Conditional Equational Specifications of Data Types with Partial Operations for Inductive Theorem Proving

Ulrich Kühler
Claus-Peter Wirth

SEKI-Report SR–96–11

November 20, 1996

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-67653 Kaiserslautern

{kuehler,wirth}@informatik.uni-kl.de

Abstract

We propose a specification language for the formalization of data types with partial or non-terminating operations as part of a rewrite-based logical framework for inductive theorem proving. The language requires constructors for designating data items and admits positive/negative conditional equations as axioms in specifications. The (total algebra) semantics for such specifications is based on so-called data models. We present admissibility conditions that guarantee the unique existence of a distinguished data model with properties similar to those of the initial model of a usual equational specification. Since admissibility of a specification requires confluence of the induced rewrite relation, we provide an effectively testable confluence criterion which does not presuppose termination.

1 Introduction

Data types such as the natural numbers, lists, strings, trees, graphs etc. are essential for the design and implementation of most software systems. In computer science a collection D of data domains and operations on these data domains is usually called a *data type* if all data items in the data domains of D are finitely generated by the operations of D. Therefore, mathematical induction – as the proof method corresponding to finitely generated or *inductively* defined objects – constitutes the basis of a suitable formal method for reasoning about data types. Since formal methods are indispensable to verification activities in the development process of safety-critical algorithms, proof by mathematical induction or *inductive theorem proving* (ITP) is likely to gain economic significance in the next few years.

In this paper we propose an algebraic (i.e. equational) specification language for the formalization of data types with partial operations which is part of a new first-order and rewrite-based logical framework for ITP. Essentially, our specification language is given by its syntax, its (inductive) semantics and its *admissibility conditions*. While the syntax determines the signature and the set of axioms admitted in a specification, the semantics indicates what particular model class is to be associated with a specification as its meaning. Furthermore, the admissibility conditions have to guarantee that the semantics is actually meaningful, i.e. the associated model class is not empty. Obviously, inductive inference methods for formal reasoning about data types form the other integral part of a logical framework for ITP. We refer to [WK95] for an inference system on the basis of the proposed specification language.

The simple examples below are intended to motivate the major objectives which have guided the development of our specification language.

Example 1.1 Let D be a data type that comprises the natural numbers as its data domain and the division along with other arithmetic operations (see below). The following set E of conditional equations could be the set of axioms in an algebraic specification $spec = (sig, E)$ of D (s denotes the successor-function).

$$\begin{array}{ll|ll}
\mathsf{plus}(x,0) & = x & \mathsf{times}(x,0) & = 0 \\
\mathsf{plus}(x,\mathsf{s}(y)) & = \mathsf{s}(\mathsf{plus}(x,y)) & \mathsf{times}(x,\mathsf{s}(y)) & = \mathsf{plus}(\mathsf{times}(x,y),x) \\
 & & & \\
 & & \mathsf{less}(x,0) & = \mathsf{false} \\
\mathsf{minus}(x,0) & = x & \mathsf{less}(0,\mathsf{s}(y)) & = \mathsf{true} \\
\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) & = \mathsf{minus}(x,y) & \mathsf{less}(\mathsf{s}(x),\mathsf{s}(y)) & = \mathsf{less}(x,y)
\end{array}$$

$$\begin{array}{lcl}
\mathsf{div}(x,y) = 0 & \leftarrow & y \neq 0 \ \wedge \ \mathsf{less}(x,y) = \mathsf{true} \\
\mathsf{div}(x,y) = \mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),y)) & \leftarrow & y \neq 0 \ \wedge \ \mathsf{less}(x,y) = \mathsf{false}
\end{array}$$

Although E yields an appropriate axiomatization of D, the axioms in E (or their respective formulations) are not admissible wrt. the specification formalisms of various first-order frameworks for ITP: Firstly, $spec = (sig, E)$ is not *sufficiently complete* wrt. the *constructors* true, false, $\mathsf{0}$ and s (see [Wir90]), since neither minus nor div are completely defined by E. However, in the specification formalisms of inductive theorem provers such as Spike (see [BR95]), Nqthm (see [BM79]) or Inka (see [HS96]), each non-constructor operation must be completely defined, i.e. for a *partial* operation (such as subtraction or division) some of its total extensions has to be axiomatized. Secondly, the rewrite-based specification languages of Spike and Rrl (see [KS96]) require the left-hand side of each conditional equation to be greater than any other term in this conditional equation wrt. a reduction order. Since $\mathsf{div}(x, y)$ is *smaller* than $\mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), y))$ wrt. any simplification order (see [DJ90]), it is fairly difficult to prove such admissibility of E. Thirdly, two conditional equations in E each contain a *negative* condition, namely $y \neq \mathsf{0}$. This is ruled out in [BR95] e.g.

Example 1.2 *(Example 1.1 continued)* A tail-recursive variant of div that is non-terminating but "efficient" on its domain can be axiomatized as follows:

$$\begin{array}{lcl} \mathsf{div1}(x, y, z_1, z_2) = z_1 & \leftarrow & x = z_2 \\ \mathsf{div1}(x, y, z_1, z_2) = \mathsf{div1}(x, y, \mathsf{s}(z_1), \mathsf{plus}(z_2, y)) & \leftarrow & x \neq z_2 \end{array}$$

If $n > 0$ then $\mathsf{div1}(\mathsf{s}^{n \cdot m}(\mathsf{0}), \mathsf{s}^n(\mathsf{0}), \mathsf{0}, \mathsf{0})$ can be evaluated to $\mathsf{s}^m(\mathsf{0})$ with E', where E' consists of E and the two axioms for $\mathsf{div1}$. Hence, the equational clause

$$y = \mathsf{0} \;\vee\; \mathsf{times}(y, z) \neq x \;\vee\; \mathsf{div1}(x, y, \mathsf{0}, \mathsf{0}) = z \tag{1}$$

formalizes an intuitively "true" statement and should be valid in the class of models (i.e. the semantics) associated with the specification $spec' = (sig', E')$.

Due to their requirement that axiomatizations of operations "terminate", none of the frameworks for ITP described in [BR95], [BM79], [HS96] or [KS96] accept E' (or its respective formulations) as an admissible set of axioms.

In the remainder of the paper we present an algebraic specification language which is to allow adequate formalizations of data types with partial and non-terminating operations. The specification language requires constructor symbols for each sort in a signature so that all data items in a data type can be designated with constructor ground terms, and provides *constructor variables* which range over data items only. Moreover, conditional equations (or rewrite rules) with positive *and* negative conditions are admitted as axioms in our so-called *specifications with constructors*. In addition to the usual notion of a model we define so-called *data models* as a basis for a suitable (total algebra) semantics of specifications with constructors (Sect. 3). For every *admissible* specification *spec* the unique existence of a "best" data model $\mathcal{M}(spec)$ is guaranteed – "best" in the sense that $\mathcal{M}(spec)$ has interesting algebraic properties resembling those of the initial model of a usual equational specification (Sect. 4). The appropriateness of our semantics is underlined by an important monotonicity result: Contrary to initial algebra semantics, the extension of an admissible specification in a "consistent" way does not result in the loss of inductive theorems. In other words, every formula valid in the class of all data models of the original specification remains valid in the class of all data models of the extended specification (Sect. 6).

Essentially, admissibility of a specification with constructors *spec* means that the rewrite relation which we associate with the positive/negative conditional rewrite rules in *spec* is confluent. Since we are also interested in formalizing data types with non-terminating operations, we provide a confluence criterion in Sect. 5 which does not presuppose termination of the rewrite relation but is based on simple syntactic properties of the specifying rewrite system. As a consequence, we obtain easily testable admissibility conditions that are fulfilled by many relevant specifications including the ones in Examples 1.1 and 1.2. Note that all proofs can be found in the Appendix.

It should be mentioned that the specification language proposed in this paper has its origin in the more general specification approach of [WG94a] and [WG94b]. We have adapted the latter to the requirements of practical ITP, which facilitated a simplified presentation, and augmented it with new and effectively testable admissibility conditions.

2 Basic Notions and Notations

We assume that the reader is familiar with the basic terminology of algebraic specification and rewriting. For more details we refer to [EM85], [Wir90] and [DJ90].

A many-sorted *signature* $sig = (S, F, \alpha)$ comprises a set S of sort symbols, a set F of function symbols and an arity function α mapping F into S^+. For $f \in F$, $\alpha(f) = s_1 \dots s_n s$ indicates the argument sorts $s_1 \dots s_n$ and the result sort s of f. For every signature $sig = (S, F, \alpha)$, we assume a fixed S-sorted family of mutually disjoint sets of *variables* $V = (V_s)_{s \in S}$ where $F \cap V = \emptyset$. The well-sorted *terms* (over sig and V) are denoted by $\mathcal{T}(sig, V) = (\mathcal{T}(sig, V)_s)_{s \in S}$, and $\mathcal{GT}(sig) = (\mathcal{GT}(sig)_s)_{s \in S}$ is used for the ground terms (over sig).

A *position* (or occurrence) p within a term t is a sequence of positive integers. By t/p, we denote the sub-term of t at position p, and $t[u]_p$ stands for the term t with its sub-term t/p replaced with a term u. We use $\mathrm{Pos}(t)$ for the set of all positions of t. A term t is called *linear* if no variable occurs more than once in t.

A *sig-algebra* $\mathcal{A} = (A, F^{\mathcal{A}})$ is given by $A = (A_s)_{s \in S}$ and $F^{\mathcal{A}} = (f^{\mathcal{A}})_{f \in F}$ where

(a) for all $s \in S$, A_s is a non-empty set called the carrier of $\mathcal{A}$ for s; and

(b) for all $f \in F$ with $\alpha(f) = s_1 \dots s_n s$, $f^{\mathcal{A}}\colon A_{s_1} \times \dots \times A_{s_n} \to A_s$ is a function.

Let $\mathcal{B} = (B, F^{\mathcal{B}})$ be another sig-algebra. A *sig-homomorphism* $h\colon \mathcal{A} \to \mathcal{B}$ is a family $h = (h_s)_{s \in S}$ of functions $h_s\colon A_s \to B_s$ such that for all $f \in F$ and for all $a_i \in A_{s_i}$

$$h_s(f^{\mathcal{A}}(a_1, \dots, a_n)) = f^{\mathcal{B}}(h_{s_1}(a_1), \dots, h_{s_n}(a_n))$$

where $\alpha(f) = s_1 \dots s_n s$. If all functions $h_s\colon A_s \to B_s$ are bijective (surjective), then h is called a *sig-isomorphism* (*sig-epimorphism*).

A sig-algebra $\mathcal{I}$ is *initial* in a class K of sig-algebras if $\mathcal{I} \in K$ and for every $\mathcal{A} \in K$ there is a unique sig-homomorphism from $\mathcal{I}$ to $\mathcal{A}$. Note that the ground term algebra $\mathcal{GT}(sig)$ is initial in the class of all sig-algebras. By $\mathrm{eval}^{\mathcal{A}}$, we denote the unique (initial) sig-homomorphism from $\mathcal{GT}(sig)$ to a sig-algebra $\mathcal{A}$, which is defined by

$$\mathrm{eval}^{\mathcal{A}}(f(t_1, \dots, t_n)) = f^{\mathcal{A}}(eval^{\mathcal{A}}(t_1), \dots, eval^{\mathcal{A}}(t_n))$$

for all $f \in F$ and for all $t_i \in \mathcal{GT}(sig)_{s_i}$. We usually write $t^{\mathcal{A}}$ instead of $\mathrm{eval}^{\mathcal{A}}(t)$.

A *sig-congruence* on $\mathcal{A}$ is a family $\sim \; = \; (\sim_s)_{s \in S}$ of equivalences $\sim_s$ on A_s such that $a_i \sim_{s_i} b_i$ for $i = 1, \dots, n$ implies $f^{\mathcal{A}}(a_1, \dots, a_n) \sim_s f^{\mathcal{A}}(b_1, \dots, b_n)$ for all $f \in F$. Every sig-homomorphism $h\colon \mathcal{A} \to \mathcal{B}$ induces a sig-congruence on $\mathcal{A}$, namely the *kernel* $\ker(h)$ of h defined by $\ker(h)_s = \{\, (a, b) \mid a, b \in A_s \text{ and } h_s(a) = h_s(b) \,\}$ for all $s \in S$. The *quotient algebra* $\mathcal{A}/{\sim}$ of $\mathcal{A}$ modulo $\sim$ is the sig-algebra $\mathcal{Q} = (Q, F^{\mathcal{Q}})$ satisfying

(a) for all $s \in S$, $Q_s = \{\, [a] \mid a \in A_s \,\}$ where $[a] = \{\, b \in A_s \mid a \sim b \,\}$; and

(b) for all $f \in F$ and for all $a_i \in A_{s_i}$, $\quad f^{\mathcal{Q}}([a_1], \dots, [a_n]) = [f^{\mathcal{A}}(a_1, \dots, a_n)]$.

