A})$: ($A$ is false w.r.t. $\mathcal{A}_{\iota\cup\varepsilon}$), we want to show this for an arbitrary formula A. If a formula is not a literal, then it is of one of the forms of α/β, α, β, δ, or γ. The induction step for the cases α/β and α is trivial.

Consider a β-formula, say $(A{\wedge}B) \in \mathrm{DHS}$ (other cases analogously), assume $\varepsilon \in \mathcal{SUB}(\mathrm{V}_\exists, \mathcal{A})$, and define $\mathrm{Y} := \mathcal{V}_\exists(A){\cap}\mathcal{V}_\exists(B)$. In case of $\mathrm{Y} = \emptyset$, by the β-Closure property we know that $A \in \mathrm{DHS}$ or $B \in \mathrm{DHS}$, thus, by induction hypothesis, A or B is false w.r.t. $\mathcal{A}_{\iota\cup\varepsilon}$, thus $(A{\wedge}B)$ is false w.r.t. $\mathcal{A}_{\iota\cup\varepsilon}$. Otherwise, in case of $\mathrm{Y} \neq \emptyset$, the Instantiation-Closure property says that $(A{\wedge}B)(\mathrm{id}|_{\mathrm{V}_{\mathrm{free}}\setminus\mathrm{Y}}\cup\epsilon) \in \mathrm{DHS}$ for any $\epsilon \in \mathcal{SUB}(\mathrm{Y}, \mathcal{B})$ with $\epsilon\mathcal{A}_\iota \subseteq \varepsilon$ (which exists since Y is finite). Since $\mathcal{V}_\exists(A(\mathrm{id}|_{\mathrm{V}_{\mathrm{free}}\setminus\mathrm{Y}}\cup\epsilon)) \cap \mathcal{V}_\exists(B(\mathrm{id}|_{\mathrm{V}_{\mathrm{free}}\setminus\mathrm{Y}}\cup\epsilon)) = \emptyset$, according to the first case shown above we know that $(A{\wedge}B)(\mathrm{id}|_{\mathrm{V}_{\mathrm{free}}\setminus\mathrm{Y}}\cup\epsilon)$ is false w.r.t. $\mathcal{A}_{\iota\cup\varepsilon}$. By Lemma 141 we know that $(A{\wedge}B)$ is false w.r.t. $\mathcal{A}_{(\mathrm{id}|_{\mathrm{V}_{\mathrm{free}}\setminus\mathrm{Y}}\cup\epsilon)\mathcal{A}_{\iota\cup\varepsilon}}$, which finishes the proof of the β-case due to $(\mathrm{id}|_{\mathrm{V}_{\mathrm{free}}\setminus\mathrm{Y}}\cup\epsilon)\mathcal{A}_{\iota\cup\varepsilon} = \iota\cup\varepsilon$.

For γ we proceed as follows: Assume $(\varsigma, s) \in \{\mathrm{SIG}, \mathrm{CONS}\}{\times}\mathbb{S}$, $x \in \mathrm{V}_{\mathrm{bound},\,\varsigma,\,s}\setminus\mathcal{V}(A)$, $y \in \mathrm{V}_{\mathrm{free},\,\varsigma,\,s}$, $B = A\{y{\mapsto}x\}$, $\exists x B$ (or else $\overline{\forall x B}$) $\in \mathrm{DHS}$. Let $\mathrm{Y} := \mathcal{V}_{\mathrm{free}}(B)$. For $\iota' \in \mathcal{SUB}(\mathrm{Y}{\uplus}\{y\}, \mathcal{A})$ with $\iota'|_{\mathrm{Y}} \subseteq \iota\cup\varepsilon$ we have to show that A (or else $\overline{A}$) is false w.r.t. $\mathcal{A}_{\iota'}$. By the γ-Closure property we have $B\{x{\mapsto}z\} \in \mathrm{DHS}$ (or else $\overline{B\{x{\mapsto}z\}} \in \mathrm{DHS}$) for some $z \in \mathrm{V}_{\exists,\,\varsigma,\,s}\setminus\mathcal{V}(B)$. Let $\sigma \in \mathcal{SUB}(\mathrm{V}_{\mathrm{free}}, \mathrm{V}_{\mathrm{free}})$ be given by $y\sigma = z$ and $\sigma|_{\mathrm{V}_{\mathrm{free}}\setminus\{y\}} \subseteq \mathrm{id}$. Due to $x \notin \mathcal{V}(A)$ we have $B\{x{\mapsto}z\} = A\{y{\mapsto}x\}\{x{\mapsto}z\} = A\{y{\mapsto}z\} = A\sigma$. By induction hypothesis, $A\sigma$ (or else $\overline{A\sigma}$) is false w.r.t. $\mathcal{A}_{\iota\cup\varepsilon'}$ for the $\varepsilon' \in \mathcal{SUB}(\mathrm{V}_\exists, \mathcal{A})$ given by $z\varepsilon' = y\iota'$ and $\varepsilon'|_{\mathrm{V}_\exists\setminus\{z\}} \subseteq \varepsilon$. By Lemma 141 we know that A (or else $\overline{A}$) is false w.r.t. $\mathcal{A}_{\sigma\mathcal{A}_{\iota\cup\varepsilon'}}$, which finishes the γ-case due to $\iota' \subseteq \sigma\mathcal{A}_{\iota\cup\varepsilon'}$. The latter is really the case: Since $z \notin \mathcal{V}(B)$ we have $z \notin \mathrm{Y}$, since $B = A\{y{\mapsto}x\}$ we have $y \notin \mathrm{Y}$; and thus we get $\iota'|_{\mathrm{Y}} = (\iota\cup\varepsilon)|_{\mathrm{Y}} = (\iota\cup\varepsilon')|_{\mathrm{Y}} \subseteq \sigma(\iota\cup\varepsilon') = \sigma\mathcal{A}_{\iota\cup\varepsilon'}$. Moreover, $y\iota' = z\varepsilon' = z\mathcal{A}_{\iota\cup\varepsilon'} = y\sigma\mathcal{A}_{\iota\cup\varepsilon'}$.

Finally, we prove the δ-case: Assume $(\varsigma, s) \in \{\mathrm{SIG}, \mathrm{CONS}\}{\times}\mathbb{S}$, $x \in \mathrm{V}_{\mathrm{bound},\,\varsigma,\,s}\setminus\mathcal{V}(A)$, $y \in \mathrm{V}_{\mathrm{free},\,\varsigma,\,s}$, $B = A\{y{\mapsto}x\}$, $\overline{\exists x B}$ (or else $\forall x B$) $\in \mathrm{DHS}$.

First, we treat the case of $\mathcal{V}_\exists(B)=\emptyset$. Then, by the δ-Closure property, there is some $t \in \mathcal{T}(\mathrm{V_{free}})_{\varsigma,s}$ such that $\overline{B\{x\mapsto t\}} \in \mathrm{DHS}$ (or else $B\{x\mapsto t\} \in \mathrm{DHS}$). Thus, for $\mathrm{Y} := \mathcal{V}_{\mathrm{free}}(B)$, we can define $\iota' \in \mathcal{SUB}(\mathrm{Y}\uplus\{y\},\mathcal{A})$ by $y\iota' = t\mathcal{A}_{\iota\cup\varepsilon} \in \mathcal{A}_{\varsigma,s}$ and $\iota'|_{\mathrm{Y}} \subseteq \iota\cup\varepsilon$. Let $\sigma \in \mathcal{SUB}(\mathrm{V_{free}},\mathcal{T}(\mathrm{V_{free}}))$ be given by $y\sigma = t$ and $\sigma|_{\mathrm{V_{free}}\setminus\{y\}} \subseteq \mathrm{id}$. Due to $x \notin \mathcal{V}(A)$ we have $B\{x\mapsto t\} = A\{y\mapsto x\}\{x\mapsto t\} = A\{y\mapsto t\} = A\sigma$. By induction hypothesis, $\overline{A\sigma}$ (or else $A\sigma$) is false w.r.t. $\mathcal{A}_{\iota\cup\varepsilon}$. By Lemma 141, $\overline{A}$ (or else A) is false w.r.t. $\mathcal{A}_{\sigma\mathcal{A}_{\iota\cup\varepsilon}}$, which finishes the proof due to $\iota' \subseteq \sigma\mathcal{A}_{\iota\cup\varepsilon}$.