Let $X \subseteq V$ be an S-sorted family of variables and $\longrightarrow$ be a relation on $\mathcal{T}(sig, X)$. Then $\longleftarrow$ is its reverse, $\longleftrightarrow$ its symmetric closure, $\stackrel{+}{\longrightarrow}$ its transitive closure and $\stackrel{*}{\longrightarrow}$ its transitive-reflexive closure. By $\downarrow$, we denote the *joinability* relation $\stackrel{*}{\longrightarrow} \circ \stackrel{*}{\longleftarrow}$ of $\longrightarrow$, that is $t_1 \downarrow t_2$ if there is a t such that $t_1 \stackrel{*}{\longrightarrow} t \stackrel{*}{\longleftarrow} t_2$.

Moreover, $\longrightarrow$ is called *confluent* if $\stackrel{*}{\longleftarrow} \circ \stackrel{*}{\longrightarrow} \subseteq \downarrow$. We call $\longrightarrow$ *monotonic* if $t_1 \longrightarrow t_2$ implies $t[t_1]_p \longrightarrow t[t_2]_p$ for all $t_1, t_2, t \in \mathcal{T}(sig, X)$ and for all $p \in \mathrm{Pos}(t)$ where $t_1, t_2, t/p \in \mathcal{T}(sig, X)_s$ for some $s \in S$. We speak of *sort-invariance* of $\longrightarrow$ if $t_1 \longrightarrow t_2$ entails that $t_1, t_2 \in \mathcal{T}(sig, X)_s$ for some $s \in S$. Note that $\stackrel{*}{\longleftrightarrow}$ is a sig-congruence on $\mathcal{T}(sig, X)$ if $\longrightarrow$ is monotonic and sort-invariant. A term t is $\longrightarrow$-*irreducible* if there is no t' such that $t \longrightarrow t'$.

3 Specifications with Constructors

As mentioned before in the introduction, our interest in inductive theorem proving is mainly due to its fundamental significance to methods that allow formal reasoning about data types. Formal proofs of statements which express valid properties of the operations of a given data type require a preceding formalization of the data type. To state more precisely what we mean by a "data type" we quote the following conceptual definition from [EM85]:

> "A data type is a collection of data domains, designated basic data items, and operations on these domains such that all data items of the data domains can be generated from the basic data items by use of the operations. Moreover the data domains are assumed to be countable."

It is generally accepted that *initial algebra semantics* for usual (positive conditional) equational specifications is rarely appropriate when data types with partial or non-terminating operations have to be formalized (see [Wir90], [WG94b]). Consider e.g. the specification whose axioms are those from Example 1.1 that define minus and less. The carrier of its initial model for the sort bool consists of *infinitely many* elements represented by "junk terms" like $\mathsf{less}(0, \mathsf{minus}(0, \mathsf{s}^{n+1}(0)))$ instead of just *two* (for true and false). In some cases, lack of sufficient completeness as the essential problem can be avoided by demanding that the specification describe total extensions of the partial operations. For our specifications with constructors, however, we do not require sufficient completeness for adequate representations of data types with partial operations any more – mainly because we acknowledge the importance constructors have in describing the data items of a data type.

3.1 Syntax of Specifications with Constructors

In order to ensure that a data type D with partial operations will be adequately represented by certain model classes of a specification of D (see below), our specification language requires the user to indicate the constructors for D, i.e. those function symbols which are needed for designating the data items of D. Let $sig = (S, F, \alpha)$ be a signature. Formally, a subset $C \subseteq F$ is said to be a set of *constructors* for sig if the signature $sig^C = (S, C, \alpha|_C)$ induced by C is *sensible*, i.e. for each $s \in S$ there is at least one constructor ground term $t \in \mathcal{GT}(sig^C)_s$ of sort s. We call sig^C the *constructor signature* of sig.

We assume that for each sort $s \in S$, the set V_s of variables for s is composed of two disjoint subsets V_s^{C} and V_s^{G}. The elements of V^{C} are called *constructor variables*, while the elements of V^{G} are called *general variables*. Intuitively, constructor variables range over data items only, whereas general variables allow statements about undefined objects as well. $\mathcal{T}(sig^C, V^{\mathrm{C}})$ denotes the set of (pure) *constructor terms*. A substitution σ: $V \to \mathcal{T}(sig, V)$ is said to be a *constructor substitution* if $\sigma(V^{\mathrm{C}}) \subseteq \mathcal{T}(sig^C, V^{\mathrm{C}})$, and we call σ an *inductive substitution* if $\sigma(V^{\mathrm{C}}) \subseteq \mathcal{GT}(sig^C)$ and $\sigma(V^{\mathrm{G}}) \subseteq \mathcal{T}(sig, V^{\mathrm{G}})$.

An *equation* is a pair $t_1 = t_2$ such that $t_1, t_2 \in \mathcal{T}(sig, V)_s$ for some $s \in S$. An *atom* is an equation or a *definedness* atom $\mathrm{def}(t)$ where 'def' is a predefined predicate symbol and $t \in \mathcal{T}(sig, V)$. Informally, $\mathrm{def}(t)$ means that the applications of operations denoted by t can be evaluated to data items. A *positive literal* is an atom, and a *negative literal* is a negated atom. A *literal* is a positive or a negative literal. The *complement* $\overline{\lambda}$ of a positive literal λ is $\neg\lambda$, and the complement $\overline{\neg\lambda}$ of a negative literal $\neg\lambda$ is λ. A (disjunctive) *clause* is a possibly empty sequence $\lambda_1 \ldots \lambda_n$ of literals.

A (positive) *conditional equation* is an expression of the form $l = r \leftarrow \Delta$ where Δ is a possibly empty sequence of (positive) *condition literals*. The clause representation of $l = r \leftarrow \lambda_1 \ldots \lambda_n$ is $(l = r)\,\overline{\lambda_1} \ldots \overline{\lambda_n}$. When emphasizing its directed use we call a conditional equation a (conditional) *rewrite rule*.

Given an expression e (e.g. a term, a clause or a rewrite rule), let $\mathrm{Var}(e)$ denote the set of variables in e.

Definition 3.1 A *specification with constructors* $spec = (sig, C, E)$ is composed of a signature sig such that C is a set of constructors for sig, and of a set E of conditional equations (over sig and V).

Note that at this point we do not place any restrictions on the conditional equations to be used in specifications. In Sect. 4, however, we will develop restricting admissibility conditions which will guarantee that appropriate notions of inductive semantics are actually meaningful for admissible specifications.

3.2 Model Semantics of Specifications with Constructors

We now have to determine the elements of a sig-algebra $\mathcal{A}$ which may be assigned to constructor variables. As constructor variables are meant to range over data items only, we assign those elements of $\mathcal{A}$ which are designated by constructor ground terms to constructor variables. These elements form the carriers of a sig^C-algebra, which we call the *data reduct* $\mathcal{A}^C$ of $\mathcal{A}$:

Definition 3.2 Let $\mathcal{A} = (A, F^{\mathcal{A}})$ be a sig-algebra. The *data reduct* of $\mathcal{A}$ is the sig^C-algebra $\mathcal{A}^C = (A^C, C^{\mathcal{A}^C})$ satisfying

(a) for each $s \in S$, $A^C_s = \{\, t^{\mathcal{A}} \in A_s \mid t \in \mathcal{GT}(sig^C)_s \,\}$; and

(b) for each $c \in C$ and for all $a_i \in A^C_{s_i}$, $c^{\mathcal{A}^C}(a_1, \ldots, a_n) = c^{\mathcal{A}}(a_1, \ldots, a_n)$ where $\alpha(c) = s_1 \ldots s_n s$.

The data reduct $\mathcal{A}^C$ is in fact a sig^C-algebra: Since C is a set of constructors for sig, sig^C is sensible, and so $A^C_s \neq \emptyset$ for each $s \in S$. Furthermore, let $a_1, \ldots, a_n \in A^C$. Then there are $t_1, \ldots, t_n \in \mathcal{GT}(sig^C)$ such that $c^{\mathcal{A}}(a_1, \ldots, a_n) = c^{\mathcal{A}}(t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}})$, and since $\mathrm{eval}^{\mathcal{A}}\colon \mathcal{GT}(sig) \to \mathcal{A}$ is a sig-homomorphism, we have $c^{\mathcal{A}}(t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}}) = c(t_1, \ldots, t_n)^{\mathcal{A}}$. Hence, $c^{\mathcal{A}}(a_1, \ldots, a_n) \in A^C$, which shows that A^C is closed under $c^{\mathcal{A}}$ for each $c \in C$.[1]

Given an S-sorted family of variables $X \subseteq V$, the data reduct of the term algebra $\mathcal{T}(sig, X)$ is obviously $\mathcal{GT}(sig^C)$. Moreover, the image of a data reduct under a sig-homomorphism is a data reduct:

Lemma 3.3 *Let $\mathcal{A}$ and $\mathcal{B}$ be sig-algebras, and let $h\colon \mathcal{A} \to \mathcal{B}$ be a sig-homomorphism. Let $h^C = (h^C_s)_{s\in S}$ be the family of functions defined by $h^C_s = h_s|_{A^C_s}$ for all $s \in S$. Then $h^C\colon \mathcal{A}^C \to \mathcal{B}^C$ is a sig^C-epimorphism.*

We can now give meaning to terms, literals and clauses.

Definition 3.4 Let $\mathcal{A} = (A, F^{\mathcal{A}})$ be a sig-algebra.

(i) Let $X \subseteq V$. A *valuation* of X in $\mathcal{A}$ is a function $\varphi\colon X \to \mathcal{A}$ such that $\varphi(x) \in A^C_s$ for every $x \in X_s \cap V^{\mathrm{C}}_s$ and $\varphi(x) \in A_s$ for every $x \in X_s \cap V^{\mathrm{G}}_s$. By $\mathrm{eval}^{\mathcal{A}}_{\varphi}$ we denote the unique sig-homomorphism from $\mathcal{T}(sig, X)$ to $\mathcal{A}$ that extends φ.

[1] $\mathcal{A}^C$ is the (sig^C-) homomorphic image of $\mathcal{GT}(sig^C)$ under $\mathrm{eval}^{\mathcal{A}}$, and hence a term-generated sub-algebra of the sig^C-reduct $\mathcal{A}|_{sig^C}$ of $\mathcal{A}$.

(ii) Let φ be a valuation of V in $\mathcal{A}$. Then $\mathcal{A}$ *satisfies* an equation $t_1 = t_2$ *with* φ if $\mathrm{eval}_\varphi^\mathcal{A}(t_1) = \mathrm{eval}_\varphi^\mathcal{A}(t_2)$, and $\mathcal{A}$ satisfies a definedness atom $\mathrm{def}(t)$ for $t \in \mathcal{T}(sig, V)_s$ with φ if $\mathrm{eval}_\varphi^\mathcal{A}(t) \in A_s^C$. Moreover, $\mathcal{A}$ satisfies a negative literal $\neg\lambda$ with φ if $\mathcal{A}$ does not satisfy λ with φ. Finally, $\mathcal{A}$ satisfies a clause Γ with φ if there is a literal in Γ which $\mathcal{A}$ satisfies with φ.

(iii) A clause Γ is *valid* in $\mathcal{A}$ if $\mathcal{A}$ satisfies Γ with every valuation of V in $\mathcal{A}$. This is denoted by $\mathcal{A} \models \Gamma$. Let K be a class of *sig*-algebras and E be a set of clauses. We write $K \models E$ iff $\mathcal{A} \models \Gamma$ for every $\mathcal{A} \in K$ and for every Γ in E.