Second, in case of $\mathrm{Y} \neq \emptyset$ for $\mathrm{Y} := \mathcal{V}_\exists(B)$, the Instantiation-Closure property says that $\overline{\exists x B}(\mathrm{id}|_{\mathrm{V_{free}}\setminus\mathrm{Y}}\cup\epsilon) \in \mathrm{DHS}$ (or else $\forall x B(\mathrm{id}|_{\mathrm{V_{free}}\setminus\mathrm{Y}}\cup\epsilon) \in \mathrm{DHS}$) for any $\epsilon \in \mathcal{SUB}(\mathrm{Y},\mathcal{B})$ with $\epsilon\mathcal{A}_\iota \subseteq \varepsilon$ (which exists since Y is finite). Since $\mathcal{V}_\exists(B(\mathrm{id}|_{\mathrm{V_{free}}\setminus\mathrm{Y}}\cup\epsilon)) = \emptyset$, according to the first case shown above we know that $\overline{\exists x B}(\mathrm{id}|_{\mathrm{V_{free}}\setminus\mathrm{Y}}\cup\epsilon)$ (or else $\forall x B(\mathrm{id}|_{\mathrm{V_{free}}\setminus\mathrm{Y}}\cup\epsilon)$) is false w.r.t. $\mathcal{A}_{\iota\cup\varepsilon}$. By Lemma 141 we know that $\overline{\exists x B}$ (or else $\forall x B$) is false w.r.t. $\mathcal{A}_{(\mathrm{id}|_{\mathrm{V_{free}}\setminus\mathrm{Y}}\cup\epsilon)\mathcal{A}_{\iota\cup\varepsilon}}$, which finishes the proof of the δ-case due to $(\mathrm{id}|_{\mathrm{V_{free}}\setminus\mathrm{Y}}\cup\epsilon)\mathcal{A}_{\iota\cup\varepsilon} = \iota\cup\varepsilon$.

Q.e.d. (Lemma 143)

Proof of Theorem 144

The soundness direction of the logical equivalence follows from Corollary 140. The completeness without the Cut-rule and the non-restricted Instantiation-rule can be shown as follows:

For each sequent Γ there exists some fair proof tree. If it is closed, then Γ is provable. Otherwise there is some non-closed fair branch that starts from the root and does not end before a leaf is reached. Let DHS be the set of all formulas of all variants of all sequents of this branch. Now DHS is a dual Hintikka set (cf. Definition 142):

The α/β-, α-, β-, γ- and δ-Closure properties are trivial since the branch is fair. For the Instantiation-Closure one has to notice that, for $t \in \mathcal{T}(\mathrm{sig},\mathrm{V}_\forall)$, $\overline{(\mathrm{Def}\,t)} \in \mathrm{DHS}$ implies that $\overline{(\mathrm{Def}\,t)}$ appears in some sequent of the branch and thus in all sequents below. Since the principal formula C of the Restricted Instantiation-rule can never be removed, fairness implies the Instantiation-Closure property, too. Since we consider all variants of formulas and since atoms as well as formulas of the form $\overline{(l{=}r)}$ are never removed from a sequent, DHS also satisfies the Leibniz-Substitutability. Finally, since the branch is not closed and literals are never removed from sequents, DHS is Reflexive-, Def-, and Predicate-Consistent.

By Lemma 143, Γ is not deductively valid. Thus, all deductively valid sequents are provable in our sequent calculus.

Q.e.d. (Theorem 144)

References

Jürgen Avenhaus, Klaus Becker (1992). *Conditional Rewriting modulo a Built-in Algebra.* SEKI-Report SR–92–11 (SFB), Fachbereich Informatik, Universität Kaiserslautern.

Jürgen Avenhaus, Klaus Becker (1994). *Operational Specifications with Built-Ins.* 11$^{\text{th}}$ STACS 1994, LNCS 775, pp. 263-274, Springer-Verlag.

Jürgen Avenhaus, Klaus Madlener (1989). *Term Rewriting and Equational Reasoning.* In: R. B. Banerji (eds.). *Formal Techniques in Artificial Intelligence.* Academic Press.

Jürgen Avenhaus, Klaus Madlener (1995). *Theorem Proving in Hierarchical Clausal Specifications.* SEKI-Report SR–95–14 (SFB), Fachbereich Informatik, Universität Kaiserslautern.

Matthias Baaz, Christian G. Fermüller (1995). *Non-elementary Speedups between Different Versions of Tableaux.* 4$^{\text{th}}$ TABLEAUX 1995, LNAI 918, pp. 217-230, Springer-Verlag.

Leo Bachmair (1988). *Proof By Consistency in Equational Theories.* 3$^{\text{rd}}$ IEEE symposium on Logic In Computer Science, pp. 228-233.

Leo Bachmair, Harald Ganzinger (1994). *Rewrite-based Equational Theorem Proving with Selection and Simplification.* J. Logic Computat. **4**, pp. 217-247, Oxford University Press.

Klaus Becker (1993). *Proving Ground Confluence and Inductive Validity in Constructor Based Equational Specifications.* TAPSOFT 1993, LNCS 668, pp. 46-60, Springer-Verlag.

Klaus Becker (1994). *Rewrite Operationalization of Clausal Specifications with Predefined Structures.* PhD thesis, Fachbereich Informatik, Universität Kaiserslautern.

Rudolf Berghammer (1993). *On the characterization of the integers: The hidden function problem revisited.* Acta Cybernetica, Vol. 11, No. 1-2, Szeged.

Eddy Bevers, Johan Lewi (1990). *Proof by Consistency in Conditional Equational Theories.* Report CW 102, Revised July 1990. Department of Computer Science, K. U. Leuven. Short version in: 2$^{\text{nd}}$ CTRS 1990, LNCS 516, pp. 194-205, Springer-Verlag.

Wolfgang Bibel, E. Eder (1993). *Methods and Calculi for Deduction.* In: Gabbay & al. (1993 ff.), Vol. 1, pp. 67-182.

Susanne Biundo, Birgit Hummel, Dieter Hutter, Christoph Walther (1986). *The Karlsruhe Induction Theorem Proving System.* 8$^{\text{th}}$ CADE 1986, LNCS 230, pp. 672-674, Springer-Verlag.

Adel Bouhoula, Michaël Rusinowitch (1995). *Implicit Induction in Conditional Theories.* J. Automated Reasoning **14**, pp. 189-235, Kluwer Acad. Publ..

Robert S. Boyer, J Strother Moore (1979). *A Computational Logic.* Academic Press.

Robert S. Boyer, J Strother Moore (1988). *A Computational Logic Handbook.* Academic Press.

François Bronsard, Uday S. Reddy (1991). *Conditional Rewriting in Focus.* 2nd CTRS 1990, LNCS 516, pp. 2-13, Springer-Verlag.

Anatoli Degtyarev, Andrei Voronkow (1995). *Reduction of Second-Order Unification to Simultaneous Rigid E-Unification.* UPMAIL Technical Report 109, Computing Science Dept., Uppsala University.

Nachum Dershowitz (1987). *Termination of Rewriting.* J. Symbolic Computation (1987) **3**, pp. 69-116, Academic Press.

Nachum Dershowitz, Mitsuhiro Okada, G. Sivakumar (1988). *Confluence of Conditional Rewrite Systems.* 1st CTRS 1988, LNCS 308, pp. 31-44, Springer-Verlag.

Hartmut Ehrig, Bernd Mahr (1985). *Fundamentals of Algebraic Specification 1.* Springer-Verlag.

Herbert B. Enderton (1973). *A mathematical introduction to logic.* Academic Press, 2nd printing.

Melvin Fitting (1990). *First-Order Logic and Automated Theorem Proving.* Springer-Verlag. 2nd (extended) edition 1996.

Ulrich Fraus (1993). *A Calculus for Conditional Inductive Theorem Proving.* 3rd CTRS 1992, LNCS 656, pp. 357-362, Springer-Verlag.