The inductive substitutions are exactly the valuations of V in $\mathcal{T}(sig, V^\mathrm{G})$. Besides the *equality axioms* of a signature *sig*, there are other clauses which are valid in all *sig*-algebras, e.g. $\mathrm{def}(c(X_1, \ldots, X_n)) \vee \neg\mathrm{def}(X_1) \vee \ldots \vee \neg\mathrm{def}(X_n)$ where $c \in C$ is a constructor and $X_i \in V_{s_i}^\mathrm{G}$ for $i = 1, \ldots, n$.

The following useful result relates valuations and constructor substitutions.

Lemma 3.5 *Let $\mathcal{A}$ be a sig-algebra, $X \subseteq V$, φ be a valuation of X in $\mathcal{A}$ and σ be a constructor substitution with $\sigma(V) \subseteq \mathcal{T}(sig, X)$. Then* $\mathrm{eval}_\varphi^\mathcal{A}(t\sigma) = \mathrm{eval}_{(\mathrm{eval}_\varphi^\mathcal{A} \circ \sigma)}^\mathcal{A}(t)$ *for all $t \in \mathcal{T}(sig, V)$.*

The models of a specification with constructors can now be defined as usual.

Definition 3.6 Let $spec = (sig, C, E)$ be a specification with constructors. A *sig*-algebra $\mathcal{A}$ is called a (*sig*-) *model* of *spec* if (the clause representation of) each conditional equation in E is valid in $\mathcal{A}$. The class of all *sig*-models of *spec* is denoted by $\mathrm{Mod}(spec)$.

Since the clause representation of any conditional equation contains at least one positive literal, every conditional equation (over *sig* and V) is valid in the trivial *sig*-algebra whose carriers consist of one element each. Therefore, $\mathrm{Mod}(spec)$ is not empty for any specification with constructors *spec*. Moreover, when restricting the variables in clauses to constructor variables, validity in $\mathrm{Mod}(spec)$ can be characterized as follows.

Proposition 3.7 *Let $spec = (sig, C, E)$ be a specification with constructors, and let Γ be a clause with $\mathrm{Var}(\Gamma) \subseteq V^\mathrm{C}$. Then the following statements are equivalent:*

(1) *$Mod(spec) \models \Gamma$*

(2) *For every inductive substitution σ there is a literal λ in Γ with $Mod(spec) \models \lambda\sigma$.*

Still, we do not really consider $\mathrm{Mod}(spec)$ an appropriate inductive semantics for a specification with constructors *spec*. The reason for that is that $\mathrm{Mod}(spec)$ includes also those *sig*-models of *spec* which unnecessarily equate or *confuse* data items. In other words, *sig*-models of this kind satisfy equations between constructor ground terms that are not valid in all *sig*-models of *spec*. An extreme example of such a *sig*-model is the trivial *sig*-algebra.

3.3 Data Models

In eliminating those models from Mod(*spec*) that confuse data items we obtain a particularly suited class of models as the semantics for a specification with constructors. Since usually the data reduct of each of the models in the resulting class yields a one-to-one representation of the data domains of the given data type we call these models *data models*.

Definition 3.8 Let $spec = (sig, C, E)$ be a specification with constructors. We say that a *sig*-model $\mathcal{A}$ of *spec* is a *data model* of *spec* if, for all constructor ground terms $t_1, t_2 \in \mathcal{GT}(sig^C)$, $t_1^{\mathcal{A}} = t_2^{\mathcal{A}}$ implies $\mathrm{Mod}(spec) \models t_1 = t_2$. Let DMod(*spec*) denote the class of all data models of *spec*.

Note that DMod(*spec*) may be empty (see below). Moreover, the data reducts of any two data models in DMod(*spec*) are isomorphic:

Lemma 3.9 *Let $\mathcal{A}$ be a sig-model of spec. Then $\mathcal{A}$ is a data model of spec if and only if its data reduct $\mathcal{A}^C$ is initial in the class of sig^C-algebras $\{\, \mathcal{B}^C \mid \mathcal{B} \in Mod(spec) \,\}$.*

Corollary 3.10 *Let $\mathcal{A}$ and $\mathcal{B}$ be data models of spec. Then their data reducts $\mathcal{A}^C$ and $\mathcal{B}^C$ are (sig^C-) isomorphic.*

Consequently, data models do not differ in the evaluation of constructor terms, i.e. for any $\mathcal{A}, \mathcal{B} \in \mathrm{DMod}(spec)$ and $t_1, t_2 \in \mathcal{T}(sig^C, V^C)$ we have $\mathcal{A} \models t_1 = t_2$ iff $\mathcal{B} \models t_1 = t_2$. For general terms, however, the corresponding statement does not hold – not even for ground terms whose definedness is valid in all models of *spec*:

Example 3.11 Let $spec = (sig, C, E)$ be the specification with constructors over the signature $sig = (S, F, \alpha)$ such that $S = \{\mathsf{any}\}$, $C = \{\mathsf{c}_1, \mathsf{c}_2\}$, $F = C \cup \{\mathsf{d}\}$, $\alpha(o) = \mathsf{any}$ for each $o \in F$, and $E = \{\, \mathsf{d} = \mathsf{c}_1 \leftarrow \mathsf{d} \neq \mathsf{c}_2 \,\}$.

Let $\mathcal{A} = (A, F^{\mathcal{A}})$ be the *sig*-algebra with $A_{\mathsf{any}} = \{a_1, a_2\}$, $\mathsf{c}_1^{\mathcal{A}} = a_1$, $\mathsf{c}_2^{\mathcal{A}} = a_2$ and $\mathsf{d}^{\mathcal{A}} = a_1$. Since $\mathsf{c}_1^{\mathcal{A}} \neq \mathsf{c}_2^{\mathcal{A}}$ and $\mathsf{d}^{\mathcal{A}} = \mathsf{c}_1^{\mathcal{A}}$ we have $\mathcal{A} \in \mathrm{DMod}(spec)$. Moreover, let $\mathcal{B} = (B, F^{\mathcal{B}})$ be the *sig*-algebra with $B_{\mathsf{any}} = \{a_1, a_2\}$, $\mathsf{c}_1^{\mathcal{B}} = a_1$, $\mathsf{c}_2^{\mathcal{B}} = a_2$ and $\mathsf{d}^{\mathcal{B}} = a_2$. Now $\mathsf{c}_1^{\mathcal{B}} \neq \mathsf{c}_2^{\mathcal{B}}$ and $\mathsf{d}^{\mathcal{B}} = \mathsf{c}_2^{\mathcal{B}}$, so $\mathcal{B}$ is also a data model of *spec*.

Hence, we have $\mathcal{A}, \mathcal{B} \in \mathrm{DMod}(spec)$, $\mathrm{Mod}(spec) \models \mathrm{def}(\mathsf{d})$ and $\mathsf{d}^{\mathcal{A}} = \mathsf{c}_1^{\mathcal{A}}$, but *not* $\mathsf{d}^{\mathcal{B}} = \mathsf{c}_1^{\mathcal{B}}$.

In section 4, however, we will show that all "defined" terms are uniformly evaluated in all data models of *admissible* specifications (see Theorem 4.8).

We have mentioned before that the class of all data models of a specification with constructors may be empty. In order to demonstrate this and to motivate our admissibility conditions for guaranteeing the existence of data models (see Sect. 4), we give the following two examples.

Example 3.12 Let $sig = (S, F, \alpha)$ be the signature with $S = \{\mathsf{any}\}$, $C = \{\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3\}$, $F = C \cup \{\mathsf{d}\}$, and $\alpha(o) = \mathsf{any}$ for each $o \in F$. Let $E_1 = \{\, \mathsf{c}_1 = \mathsf{c}_2 \leftarrow \mathsf{c}_1 \neq \mathsf{c}_3 \,\}$ and $spec_1 = (sig, C, E_1)$.

Obviously, neither $\mathrm{Mod}(spec_1) \models \mathsf{c}_1 = \mathsf{c}_2$ holds nor $\mathrm{Mod}(spec_1) \models \mathsf{c}_1 = \mathsf{c}_3$, but for any $\mathcal{A} \in \mathrm{Mod}(spec_1)$ we have $\mathsf{c}_1^{\mathcal{A}} = \mathsf{c}_2^{\mathcal{A}}$ or $\mathsf{c}_1^{\mathcal{A}} = \mathsf{c}_3^{\mathcal{A}}$ (because of E_1). Hence $\mathcal{A}$ cannot be a data model of $spec_1$.

Example 3.13 Let sig and C be as in the preceding example, $E_2 = \{\, \mathsf{d} = \mathsf{c}_1 \leftarrow \mathsf{c}_1 \neq \mathsf{c}_3,$ $\mathsf{d} = \mathsf{c}_2 \leftarrow \mathsf{c}_1 \neq \mathsf{c}_3 \,\}$, and let $spec_2 = (sig, C, E_2)$.

One easily shows that neither $\mathrm{Mod}(spec_2) \models \mathsf{c}_1 = \mathsf{c}_3$ nor $\mathrm{Mod}(spec_2) \models \mathsf{c}_1 = \mathsf{c}_2$. Let $\mathcal{A} \in \mathrm{Mod}(spec_2)$. If $\mathsf{c}_1^{\mathcal{A}} = \mathsf{c}_3^{\mathcal{A}}$ then $\mathcal{A}$ is not a data model of $spec_2$. Otherwise, due to E_2, we have $\mathsf{c}_1^{\mathcal{A}} = \mathsf{d}^{\mathcal{A}} = \mathsf{c}_2^{\mathcal{A}}$, so $\mathcal{A}$ is not a data model of $spec_2$ either.

4 Admissibility Conditions

So far, we have not placed any restrictions on the set of positive/negative conditional equations in a specification with constructors. The two preceding examples, however, show that certain restrictions are necessary to ensure that $\mathrm{DMod}(spec)$ is meaningful (i.e. not empty) as an inductive semantics. In this section we therefore present the admissibility conditions of our specification language which guarantee the existence of a distinguished data model $\mathcal{M}(spec)$ for any admissible specification $spec$. Hence, $\mathrm{DMod}(spec) \neq \emptyset$.

Our admissibility conditions are based on terminology and concepts from the theory of term rewriting; recall that every conditional equation can be regarded as a (conditional) rewrite rule. We will show in the following that the rewrite relation $\longrightarrow_R$ associated with an admissible specification $spec = (sig, C, R)$ can be defined in such a way that $\longrightarrow_R$ yields a sound and complete operationalization of equality in all data models: $t_1 \overset{*}{\longleftrightarrow}_R t_2$ iff $\mathrm{DMod}(spec) \models t_1 = t_2$, for $t_1, t_2 \in \mathcal{T}(sig, V^{\mathrm{G}})$. Moreover, admissibility of $spec$ will be proved to ensure that the sig-algebra $\mathcal{T}(sig, V^{\mathrm{G}})/\overset{*}{\longleftrightarrow}_R$ is a data model of $spec$, namely the so-called *standard data model* $\mathcal{M}(spec)$, which is free over V^{G} in $\mathrm{DMod}(spec)$.

4.1 Positive/Negative Conditional Rewrite Specifications

We begin with the idea of extending the distinction made between constructors and the other function symbols in signatures to the axioms in specifications. That is, we require that the set R of rewrite rules (or conditional equations) in a specification with constructors can be partitioned into a set R^C of *constructor rules* and a set R^D of *defining rules*. Intuitively, the constructor rules in R^C are to specify the relations on the constructor ground terms necessary for representing the data items of the given data type, while the defining rules in R^D describe the effects of the other operations of the data type *consistently* (see below).

Note that we have to forbid negative equational condition literals in constructor rules, as is shown in Example 3.12: The class of data models of a specification including a constructor rule with a negative equational condition may be empty, since the class of the data reducts of all models of such a specification need not contain an initial element (see Lemma 3.9). In defining rules, however, we do admit negative equational conditions. To prevent a defining rule $l = r \leftarrow \Delta$ from being applied to a constructor (ground) term, its left-hand side l must contain at least one non-constructor function symbol.

Definition 4.1 Let $sig = (S, F, \alpha)$ be a signature such that $C \subseteq F$ is a set of constructors for sig.