Ulrich Fraus (1994). *Mechanizing Inductive Theorem Proving in Conditional Theories.* PhD thesis, Universität Passau.

Dov M. Gabbay, C. J. Hogger, J. A. Robinson (eds.) (1993 ff.). *Handbook of Logic in Artificial Intelligence and Logic Programming.* Clarendon Press.

Harald Ganzinger (1988). *A Completion Procedure for Conditional Equations.* 1st CTRS 1988, LNCS 308, pp. 62-83, Springer-Verlag.

Harald Ganzinger, Jürgen Stuber (1992). *Inductive Theorem Proving by Consistency for First-Order Clauses.* In: Informatik-Festschrift zum 60. Geburtstag von Günter Hotz. Teubner Verlag, Stuttgart. Also in: 3rd CTRS 1992, LNCS 656, pp. 226-241, Springer-Verlag.

Gerhard Gentzen (1935). *Untersuchungen über das logische Schließen.* Mathematische Zeitschrift **39**, pp. 176-210, 405-431.

Kurt Gödel (1931). *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I.* Monatshefte für Mathematik und Physik, Vol. 38, pp. 173-198.

Martin Gogolla (1983). *Algebraic Specifications with Partially Ordered Sorts and Declarations.* Forschungsbericht Nr. 169. Universität Dortmund.

Bernhard Gramlich (1989). *Inductive Theorem Proving Using Refined Unfailing Completion Techniques.* SEKI-Report SR–89–14 (SFB), Fachbereich Informatik, Universität Kaiserslautern. Short version in: 9th ECAI 1990, pp. 314-319, Pitman Publishing.

Bernhard Gramlich, Wolfgang Lindner (1991). *A Guide to UNICOM, an Inductive Theorem Prover Based on Rewriting and Completion Techniques.* SEKI-Report SR–91–17 (SFB), Fachbereich Informatik, Universität Kaiserslautern.

S. Kanger (1963). *A simplified proof method for elementary logic.* In: Siekmann & Wrightson (1983), Vol. 1, pp. 364-371.

Stéphane Kaplan (1987). *Simplifying Conditional Term Rewriting Systems: Unification, Termination and Confluence.* J. Symbolic Computation (1987) **4**, pp. 295-334, Academic Press.

Stéphane Kaplan (1988). *Positive/Negative Conditional Rewriting.* 1st CTRS 1988, LNCS 308, pp. 129-143, Springer-Verlag.

Deepak Kapur, Hantao Zhang (1989). *An Overview of Rewrite Rule Laboratory (RRL).* 3rd RTA 1989, LNCS 355, pp. 559-563, Springer-Verlag.

Jan Willem Klop (1992). *Term Rewriting Systems.* In: S. Abramsky, Dov M. Gabbay, T. S. E. Maibaum (eds.). *Handbook of Logic in Computer Science, Vol. 2.* Clarendon Press.

Ulrich Kühler (1991). *Ein funktionaler und struktureller Vergleich verschiedener Induktionsbeweiser.* Diplomarbeit, Fachbereich Informatik, Universität Kaiserslautern.

Vladimir A. Lifschitz (1971). *Specialization of the Form of Deduction in the Predicate Calculus with Equality and Function Symbols.* In: Orevkov (1971), pp. 1-24.

Vladimir A. Lifschitz (1989). *What Is the Inverse Method?.* J. Automated Reasoning **5**, pp. 1-23, Kluwer Acad. Publ..

Sergey Yu. Maslov (1971). *The Inverse Method for Establishing Deducibility for Logic Calculi.* In: Orevkov (1971), pp. 25-96.

Aart Middeldorp, Erik Hamoen (1994). *Completeness Results for Basic Narrowing.* J. of Applicable Algebra in Engineering, Communication and Computing (AAECC) **5**, pp. 313-253, Springer-Verlag.

Vladimir P. Orevkov (1971). *The Calculi of Symbolic Logic.I.* American Mathematical Society, Providence, Rhode Island.

Peter Padawitz (1992). *Deduction and Declarative Programming.* Cambridge University Press.

Dag Prawitz (1960). *An Improved Proof Procedure.* In: Siekmann & Wrightson (1983), Vol. 1, pp. 259-199.