(i) A *constructor rule* is a rewrite rule $l = r \leftarrow u_1 = v_1, \ldots, u_n = v_n$ such that

  (a) $l, r \in \mathcal{T}(sig^C, V)$ and $u_i, v_i \in \mathcal{T}(sig^C, V^{\mathrm{C}})$ for $i = 1, \ldots, n$

  (b) $\mathrm{Var}(r) \subseteq \mathrm{Var}(l)$ and $\mathrm{Var}(u_i), \mathrm{Var}(v_i) \subseteq \mathrm{Var}(l)$ for $i = 1, \ldots, n$

(ii) A *defining rule* is a rewrite rule $l = r \leftarrow \Delta$ satisfying

  (a) $l \in \mathcal{T}(sig, V) \backslash \mathcal{T}(sig^C, V)$

  (b) Δ does not contain any literal of the form $\neg\mathrm{def}(t)$.

A major purpose of the rewrite relation $\longrightarrow_R$ associated with a specification with constructors $spec = (sig, C, R)$ is to give an explicit characterization of the "best" data model $\mathcal{M}(spec)$ as the quotient algebra $\mathcal{T}/\overset{*}{\longleftrightarrow}_R$ of a suitable term algebra $\mathcal{T}$. It is possible to define $\longrightarrow_R$ on $\mathcal{GT}(sig)$ in such a way that admissibility of $spec$ implies initiality of $\mathcal{GT}(sig)/\overset{*}{\longleftrightarrow}_R$ in $\mathrm{DMod}(spec)$ (see [AM95], [WG94a]). In this paper, however, we will define $\longrightarrow_R$ on $\mathcal{T}(sig, V^{\mathrm{G}})$, since for $\mathcal{T} := \mathcal{GT}(sig)$ a respective formulation of Theorem 4.8 does not hold for clauses with general variables.

In accordance with the distinction we make between the constructor rules R^C and the defining rules R^D in R, we define the rewrite relation $\longrightarrow_R$ in two steps. Firstly, rewriting constructor ground (sub-) terms is only possible with the rewrite relation $\longrightarrow_{R^C}$ induced by R^C. Since R^C is a positive conditional rewrite system $\longrightarrow_{R^C}$ can be defined as usual. Secondly, for a rewrite step $t[l\sigma]_p \longrightarrow_R t[r\sigma]_p$ with a defining rule $l = r \leftarrow \Delta$ in R^D, each condition literal in $\Delta\sigma$ must be fulfilled. This means for a negative condition $u \neq v$ in Δ that both $u\sigma$ and $v\sigma$ can be rewritten (using $\longrightarrow_R$) to constructor ground terms which are not joinable using $\longrightarrow_{R^C}$. A definedness atom $\mathrm{def}(u)$ in Δ is fulfilled if $u\sigma \overset{*}{\longrightarrow}_R \hat{u}$ for some constructor ground term $\hat{u}$.

Definition 4.2 Let $spec = (sig, C, R)$ be a specification with constructors and $R = R^C \uplus R^D$ where R^C is a set of constructor rules and R^D a set of defining rules.

(i) Let the sequence $(\longrightarrow_{R^C,i})_{i \in \mathbb{N}}$ of relations on $\mathcal{T}(sig, V^{\mathrm{G}})$ be defined by

  (a) $\longrightarrow_{R^C,0} = \emptyset$.

  (b) $t_1 \longrightarrow_{R^C,i+1} t_2$ if there is a rewrite rule $l = r \leftarrow u_1 = v_1, \ldots, u_n = v_n$ in R^C, a position $p \in \mathrm{Pos}(t_1)$ and an inductive substitution σ such that (1) $t_1/p = l\sigma$ (2) $t_2 = t_1[r\sigma]_p$ and (3) $u_k\sigma \downarrow_{R^C,i} v_k\sigma$ for $k = 1, \ldots, n$.

Then $\longrightarrow_{R^C} = \bigcup_{i \in \mathbb{N}} \longrightarrow_{R^C,i}$.

(ii) Let the sequence $(\longrightarrow_{R,i})_{i\in\mathbb{N}}$ of relations on $\mathcal{T}(sig, V^{\mathrm{G}})$ be defined by

(a) $\longrightarrow_{R,0} = \longrightarrow_{R^C}$.

(b) $t_1 \longrightarrow_{R,i+1} t_2$ if $t_1 \longrightarrow_{R^C} t_2$ or there is a rewrite rule $l = r \leftarrow \Delta$ in R^D, a position $p \in \mathrm{Pos}(t_1)$ and an inductive substitution σ such that (1) $t_1/p = l\sigma$ (2) $t_2 = t_1[r\sigma]_p$ (3) for each $u = v$ in Δ, $u\sigma \downarrow_{R,i} v\sigma$ (4) for each $\mathrm{def}(u)$ in Δ there is a $\hat{u} \in \mathcal{GT}(sig^C)$ such that $u\sigma \xrightarrow{*}_{R,i} \hat{u}$ and (5) for each $u \neq v$ in Δ there are $\hat{u}, \hat{v} \in \mathcal{GT}(sig^C)$ such that $u\sigma \xrightarrow{*}_{R,i} \hat{u}$, $v\sigma \xrightarrow{*}_{R,i} \hat{v}$ and $\hat{u} \not\downarrow_{R^C} \hat{v}$.

Then $\longrightarrow_R = \bigcup_{i\in\mathbb{N}} \longrightarrow_{R,i}$.

Basic properties of $\longrightarrow_{R^C}$ and $\longrightarrow_R$ are listed in the following lemma.

Lemma 4.3

(1) $\longrightarrow_{R^C,i} \subseteq \longrightarrow_{R^C,i+1} \subseteq \longrightarrow_{R^C}$ *for all* $i \in \mathbb{N}$

(2) $\longrightarrow_{R^C} \subseteq \longrightarrow_{R,i} \subseteq \longrightarrow_{R,i+1} \subseteq \longrightarrow_R$ *for all* $i \in \mathbb{N}$

(3) *If* $t \xrightarrow{*}_R t'$ *for* $t \in \mathcal{GT}(sig^C)$ *and* $t' \in \mathcal{T}(sig, V^{\mathrm{G}})$ *then* $t \xrightarrow{*}_{R^C} t'$ *and* $t' \in \mathcal{GT}(sig^C)$.

(4) $\longrightarrow_R$ *is sort-invariant and monotonic.*

Example 3.13 shows that requiring R to consist of constructor rules and defining rules only is not sufficient for the existence of data models. Note that the specification in Example 3.13 contains an *inconsistent* definition for the function symbol d – inconsistent in the sense that there are constructor ground terms t_1 and t_2 such that $t_1 \xleftrightarrow{*}_R t_2$ but not $t_1 \xleftrightarrow{*}_{R^C} t_2$. Such inconsistencies cannot arise if $\longrightarrow_R$ is confluent: Then $t_1 \xleftrightarrow{*}_R t_2$ entails $t_1 \downarrow_R t_2$ for $t_1, t_2 \in \mathcal{GT}(sig^C)$, and by applying Lemma 4.3(3) one obtains $t_1 \downarrow_{R^C} t_2$ and hence $t_1 \xleftrightarrow{*}_{R^C} t_2$. To put it another way, confluence of $\longrightarrow_R$ guarantees that $spec = (sig, C, R)$ is a *consistent extension* of the "base" or constructor specification $spec^C = (sig^C, R^C)$ (see [EM85]).

A further admissibility condition is needed to ensure that $\mathcal{T}(sig, V^{\mathrm{G}})/\xleftrightarrow{*}_R$ is a (data) model of *spec* (see Example 3.11). In order to achieve correspondence of the model semantics with our method of testing negative condition literals in defining rules, there needs to be a definedness condition literal for each (non-constructor) term occurring in a negative condition literal.

Definition 4.4 A specification with constructors $spec = (sig, C, R)$ is called an *admissible* specification if *spec* satisfies the following conditions:

(a) $R = R^C \uplus R^D$ where R^C is a set of constructor rules and R^D a set of defining rules.

(b) $\longrightarrow_R$ is confluent.

(c) For each $l = r \leftarrow \Delta$ in R^D and for each $t \in \mathcal{T}(sig, V) \backslash \mathcal{T}(sig^C, V^{\mathrm{C}})$ occurring (on top-level) in a negative literal in Δ there is a literal $\mathrm{def}(t)$ in Δ.

Note that we do *not* require termination of $\longrightarrow_R$ in our admissibility conditions.

4.2 The Standard Data Model $\mathcal{M}(spec)$

Having defined our admissibility conditions we can now show that, for any admissible specification $spec = (sig, C, R)$, the sig-algebra $\mathcal{T}(sig, V^{\mathrm{G}})/\overset{*}{\longleftrightarrow}_R$ is a data model of $spec$ that plays a particular role in the class of all data models of $spec$: It is free over V^{G} in DMod($spec$). Thus, $\mathcal{M}(spec) := \mathcal{T}(sig, V^{\mathrm{G}})/\overset{*}{\longleftrightarrow}_R$ has algebraic properties similar to those of the initial model of a usual equational specification in that $\mathcal{M}(spec)$ may be regarded as a distinguished "representative" of the class of all data models (see Theorem 4.8). This gives rise to two appropriate kinds of semantics for specifications with constructors of data types with partial operations, namely (i) DMod($spec$) and (ii) (the isomorphism class of) the so-called *standard data model* $\mathcal{M}(spec)$.

Proposition 4.5 *Let $spec = (sig, C, R)$ be an admissible specification. Then*

(1) *$\mathcal{T}(sig, V^{\mathrm{G}})/\overset{*}{\longleftrightarrow}_R$ is a data model of spec.*

(2) *Let $\mathcal{A}$ be any data model of spec. Then its data reduct $\mathcal{A}^C$ and $\mathcal{GT}(sig^C)/\overset{*}{\longleftrightarrow}_{R^C}$ are (sig^C-) isomorphic.*

Hence, the data reduct of *every* data model of an admissible specification can be explicitly characterized in terms of $\mathcal{GT}(sig^C)$ and $\longrightarrow_{R^C}$.

The following lemma justifies the view that $\longrightarrow_R$ yields a sound and complete operationalization of equality in all data models of $spec$. It is of central importance in the proofs of Theorems 4.7 and 4.8.

Lemma 4.6 *Let $spec = (sig, C, R)$ be an admissible specification, and let $t_1, t_2 \in \mathcal{T}(sig, V^{\mathrm{G}})$. Then $t_1 \overset{*}{\longleftrightarrow}_R t_2$ iff DMod(spec) $\models t_1 = t_2$.*

Freeness of an algebra is usually defined as follows. Let K be a class of sig-algebras, and let $X \subseteq V$ be an S-sorted family of variables. A sig-algebra $\mathcal{F}(X)$ is *free* over X in K if $\mathcal{F}(X) \in K$ and there is a valuation u of X in $\mathcal{F}(X)$ such that for every valuation φ of X in a sig-algebra $\mathcal{A} \in K$ there is a unique sig-homomorphism h: $\mathcal{F}(X) \to \mathcal{A}$ such that the following diagram commutes, i.e. $\varphi = h \circ u$.

$$\begin{tikzcd}
X \arrow[r, "\varphi"] \arrow[d, "u"'] \arrow[dr, phantom, "=" near start] & \mathcal{A} \\
\mathcal{F}(X) \arrow[ur, "h"'] &
\end{tikzcd}$$

Note that the free algebra is unique up to isomorphism: If sig-algebras $\mathcal{F}_1(X)$ and $\mathcal{F}_2(X)$ are free over X in K then $\mathcal{F}_1(X)$ and $\mathcal{F}_2(X)$ are isomorphic (see [EM85]).

Theorem 4.7 *Let $spec = (sig, C, R)$ be an admissible specification. Then $\mathcal{T}(sig, V^{\mathrm{G}})/\overset{*}{\longleftrightarrow}_R$ is free over V^{G} in DMod(spec).*

We call $\mathcal{T}(sig, V^{\mathrm{G}})/\overset{*}{\longleftrightarrow}_R$ the *standard data model* of an admissible specification *spec* and use $\mathcal{M}(spec)$ to denote it. Its significance as a "representative" of DMod($spec$) is confirmed in the following characterization of the relation between validity in $\mathcal{M}(spec)$ and validity in all data models of *spec*.

Theorem 4.8 *Let spec* $= (sig, C, R)$ *be an admissible specification, and let* Γ *be a clause such that* $\mathcal{M}(spec) \models \mathrm{def}(t)$ *for every (top-level) term* t *occurring in a negative literal of* Γ*. Then* $\mathcal{M}(spec) \models \Gamma$ *is sufficient for DMod(spec)* $\models \Gamma$*.*

In particular, every equation valid in $\mathcal{M}(spec)$ is valid in every data model of *spec*.

It should be noted that Theorems 4.7, 4.8 and 6.2 have counterparts in the more general specification approach of [WG94a] and [WG94b] from which the specification language presented in this paper originates (see Sect. 1).

5 A Confluence Criterion

Contrary to most other specification formalisms for ITP, the one proposed in this paper does not require termination. However, its admissibility conditions require confluence, and since many interesting rewrite systems (i.e. sets of rewrite rules) are non-decreasing (see [DOS88] for a definition of "decreasing") or even non-terminating (see Example 1.2), we need a confluence criterion that does not presuppose termination.

Note that several basic results on confluence of unconditional rewrite systems that are based on syntactic considerations do not hold in the conditional case. In particular, local confluence of conditional rewrite systems is not equivalent to joinability of all critical pairs. In other words, variable overlaps may be "critical" as well. This may even happen when termination is given (combined with left-linearity and normality; see [DOS88], Example B, p. 36). If we do not require termination, the situation is even more complicated: There are left-linear positive conditional rewrite systems that do not have any critical pairs but lack confluence (see [DOS88], Example A, p. 36; taken from [BK86]). Therefore, reasonable syntactic confluence criteria for non-terminating rewrite systems need strengthened forms of joinability of critical pairs and syntactic restrictions on rewrite rules such as left-linearity and (weakened forms of) normality.

Another major problem is caused by the infinite number of substitutions that must be tested for fulfilling the conditions in critical pairs. Therefore, effectively testable conditions which guarantee the *infeasibility* of the critical pairs have practical relevance. In this paper, we introduce a confluence criterion (see Theorem 5.2) which essentially makes use of the fact that critical pairs with complementary literals in the conditions are infeasible and need not be considered hence. It is not the strongest known confluence criterion applicable to non-terminating constructor-based rewrite systems[2] but it is effectively testable. Moreover, it extends the class of specifications admitted for ITP since it no longer requires termination. To our knowledge it is the strongest confluence

[2]See Theorems 68 and 71 of [Wir95]. Theorem 68 is the version for ω-shallow confluence, Theorem 71 the one for ω-level confluence.

criterion without a termination precondition that can be effectively used in practice. Furthermore, it also applies to terminating systems, which may be attractive if one does not know how to (effectively) show termination or if the correctness of the technique for proving termination requires confluence.

As our rewrite systems consist of constructor rules and defining rules the problem of establishing confluence of the whole rewrite system can be decomposed into three smaller sub-problems: Firstly, we show confluence of $\longrightarrow_{R^C}$, then commutation of the constructor rules with the defining rules, and finally, using these assumptions, confluence of $\longrightarrow_R$. Thus, different criteria may be applied to handle these sub-problems. For example, unless it is trivial, proving confluence of $\longrightarrow_{R^C}$ may often call for sophisticated semantic considerations or confluence criteria that apply to terminating rewrite systems only. For $\longrightarrow_R$, however, neither semantic confluence criteria nor confluence criteria with termination preconditions are practically feasible in general. One reason for this may be that effective applications of semantic confluence criteria require the specification given by the whole rewrite system to have been modeled before in some formalism. Another reason is that termination of the whole rewrite system may not be given or difficult to show without any confluence assumptions.

Since the confluence criterion we present in this section presupposes confluence of $\longrightarrow_{R^C}$, we first discuss how to establish that $\longrightarrow_{R^C}$ is confluent. Without constructor rules, i.e. $R^C = \emptyset$, confluence of $\longrightarrow_{R^C}$ is trivial. While this seems rather restrictive, this case of *free constructors* is very important in practice since a lot of data structures are freely generated. Besides, non-free constructors pose serious problems in most frameworks for ITP – if they can be handled at all. Another way to prove confluence of $\longrightarrow_{R^C}$ is to use one of the known confluence criteria for positive conditional rewrite systems (see [DOS88]). Note that the application of most of these confluence criteria requires termination of $\longrightarrow_{R^C}$. Termination of the constructor rules, however, does not mean termination of the whole rewrite system. More syntactic criteria for confluence of $\longrightarrow_{R^C}$ can be found in Sect. 15 of [Wir95]. Sometimes, however, confluence of $\longrightarrow_{R^C}$ can only be shown by using the semantic knowledge of the specifier: either with semantic confluence criteria in the style of [Pla85] (see Theorem 6.5 of [WG94b]) or in a way that only works for the concrete set of constructor rules at hand.

Before formally presenting our syntactic confluence criterion we have to introduce more notions concerning (conditional) critical pairs and properties of rewrite systems.

Let $t_1, t_2 \in \mathcal{T}(sig, V)$. We call a constructor substitution σ a *unifier* of $t_1 = t_2$ if $t_1\sigma = t_2\sigma$. A unifier σ of $t_1 = t_2$ is said to be *most general* on a finite set $X \subseteq V$ if for every unifier μ of $t_1 = t_2$ there is a constructor substitution τ such that $(\sigma\tau)|_X = \mu|_X$. Note that if $t_1 = t_2$ has a unifier, then it also has a most general unifier on X, which we denote by $\mathrm{mgu}(t_1 = t_2,\ X)$.

Definition 5.1 Let $spec = (sig, C, R)$ be a specification with constructors and $R = R^C \uplus R^D$ where R^C is a set of constructor rules and R^D a set of defining rules.

(i) Let $l_i = r_i \leftarrow \Delta_i$ be a rewrite rule in R with $X_i := \mathrm{Var}(l_i, r_i, \Delta_i)$ for $i \in \{0,1\}$. Assume w.l.o.g. $X_0 \cap X_1 = \emptyset$. If there is a non-variable position $p \in \mathrm{Pos}(l_1)$ such that $\sigma = \mathrm{mgu}(l_0 = l_1/p,\ X_0 \cup X_1)$ exists and $l_1[r_0]_p\sigma \neq r_1\sigma$, then
$$\Big(\ (l_1[r_0]_p\sigma,\ \Delta_0\sigma,\ a_0),\ (r_1\sigma,\ \Delta_1\sigma,\ a_1)\ \Big)$$
is a (non-trivial) *critical pair of the form* (a_0, a_1) where, for $i \in \{0,1\}$, $a_i = 0$ if $l_i = r_i \leftarrow \Delta_i$ is in R^C and $a_i = 1$ if $l_i = r_i \leftarrow \Delta_i$ is in R^D.
The set of all critical pairs between rewrite rules in R is denoted by $\mathrm{CP}(R)$.

(ii) The above critical pair is *complementary* if

(a) there are $u, v \in \mathcal{T}(sig, V)$ and an $i \in \{0,1\}$ such that $u = v$ or $v = u$ occurs in $\Delta_i\sigma$ and $u \neq v$ occurs in $\Delta_{1-i}\sigma$; or

(b) there are $t, \hat{u}, \hat{v} \in \mathcal{T}(sig, V)$ such that $\hat{u}$ and $\hat{v}$ are distinct $\longrightarrow_R$-irreducible ground terms, $t = \hat{u}$ or $\hat{u} = t$ occurs in $\Delta_0\sigma$ and $t = \hat{v}$ or $\hat{v} = t$ occurs in $\Delta_1\sigma$.

Consider e.g. the critical pairs of the rewrite system in Example 1.1: There are only two critical pairs resulting from overlapping the two div-rules into each other. Since the two critical pairs are symmetric and the notion of complementarity is symmetric too, we just have to consider one of the critical pairs, say
$$\Big(\ (\mathsf{0}, y \neq \mathsf{0} \wedge \mathsf{less}(x,y) = \mathsf{true}, 1), (\mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),y)), y \neq \mathsf{0} \wedge \mathsf{less}(x,y) = \mathsf{false}, 1)\ \Big).$$
It is complementary according to Part (b) of Definition 5.1 (instantiate t with $\mathsf{less}(x,y)$, $\hat{u}$ with true and $\hat{v}$ with false). Similarly, for the complementarity of the critical pairs of the rewrite system of Example 1.2 we only have to check that the critical pair $\Big(\ (z_1, x = z_2, 1), (\mathsf{div1}(x, y, \mathsf{s}(z_1), \mathsf{plus}(z_2, y)), x \neq z_2, 1)\ \Big)$ is complementary, which is obviously the case (see Part (a) of Definition 5.1).

In addition, two further notions are presupposed in our confluence criterion, namely left-linearity and a weakened form of normality. These properties of rewrite systems are necessary for establishing confluence of non-terminating rewrite systems (see above).

A rewrite system R is called *left-linear* if the left-hand side l of each rewrite rule $l = r \leftarrow \Delta$ in R is linear. Moreover, we call R *weakly normal* if each $l = r \leftarrow \Delta$ in R satisfies the following condition: For each $t_1 = t_2$ in Δ there is an $i \in \{1,2\}$ such that t_i is a $\longrightarrow_R$-irreducible ground term or $t_i \in \mathcal{T}(sig^C, V^C)$ or $\mathrm{def}(t_i)$ occurs in Δ.

Obviously, the rewrite systems in Examples 1.1 and 1.2 are left-linear. Assuming that all variables are constructor variables these rewrite systems are also weakly normal, because the right-hand sides of all equational conditions are constructor terms. Thus, we can directly apply the following syntactic confluence criterion to prove confluence of the rewrite relations in these examples.

Theorem 5.2 *Let spec* $= (sig, C, R)$ *be a specification with constructors such that* R *is left-linear as well as weakly normal and* $R = R^C \uplus R^D$ *where* R^C *is a set of constructor rules and* R^D *a set of defining rules.*
Assume that $\longrightarrow_{R^C}$ *is confluent. If each critical pair in* $\mathrm{CP}(R)$ *of the form* $(0,1)$, $(1,0)$ *or* $(1,1)$ *is complementary, then* $\longrightarrow_R$ *is confluent.*

Note that this confluence criterion is stronger than a similar theorem of [BK86] (also cited in [DOS88]) since instead of normality it only requires weak normality, a property that is less restrictive because we can always achieve it by adding definedness atoms to the condition literals. Moreover, instead of requiring orthogonality, our theorem can deal with critical pairs provided they are complementary – a property which could again be weakened, but (to our knowledge) not in an effective manner that would satisfy the practical requirements of an admissibility condition for a specification language.

6 Discussion

We conclude this paper by providing some evidence for the usefulness of the proposed specification language as part of a rewrite-based logical framework for ITP.

Concerning the *syntax* of the language it should be noted that an additional kind of variables, namely constructor variables, is offered. Since our constructor variables restrict the validity of statements to the finitely generated data items, they are mostly preferred to general variables in the context of ITP. We also offer negative conditions in conditional equations. Therefore, tedious axiomatizations of equality predicates for various sorts in terms of bool-valued eq-operations can be avoided (consider the natural definitions of div and $\mathsf{div1}$ in Sect. 1).

The appropriateness of the *semantics* of our specification language can also be demonstrated with Examples 1.1 and 1.2. Consider the class of data models of the specification $spec'$ in Example 1.2. Since $spec'$ is an admissible specification (see Theorem 5.2), the standard data model $\mathcal{M}(spec')$ exists so that $\mathrm{DMod}(spec')$ is not empty. It is obvious that every data model of $spec'$ has the two truth values and the natural numbers as its data reduct. Hence, in spite of the existence of "junk terms", clauses such as $\mathsf{plus}(x,y)=\mathsf{plus}(y,x)$ or clause (1) (see Example 1.2) are valid in $\mathrm{DMod}(spec')$ assuming that all variables are constructor variables. Furthermore, the "invariant" for a proof of clause (1), namely

$$y = \mathsf{0} \;\vee\; \mathsf{less}(x, \mathsf{times}(y,z)) = \mathsf{true} \;\vee\; \mathsf{div1}(x,y,\mathsf{0},\mathsf{0}) = \mathsf{div1}(x,y,z,\mathsf{times}(y,z))$$

is also valid in $\mathrm{DMod}(spec')$ (as can be proved by induction on z).

As inductive proofs often call for extensions of the specification in order to facilitate the formulation of missing lemmas or stronger induction hypotheses, an essential requirement for an inductive semantics is its monotonicity of validity wrt. "consistent" extension of the specification. For $\mathrm{DMod}(spec)$, this monotonicity property is given as is shown in the following.