Dag Prawitz (1965). *Natural Deduction.* Almquist & Wiksells, Uppsala.

Martin Protzen (1994). *Lazy Generation of Induction Hypotheses.* 12th CADE 1994, LNAI 814, pp. 42-56, Springer-Verlag.

Uday S. Reddy (1990). *Term Rewriting Induction.* 10th CADE 1990, LNAI 449, pp. 162-177, Springer-Verlag.

Jörg Siekmann, G. Wrightson (eds.) (1983). *Automation of Reasoning.* Springer-Verlag.

Gerd Smolka, Werner Nutt, Joseph A. Goguen, José Meseguer (1989). *Order-Sorted Equational Computation.* In: H. Ait-Kaci, M. Nivat (eds.). *Resolution of Equations in Algebraic Structures, Vol. 2.* Academic Press.

Raymond M. Smullyan (1968). *First-Order Logic.* Springer-Verlag.

Christoph Walther (1992). *Computing Induction Axioms.* 3rd LPAR 1992, LNAI 624, pp. 381-392, Springer-Verlag.

Christoph Walther (1994). *Mathematical Induction.* In: Gabbay & al. (1993 ff.), Vol. 2.

Hao Wang (1960). *Toward Mechanical Mathematics.* In: Siekmann & Wrightson (1983), Vol. 1, pp. 244-264.

Claus-Peter Wirth (1991). *Inductive Theorem Proving in Theories specified by Positive/Negative Conditional Equations.* Diplomarbeit, Fachbereich Informatik, Universität Kaiserslautern.

Claus-Peter Wirth (1995). *Syntactic Confluence Criteria for Positive/Negative-Conditional Term Rewriting Systems.* SEKI-Report SR–95–09 (SFB), Fachbereich Informatik, Universität Kaiserslautern.

Claus-Peter Wirth, Klaus Becker (1995). *Abstract Notions and Inference Systems for Proofs by Mathematical Induction.* 4th CTRS 1994, LNCS 968, pp. 353-373, Springer-Verlag.

Claus-Peter Wirth, Bernhard Gramlich (1993). *A Constructor-Based Approach for Positive/Negative-Conditional Equational Specifications.* 3rd CTRS 1992, LNCS 656, pp. 198-212, Springer-Verlag. Revised and extended version is Wirth & Gramlich (1994a).

Claus-Peter Wirth, Bernhard Gramlich (1994a). *A Constructor-Based Approach for Positive/Negative-Conditional Equational Specifications.* J. Symbolic Computation (1994) **17**, pp. 51-90, Academic Press.

Claus-Peter Wirth, Bernhard Gramlich (1994b). *On Notions of Inductive Validity for First-Order Equational Clauses.* 12th CADE 1994, LNAI 814, pp. 162-176, Springer-Verlag.

Claus-Peter Wirth, Rüdiger Lunde (1994). *Writing Positive/Negative-Conditional Equations Conveniently.* SEKI-Working-Paper SWP–94–04 (SFB), Fachbereich Informatik, Universität Kaiserslautern.

Claus-Peter Wirth, Bernhard Gramlich, Ulrich Kühler, Horst Prote (1993). *Constructor-Based Inductive Validity in Positive/Negative-Conditional Equational Specifications.* SEKI-Report SR–93–05 (SFB), Fachbereich Informatik, Universität Kaiserslautern. Revised and extended version of first part is Wirth & Gramlich (1994a), revised version of second part is Wirth & Gramlich (1994b).

Hantao Zhang (1993). *Implementing Contextual Rewriting.* 3rd CTRS 1992, LNCS 656, pp. 363-377, Springer-Verlag.

Hantao Zhang, Deepak Kapur (1988). *First-Order Theorem Proving using Conditional Rewrite Rules.* 9th CADE 1988, LNAI 310, pp. 1-20, Springer-Verlag.

Acknowledgements: We would like to thank Jürgen Avenhaus, Klaus Becker, Roland Fettig, Alfons Geser, Bernhard Gramlich, Klaus Madlener, Rodrigo Readi-Nasser, Birgit Reinert, and Andrea Sattler-Klein for many fruitful discussions.