Definition 6.1 For $i \in \{0,1\}$ let $spec_i = (sig_i, C_i, R_i)$ be a specification with constructors where $sig_i = (S_i, F_i, \alpha_i)$, and let $V_i = (V_{i,s})_{s \in S_i}$ be a variable system for sig_i. We say that $spec_1$ is a *constructor-consistent extension* of $spec_0$ if the following conditions are met: (1) $S_0 \subseteq S_1$, $F_0 \subseteq F_1$, $C_0 \subseteq C_1$ and $\alpha_0 \subseteq \alpha_1$ (2) $V_{0,s} = V_{1,s}$ for each $s \in S_0$ (3) $R_0 \subseteq R_1$ (4) for each $c \in C_1 \backslash C_0$ with $\alpha_1(c) = s_1 \ldots s_n s$ we have $s \notin S_0$ and (5) for each constructor rule $l = r \leftarrow \Delta$ in $R_1 \backslash R_0$ we have $l \notin \mathcal{T}(sig_0^{C_0}, V_0)$.

Theorem 6.2 *For $i \in \{0,1\}$ let $spec_i = (sig_i, C_i, R_i)$ be an admissible specification. Assume that $spec_1$ is a constructor-consistent extension of $spec_0$, and let Γ be a clause (over sig_0 and V_0). Now if $DMod(spec_0) \models \Gamma$, then $DMod(spec_1) \models \Gamma$.*

Theorems 4.8 and 6.2 imply a similar monotonicity for the validity in $\mathcal{M}(spec)$, which, however, requires the definedness of the terms in the negative literals of the clauses. This requirement is in fact needed: The clause $\mathsf{minus}(\mathsf{0}, \mathsf{s}(\mathsf{0})) \neq \mathsf{0}$ is valid in $\mathcal{M}(spec)$ (but not in DMod($spec$)) where $spec$ is the specification of Example 1.1. However, after a constructor-consistent extension of $spec$ with the rewrite rule $\mathsf{minus}(\mathsf{0}, \mathsf{s}(y)) = \mathsf{0}$ the clause $\mathsf{minus}(\mathsf{0}, \mathsf{s}(\mathsf{0})) \neq \mathsf{0}$ is no longer valid in the resulting standard data model. Mainly because of Theorem 6.2 we prefer DMod($spec$) as the inductive semantics of our specification language.

Finally, we would like to emphasize the (logical) weakness and suitability of our *admissibility conditions*. Contrary to many (first-order) specification formalisms for ITP, we can model partial operations that may result from incomplete defining case distinctions or from non-termination. Note that axiomatizing an arbitrary completion (i.e. a total extension) of a partially defined function is not adequate because it may result in unintended inductive theorems and unnecessary implementation requirements. Moreover, defining the semantics of a partially specified operation as that of all its consistent completions is not reasonable either: A consistent completion is often not possible (consider e.g. $\mathsf{a} = \mathsf{s}(\mathsf{a})$ for a non-constructor constant a) making this approach meaningless for non-terminating rewrite systems.

Furthermore, even if the specifying rewrite system terminates, it may be difficult to show this (see our discussion of Example 1.1). Since confluence as the essential admissibility condition of our specification language can often be effectively proved by means of our new confluence criterion that does not presuppose termination, we do not have the typical termination related problems as in the rewrite-based approaches of [BR95] and [KS96]: For example, we can easily axiomatize operations that are naturally defined by *destructor recursion* (such as div or merge-sort).

Note that the reduction order given by a terminating rewrite relation is usually not necessary for rewrite-based ITP as is shown in [WK95], where we propose the use of an independent *induction order* for justifying the applications of induction hypotheses: For example, we can easily prove the above “invariant” by structural induction on z.

A The Proofs

Proof of Lemma 3.3. $\mathcal{GT}(sig)$ is initial in the class of all sig-algebras, and so there is only one sig-homomorphism from $\mathcal{GT}(sig)$ to $\mathcal{B}$, namely $\mathrm{eval}^{\mathcal{B}}$. Hence, we have $h_s(t^{\mathcal{A}}) = t^{\mathcal{B}}$ for all $t \in \mathcal{GT}(sig)_s$ and for all $s \in S$, which implies that $h_s(a) \in B_s^C$ for all $a \in A_s^C$. Thus, h_s^C is a function with $h_s^C\colon A_s^C \to B_s^C$. With the assumption that h is a sig-homomorphism, it is easy to show that h^C is a sig^C-homomorphism. As $\mathcal{GT}(sig^C)$ is initial in the class of all sig^C-algebras, we have $\mathrm{eval}^{\mathcal{B}^C} = h^C \circ \mathrm{eval}^{\mathcal{A}^C}$. Since $\mathrm{eval}^{\mathcal{B}^C}$ is a sig^C-epimorphism, h^C must be a sig^C-epimorphism as well. □

Proof of Lemma 3.5. Since σ is a constructor substitution, $(\mathrm{eval}_\varphi^{\mathcal{A}} \circ \sigma)(x) \in A_s^C$ for all $x \in X \cap V^{\mathrm{C}}$, so $\mathrm{eval}_\varphi^{\mathcal{A}} \circ \sigma$ is indeed a valuation of X in $\mathcal{A}$. A simple structural induction on t shows that $\mathrm{eval}_\varphi^{\mathcal{A}}(t\sigma) = \mathrm{eval}^{\mathcal{A}}_{(\mathrm{eval}_\varphi^{\mathcal{A}} \circ \sigma)}(t)$ for all $t \in \mathcal{T}(sig, V)$. □

Proof of Proposition 3.7. Let $\mathcal{A} \in \mathrm{Mod}(spec)$ and φ be a valuation of V^{C} in $\mathcal{A}$.

(1)⇒(2): Let σ be an inductive substitution. Since $\mathcal{A} \in \mathrm{Mod}(spec)$, $\mathcal{A}$ satisfies Γ with $\mathrm{eval}_\varphi^{\mathcal{A}} \circ \sigma$. Thus, there is a literal λ in Γ such that $\mathcal{A}$ satisfies λ with $\mathrm{eval}_\varphi^{\mathcal{A}} \circ \sigma$. Now Lemma 3.5 shows that $\mathcal{A}$ satisfies $\lambda\sigma$ with φ. Since $\lambda\sigma$ is a ground literal we obtain $\mathcal{A} \models \lambda\sigma$.

(2)⇒(1): Since $\mathcal{A}^C$ is term-generated, there must be an inductive substitution σ such that $\varphi = \mathrm{eval}^{\mathcal{A}^C} \circ \sigma = \mathrm{eval}_\varphi^{\mathcal{A}} \circ \sigma$. Using Lemma 3.5 and that $t\sigma$ is a ground term if $t \in \mathcal{T}(sig, V^{\mathrm{C}})$, we obtain $\mathrm{eval}_\varphi^{\mathcal{A}}(t) = \mathrm{eval}^{\mathcal{A}}_{(\mathrm{eval}_\varphi^{\mathcal{A}} \circ \sigma)}(t) = \mathrm{eval}_\varphi^{\mathcal{A}}(t\sigma) = t\sigma^{\mathcal{A}}$ for all $t \in \mathcal{T}(sig, V^{\mathrm{C}})$. Because of (2), $\mathcal{A} \models \lambda\sigma$ holds for some literal λ in Γ, and since $\mathrm{eval}_\varphi^{\mathcal{A}}(t) = t\sigma^{\mathcal{A}}$ it is clear that $\mathcal{A}$ satisfies λ with φ. Hence, $\mathcal{A}$ satisfies Γ with φ, which proves that $\mathcal{A} \models \Gamma$. □

Proof of Lemma 3.9. Let $\mathcal{A}$ be a data model, and let $\mathcal{B} \in \mathrm{Mod}(spec)$. Define a family $h = (h_s)_{s \in S}$ of functions $h_s\colon A_s^C \to B_s^C$ by $h_s(a) = t^{\mathcal{B}}$ where $a = t^{\mathcal{A}}$ and $t \in \mathcal{GT}(sig^C)_s$. We show that h_s is well-defined. Let $t_1, t_2 \in \mathcal{GT}(sig^C)_s$ such that $t_1^{\mathcal{A}} = a = t_2^{\mathcal{A}}$. Since $\mathcal{A}$ is a data model, $\mathrm{Mod}(spec) \models t_1 = t_2$ and hence $t_1^{\mathcal{B}} = t_2^{\mathcal{B}}$. Thus we have $h_s(t_1^{\mathcal{A}}) = t_1^{\mathcal{B}} = t_2^{\mathcal{B}} = h_s(t_2^{\mathcal{A}})$. A simple argument proves that $h\colon \mathcal{A}^C \to \mathcal{B}^C$ is a sig^C-homomorphism. Now let $h'\colon \mathcal{A}^C \to \mathcal{B}^C$ be another sig^C-homomorphism from $\mathcal{A}^C$ to $\mathcal{B}^C$. An induction over $\mathcal{GT}(sig^C)$ shows that $h(t^{\mathcal{A}}) = h'(t^{\mathcal{A}})$ for all $t \in \mathcal{GT}(sig^C)$, which implies that $h(a) = h'(a)$ for all $a \in A^C$. Hence, there is exactly one sig^C-homomorphism from $\mathcal{A}^C$ to $\mathcal{B}^C$, and so $\mathcal{A}^C$ is initial in $\{\, \mathcal{B}^C \mid \mathcal{B} \in \mathrm{Mod}(spec) \,\}$.

Conversely, let $\mathcal{A}^C$ be initial in $\{\, \mathcal{B}^C \mid \mathcal{B} \in \mathrm{Mod}(spec) \,\}$. Let $\mathcal{B} \in \mathrm{Mod}(spec)$ and $h\colon \mathcal{A}^C \to \mathcal{B}^C$ be the unique sig^C-homomorphism from $\mathcal{A}^C$ to $\mathcal{B}^C$. If $t_1^{\mathcal{A}} = t_2^{\mathcal{A}}$ for any $t_1, t_2 \in \mathcal{GT}(sig^C)$ then $t_1^{\mathcal{B}} = h(t_1^{\mathcal{A}}) = h(t_2^{\mathcal{A}}) = t_2^{\mathcal{B}}$ due to the definition of h. As $t_1 = t_2$ is a ground equation, $t_1 = t_2$ is valid in $\mathcal{B}$. Thus, $\mathrm{Mod}(spec) \models t_1 = t_2$, and so $\mathcal{A}$ is a data model. □

Proof of Corollary 3.10. As both $\mathcal{A}^C$ and $\mathcal{B}^C$ are initial in $\{\, \mathcal{B}^C \mid \mathcal{B} \in \mathrm{Mod}(spec) \,\}$, they must be ($sig^C$-) isomorphic (see e.g. [EM85]). □

Proof sketch of Lemma 4.3. (1) and (2) can be shown by induction on i, and (3) and (4) follow immediately from Definitions 4.1 and 4.2. $\square$

In the following proofs, let $[t]$ denote the *sig*-congruence class of $t \in \mathcal{T}(sig, V^{\mathrm{G}})$ with respect to $\stackrel{*}{\longleftrightarrow}_R$, i.e. $[t] = \{\, t' \in \mathcal{T}(sig, V^{\mathrm{G}}) \mid t' \stackrel{*}{\longleftrightarrow}_R t \,\}$.

Proof of Proposition 4.5. (1) Let $\mathcal{M} = \mathcal{T}(sig, V^{\mathrm{G}})/\stackrel{*}{\longleftrightarrow}_R$. By Lemma 4.3(4), $\longrightarrow_R$ is sort-invariant and monotonic. Thus, $\stackrel{*}{\longleftrightarrow}_R$ is a *sig*-congruence on $\mathcal{T}(sig, V^{\mathrm{G}})$ so that $\mathcal{M} = (M, F^{\mathcal{M}})$ is actually a *sig*-algebra satisfying

- $M_s = \{\, [t] \mid t \in \mathcal{T}(sig, V^{\mathrm{G}})_s \,\}$ and $M^C_s = \{\, [t] \mid t \in \mathcal{GT}(sig^C)_s \,\}$ for all $s \in S$,
- $f^{\mathcal{M}}([t_1], \ldots, [t_n]) = [f(t_1, \ldots, t_n)]$ for all $f \in F$ and for all $t_i \in \mathcal{T}(sig, V^{\mathrm{G}})_{s_i}$ where $\alpha(f) = s_1 \ldots s_n s$ and $i = 1, \ldots, n$.

First we prove that $\mathcal{M}$ is a *sig*-model of *spec*. Let $l = r \leftarrow \Delta$ be a conditional rewrite rule in R and φ be a valuation of V in $\mathcal{M}$ such that $\mathcal{M}$ satisfies each literal in Δ with φ. We have to show that $\mathrm{eval}^{\mathcal{M}}_{\varphi}(l) = \mathrm{eval}^{\mathcal{M}}_{\varphi}(r)$. Let u be the valuation of V^{G} in $\mathcal{M}$ with $u(x) = [x]$ for all $x \in V^{\mathrm{G}}$. By the Axiom of Choice, there is an inductive substitution σ for φ such that $\varphi(x) = (\mathrm{eval}^{\mathcal{M}}_u \circ \sigma)(x) = [x\sigma]$ for all $x \in V$. Then, by Lemma 3.5, $\mathrm{eval}^{\mathcal{M}}_{\varphi}(t) = \mathrm{eval}^{\mathcal{M}}_{(\mathrm{eval}^{\mathcal{M}}_u \circ \sigma)}(t) = [t\sigma]$ for all $t \in \mathcal{T}(sig, V)$. We claim that $l\sigma \longrightarrow_R r\sigma$ with $l = r \leftarrow \Delta$: For $u = v$ in Δ we have $\mathrm{eval}^{\mathcal{M}}_{\varphi}(u) = \mathrm{eval}^{\mathcal{M}}_{\varphi}(v)$ and hence $[u\sigma] = [v\sigma]$. Since $\longrightarrow_R$ is confluent, it follows that $u\sigma \downarrow_R v\sigma$. For $\mathrm{def}(u)$ in Δ we have $\mathrm{eval}^{\mathcal{M}}_{\varphi}(u) \in M^C$. Thus, there is a $t \in \mathcal{GT}(sig^C)$ with $[u\sigma] = [t]$, and hence $u\sigma \downarrow_R t$. Because of $t \in \mathcal{GT}(sig^C)$, confluence of $\longrightarrow_R$ and Lemma 4.3(3), there must be a $\hat{u} \in \mathcal{GT}(sig^C)$ with $u\sigma \stackrel{*}{\longrightarrow}_R \hat{u}$. For $u \neq v$ in Δ we have $\mathrm{eval}^{\mathcal{M}}_{\varphi}(u) \neq \mathrm{eval}^{\mathcal{M}}_{\varphi}(v)$. Condition (c) in Definition 4.4 implies that $\mathrm{eval}^{\mathcal{M}}_{\varphi}(u), \mathrm{eval}^{\mathcal{M}}_{\varphi}(v) \in M^C$. Thus, there are $\hat{u}, \hat{v} \in \mathcal{GT}(sig^C)$ such that $u\sigma \stackrel{*}{\longrightarrow}_R \hat{u}$ and $u\sigma \stackrel{*}{\longrightarrow}_R \hat{v}$. Since $[\hat{u}] = \mathrm{eval}^{\mathcal{M}}_{\varphi}(u) \neq \mathrm{eval}^{\mathcal{M}}_{\varphi}(v) = [\hat{v}]$, it follows that $\hat{u} \not\downarrow_R \hat{v}$ and hence $\hat{u} \not\downarrow_{R^C} \hat{v}$. Consequently, $l = r \leftarrow \Delta$ is applicable to $l\sigma$, and we obtain $[l\sigma] = [r\sigma]$. Hence, $\mathrm{eval}^{\mathcal{M}}_{\varphi}(l) = \mathrm{eval}^{\mathcal{M}}_{\varphi}(r)$.

That $\mathcal{M}$ is also a data model of *spec* can be seen as follows. A simple induction on i shows that $t_1 \longrightarrow_{R^C, i} t_2$ entails $\mathrm{Mod}(spec) \models t_1 = t_2$ for all $t_1, t_2 \in \mathcal{GT}(sig^C)$. Thus, $\mathrm{Mod}(spec) \models t_1 = t_2$ obviously holds if $t_1 \downarrow_{R^C} t_2$, which is implied by $t_1^{\mathcal{M}} = t_2^{\mathcal{M}}$ as $\longrightarrow_R$ is confluent. Hence, $\mathcal{M} \in \mathrm{DMod}(spec)$.

(2) For $s \in S$ and for $t \in \mathcal{GT}(sig^C)_s$ define h_s: $(\mathcal{GT}(sig^C)/\stackrel{*}{\longleftrightarrow}_{R^C})_s \to M^C_s$ by $h_s([t]_C) = [t]$ where $[t]_C = \{\, t' \in \mathcal{GT}(sig^C)_s \mid t' \stackrel{*}{\longleftrightarrow}_{R^C} t \,\}$. Then h_s is obviously well-defined and surjective, and by making use of the confluence of $\longrightarrow_R$, one easily proves that h_s is injective. Moreover, it is trivial to show that h is a sig^C-homomorphism. Therefore, $\mathcal{M}^C$ and $\mathcal{GT}(sig^C)/\stackrel{*}{\longleftrightarrow}_{R^C}$ are sig^C-isomorphic. Hence, (2) follows from (1) and Corollary 3.10. $\square$

Proof of Lemma 4.6. Let $\mathcal{A} \in \mathrm{DMod}(spec)$ and φ be a valuation of V^{G} in $\mathcal{A}$. Given that $t_1 \stackrel{*}{\longleftrightarrow}_R t_2$ we have to show that $\mathrm{eval}^{\mathcal{A}}_{\varphi}(t_1) = \mathrm{eval}^{\mathcal{A}}_{\varphi}(t_2)$. First we claim that for all $i \in \mathbb{N}$ and $s_1, s_2 \in \mathcal{T}(sig, V^{\mathrm{G}})$ the following statements hold:

(i) If $s_1 \longrightarrow_{R^C, i} s_2$ then $\mathrm{eval}^{\mathcal{A}}_{\varphi}(s_1) = \mathrm{eval}^{\mathcal{A}}_{\varphi}(s_2)$.

(ii) If $s_1 \longrightarrow_{R, i} s_2$ then $\mathrm{eval}^{\mathcal{A}}_{\varphi}(s_1) = \mathrm{eval}^{\mathcal{A}}_{\varphi}(s_2)$.

We omit the simple proof of (i) and show (ii) by induction on i. If $i = 0$ or if $s_1 \longrightarrow_{R,i+1} s_2$ holds because of $s_1 \longrightarrow_{R^C} s_2$ then (ii) follows from (i). Otherwise we have $s_1 \longrightarrow_{R,i+1} s_2$ since there is a $p \in \mathrm{Pos}(s_1)$, an inductive substitution σ and a defining rule $l = r \leftarrow \Delta$ such that $s_1/p = l\sigma, s_2 = s_1[r\sigma]_p$ and each literal in $\Delta\sigma$ is "fulfilled" by $\longrightarrow_{R,i}$ (see Definition 4.2). We show that $\mathcal{A}$ satisfies each literal in $\Delta\sigma$ with φ which implies $\mathrm{eval}_\varphi^{\mathcal{A}}(l\sigma) = \mathrm{eval}_\varphi^{\mathcal{A}}(r\sigma)$ as $\mathcal{A} \in \mathrm{Mod}(\mathit{spec})$, and hence $\mathrm{eval}_\varphi^{\mathcal{A}}(s_1) = \mathrm{eval}_\varphi^{\mathcal{A}}(s_2)$. For $u = v$ in $\Delta\sigma$ we have $u \downarrow_{R,i} v$, and $\mathrm{eval}_\varphi^{\mathcal{A}}(u) = \mathrm{eval}_\varphi^{\mathcal{A}}(v)$ follows from the induction hypothesis. For $\mathrm{def}(u)$ in $\Delta\sigma$ there is a $\hat{u} \in \mathcal{GT}(sig^C)$ such that $u \xrightarrow{*}_{R,i} \hat{u}$, and the induction hypothesis implies that $\mathrm{eval}_\varphi^{\mathcal{A}}(u) = \mathrm{eval}_\varphi^{\mathcal{A}}(\hat{u}) = \hat{u}^{\mathcal{A}}$, and hence $\mathrm{eval}_\varphi^{\mathcal{A}}(u) \in A^C$. For $u \neq v$ in $\Delta\sigma$ there are $\hat{u}, \hat{v} \in \mathcal{GT}(sig^C)$ such that $u \xrightarrow{*}_{R,i} \hat{u}$, $v \xrightarrow{*}_{R,i} \hat{v}$ and $\hat{u} \not\downarrow_{R^C} \hat{v}$. Again, it follows from the induction hypothesis that $\mathrm{eval}_\varphi^{\mathcal{A}}(u) = \hat{u}^{\mathcal{A}}$ and $\mathrm{eval}_\varphi^{\mathcal{A}}(v) = \hat{v}^{\mathcal{A}}$. As $\hat{u} \not\downarrow_{R^C} \hat{v}$, we have $\hat{u} \not\downarrow_R \hat{v}$ by Lemma 4.3(3), and thus $\hat{u}^{\mathcal{M}} \neq \hat{v}^{\mathcal{M}}$ for $\mathcal{M} = \mathcal{T}(sig, V^{\mathrm{G}})/\xleftrightarrow{*}_R$. Now $\mathcal{A}$ is a data model of *spec*, so $\mathcal{A}^C$ is isomorphic to $\mathcal{M}^C$ by Corollary 3.10 and Proposition 4.5. Hence, $\hat{u}^{\mathcal{A}} \neq \hat{v}^{\mathcal{A}}$ which shows that $\mathrm{eval}_\varphi^{\mathcal{A}}(u) \neq \mathrm{eval}_\varphi^{\mathcal{A}}(v)$. This completes the proof of (ii), and by applying (ii) we can easily conclude $\mathrm{eval}_\varphi^{\mathcal{A}}(t_1) = \mathrm{eval}_\varphi^{\mathcal{A}}(t_2)$ from $t_1 \xleftrightarrow{*}_R t_2$.

Conversely, let $\mathrm{DMod}(\mathit{spec}) \models t_1 = t_2$ and $\mathcal{M} = \mathcal{T}(sig, V^{\mathrm{G}})/\xleftrightarrow{*}_R$. By Proposition 4.5, $\mathcal{M}$ is a data model of *spec*. Hence, $\mathrm{eval}_\varphi^{\mathcal{M}}(t_1) = \mathrm{eval}_\varphi^{\mathcal{M}}(t_2)$ for every valuation φ of V^{G} in $\mathcal{M}$. Let u be the valuation of V^{G} in $\mathcal{M}$ defined by $u(x) = [x]$ for all $x \in V^{\mathrm{G}}$. Then $[t_1] = \mathrm{eval}_u^{\mathcal{M}}(t_1) = \mathrm{eval}_u^{\mathcal{M}}(t_2) = [t_2]$, which entails that $t_1 \xleftrightarrow{*}_R t_2$. $\square$

Proof of Theorem 4.7. Let $\mathcal{M} = \mathcal{T}(sig, V^{\mathrm{G}})/\xleftrightarrow{*}_R$ and u be the valuation of V^{G} in $\mathcal{M}$ with $u(x) = [x]$ for all $x \in V^{\mathrm{G}}$. Since $\mathcal{M} \in \mathrm{DMod}(\mathit{spec})$ by Proposition 4.5, we still have to show that, given a *sig*-algebra $\mathcal{A} \in \mathrm{DMod}(\mathit{spec})$ and a valuation φ of V^{G} in $\mathcal{A}$, there is a unique *sig*-homomorphism $h\colon \mathcal{M} \to \mathcal{A}$ such that $\varphi = h \circ u$, i.e. $\varphi(x) = h([x])$ for all $x \in V^{\mathrm{G}}$. By Lemma 4.6, $t_1 \xleftrightarrow{*}_R t_2$ implies $\mathcal{A} \models t_1 = t_2$ which means that $\mathrm{eval}_\varphi^{\mathcal{A}}(t_1) = \mathrm{eval}_\varphi^{\mathcal{A}}(t_2)$ for all $t_1, t_2 \in \mathcal{T}(sig, V^{\mathrm{G}})$. Thus we have $\xleftrightarrow{*}_R \subseteq \ker(\mathrm{eval}_\varphi^{\mathcal{A}})$, and by the Homomorphism Theorem (see [EM85]) we obtain a *sig*-homomorphism $h\colon \mathcal{M} \to \mathcal{A}$ such that $\mathrm{eval}_\varphi^{\mathcal{A}}(t) = (h \circ \mathrm{nat})(t) = h([t])$ for all $t \in \mathcal{T}(sig, V^{\mathrm{G}})$. In particular, $\varphi = h \circ u$. Now let $h'\colon \mathcal{M} \to \mathcal{A}$ be another *sig*-homomorphism with $\varphi = h' \circ u$. A simple induction on t proves that $h'(t) = h(t)$ for all $t \in \mathcal{T}(sig, V^{\mathrm{G}})$. Hence, h is unique. $\square$

The following lemma is applied in the proof of Theorem 4.8.

Lemma A.1 *Let spec* $= (sig, C, R)$ *be an admissible specification, and let* Γ *be a clause. Then* $\mathcal{M}(\mathit{spec}) \models \Gamma$ *iff for every inductive substitution* σ *there is a literal* λ *in* Γ *such that*

(1) λ *is of the form* $t_1 = t_2$ *and* $t_1\sigma \downarrow_R t_2\sigma$ *or*

(2) λ *is of the form* $\mathrm{def}(t)$ *and* $t\sigma \xrightarrow{*}_R \hat{t}$ *for some* $\hat{t} \in \mathcal{GT}(sig^C)$ *or*

(3) λ *is of the form* $t_1 \neq t_2$ *and* $t_1\sigma \not\downarrow_R t_2\sigma$ *or*

(4) λ *is of the form* $\neg\mathrm{def}(t)$ *and not* $t\sigma \xrightarrow{*}_R \hat{t}$ *for every* $\hat{t} \in \mathcal{GT}(sig^C)$

Proof sketch of Lemma A.1. Essentially, the lemma follows from the fact that for every valuation φ of V in $\mathcal{M}(spec)$ there is an inductive substitution σ (and vice versa) such that $\mathrm{eval}_{\varphi}^{\mathcal{M}(spec)}(t) = [t\sigma]$ for all $t \in \mathcal{T}(sig, V)$ (see the proof of Proposition 4.5). □

Proof of Theorem 4.8. Let $\mathcal{A} \in \mathrm{DMod}(spec)$ and φ be a valuation of V in $\mathcal{A}$. We have to show that there is a literal λ in Γ such that $\mathcal{A}$ satisfies λ with φ. By the Axiom of Choice, there is an inductive substitution σ such that $\sigma(x) = t$ for some $t \in \mathcal{GT}(sig^C)$ with $t^{\mathcal{A}} = \varphi(x)$ for every $x \in V^{\mathrm{C}}$, and $\sigma(x) = x$ for every $x \in V^{\mathrm{G}}$. Now (*) $\mathrm{eval}_{\varphi}^{\mathcal{A}}(t) = \mathrm{eval}_{\varphi}^{\mathcal{A}}(t\sigma)$ for all $t \in \mathcal{T}(sig, V)$, which can be shown by induction on t. Since $\mathcal{M}(spec) \models \Gamma$, there is a literal λ in Γ for σ such that one of the cases (1) to (3) of Lemma A.1 must hold – case (4) contradicts the assumption made for Γ. In case of (1), λ is of the form $t_1 = t_2$ and $\mathcal{A} \models t_1\sigma = t_2\sigma$ by Lemma 4.6. Hence, $\mathrm{eval}_{\varphi}^{\mathcal{A}}(t_1\sigma) = \mathrm{eval}_{\varphi}^{\mathcal{A}}(t_2\sigma)$, and by (*) it follows that $\mathrm{eval}_{\varphi}^{\mathcal{A}}(t_1) = \mathrm{eval}_{\varphi}^{\mathcal{A}}(t_2)$. In case of (2), λ is of the form $\mathrm{def}(t)$ and $\mathcal{A} \models t\sigma = \hat{t}$ for some $\hat{t} \in \mathcal{GT}(sig^C)$ by Lemma 4.6. Thus, $\mathrm{eval}_{\varphi}^{\mathcal{A}}(t) = \mathrm{eval}_{\varphi}^{\mathcal{A}}(t\sigma) = \mathrm{eval}_{\varphi}^{\mathcal{A}}(\hat{t}) = \hat{t}^{\mathcal{A}}$, and $\mathcal{A}$ satisfies $\mathrm{def}(t)$ with φ. In case of (3), λ is of the form $t_1 \neq t_2$ and $t_1\sigma \not\downarrow_R t_2\sigma$. The assumption for Γ and the preceding case imply that $\mathrm{eval}_{\varphi}^{\mathcal{A}}(t_1) = \hat{t_1}^{\mathcal{A}}$ and $\mathrm{eval}_{\varphi}^{\mathcal{A}}(t_2) = \hat{t_2}^{\mathcal{A}}$ for some $\hat{t_1}, \hat{t_2} \in \mathcal{GT}(sig^C)$. Obviously, we have $\hat{t_1} \not\downarrow_{R^C} \hat{t_2}$, and hence $\hat{t_1}^{\mathcal{M}(spec)} \neq \hat{t_2}^{\mathcal{M}(spec)}$. By Proposition 4.5(2), we obtain $\hat{t_1}^{\mathcal{A}} \neq \hat{t_2}^{\mathcal{A}}$, and thus $\mathrm{eval}_{\varphi}^{\mathcal{A}}(t_1) \neq \mathrm{eval}_{\varphi}^{\mathcal{A}}(t_2)$. □

Proof of Theorem 5.2. For a proof of the theorem we will apply Theorem 68(I) of [Wir95], which guarantees ω-shallow confluence of R, V^{G}. By Corollary 23 of [Wir95], ω-shallow confluence of R, V^{G} is sufficient for confluence of $\longrightarrow_R$.

The rewrite relation $\longrightarrow_R$ is not changed when, for a term $t \in \mathcal{T}(sig^C, V^{\mathrm{C}})$, the literal $\mathrm{def}(t)$ is added to the condition literals of a rewrite rule. Thus, w.l.o.g., each $l = r \leftarrow \Delta$ in R satisfies the following (simplified) weak normality condition: For each $t_1 = t_2$ in Δ there is an $i \in \{1, 2\}$ such that t_i is a $\longrightarrow_R$-irreducible ground term or $\mathrm{def}(t_i)$ occurs in Δ. This implies the quasi-normality of Definition 58 of [Wir95] which is required for the application of Theorem 68(I) – just like conservative constructors (which follows directly from our definition of constructor rule), (a weak form of) left-linearity of R and confluence of $\longrightarrow_{R^C}$.

Theorem 68(I) requires us to show some sophisticated ω-shallow joinability properties for the critical pairs of the form $(0,1)$, $(1,0)$ or $(1,1)$. Since we want to make use of the assumed complementarity of the critical pairs for showing that their conditions are infeasible, we do not really need the complete definitions of these joinability properties, but only show that the conditions under which the critical pairs must be joined are never satisfied.

Let us consider the critical pairs $((t_0\sigma, \Delta_0\sigma, a_0),\ (t_1\sigma, \Delta_1\sigma, a_1))$ of the form $(0,1)$ or $(1,0)$ at first. The conditions of the ω-shallow joinability properties allow us to assume that $(\Delta_0\Delta_1)\sigma\tau$ is "fulfilled" by $\longrightarrow_R$ (see Definition 4.2) for those inductive substitutions τ for which some special form of joinability of $t_0\sigma\tau$ and $t_1\sigma\tau$ is to be given. Due to the assumed complementarity, there are two possible cases:

Case 1. There are $u, v \in \mathcal{T}(sig, V)$ and an $i \in \{0,1\}$ such that $u = v$ or $v = u$ occurs in $\Delta_i\sigma$ and $u \neq v$ occurs in $\Delta_{1-i}\sigma$:
Under this assumption there are some $\hat{u}, \hat{v} \in \mathcal{GT}(sig^C)$ such that $\hat{u} \overset{*}{\longleftarrow}_R u\tau \overset{*}{\longrightarrow}_R \circ \overset{*}{\longleftarrow}_R v\tau \overset{*}{\longrightarrow}_R \hat{v}$ and $\hat{u} \not\downarrow_{R^C} \hat{v}$. By Lemma 4.3(3) and by Claim 1 below we get $\hat{u} \overset{*}{\longleftarrow}_{R^C} u\tau \overset{*}{\longrightarrow}_{R^C} \circ \overset{*}{\longleftarrow}_{R^C} v\tau \overset{*}{\longrightarrow}_{R^C} \hat{v}$ and then by confluence of $\longrightarrow_{R^C}$ the contradictory $\hat{u} \downarrow_{R^C} \hat{v}$.
Claim 1. We have $u\tau, v\tau \in \mathcal{GT}(sig^C)$.
Proof of Claim 1. Since the critical pair is of the form $(0,1)$ or $(1,0)$, one of the rules generating the critical pair must be a constructor rule. Thus, since u and v occur in both instantiated condition lists, they occur also in the instantiated condition list of the constructor rule, such that we have $u, v \in \mathcal{T}(sig^C, V^{\mathrm{C}})$ by Definition 4.1 and because σ is a constructor substitution, and thus $u\tau, v\tau \in \mathcal{GT}(sig^C)$ because τ is an inductive substitution.

Case 2. There are $t, \hat{u}, \hat{v} \in \mathcal{T}(sig, V)$ such that $\hat{u}, \hat{v}$ are distinct $\longrightarrow_R$-irreducible ground terms, $t = \hat{u}$ or $\hat{u} = t$ occurs in $\Delta_0\sigma$ and $t = \hat{v}$ or $\hat{v} = t$ occurs in $\Delta_1\sigma$:
Under this assumption we have $\hat{u} \overset{*}{\longleftarrow}_R t\tau \overset{*}{\longrightarrow}_R \hat{v}$. By Lemma 4.3(3) and by Claim 2 below we get $\hat{u} \overset{*}{\longleftarrow}_{R^C} t\tau \overset{*}{\longrightarrow}_{R^C} \hat{v}$ and then by the assumed confluence of $\longrightarrow_{R^C}$ the contradictory $\hat{u} \downarrow_{R^C} \hat{v}$.
Claim 2. We have $t\tau \in \mathcal{GT}(sig^C)$.
Proof of Claim 2. Since the critical pair is of the form $(0,1)$ or $(1,0)$, one of the rules generating the critical pair must be a constructor rule. Thus, since t occurs in both instantiated condition lists, it occurs also in the instantiated condition list of the constructor rule, such that we have $t \in \mathcal{T}(sig^C, V^{\mathrm{C}})$ by Definition 4.1 and because σ is a constructor substitution, and thus $t\tau \in \mathcal{GT}(sig^C)$ because τ is an inductive substitution.

Let us now consider the critical pairs $((t_0\sigma, \Delta_0\sigma, a_0),\ (t_1\sigma, \Delta_1\sigma, a_1))$ of the form $(1,1)$. The conditions of the ω-shallow joinability properties allow us to assume the following for those $n_0, n_1 \in \mathbb{N}$ and inductive substitution τ for which some special form of joinability of $t_0\sigma\tau$ and $t_1\sigma\tau$ is to be given: For $i \in \{0,1\}$, $\Delta_i\sigma\tau$ is "fulfilled" by $\longrightarrow_{R,n_i}$, and R, V^{G} is ω-shallow confluent up to $\omega{+}n_0{+}n_1$. Due to the assumed complementarity, there are two possible cases:

Case 1. There are $u_0, u_1 \in \mathcal{T}(sig, V)$ and an $i \in \{0,1\}$ such that $u_0 = u_1$ or $u_1 = u_0$ occurs in $\Delta_i\sigma$ and $u_0 \neq u_1$ occurs in $\Delta_{1-i}\sigma$:
Since $u_0 = u_1$ occurs in $\Delta_i\sigma$, by the above statement there is some $j \in \{0,1\}$ such that $\mathrm{def}(u_j)$ occurs in $\Delta_i\sigma$ or u_j is a $\longrightarrow_R$-irreducible ground term. Thus, by the above fulfilledness, there are some $u'_0, u'_1, t' \in \mathcal{GT}(sig^C)$ such that $u'_0 \overset{*}{\longleftarrow}_{R,n_{1-i}} u_0\tau \overset{*}{\longrightarrow}_{R,n_i} \circ \overset{*}{\longleftarrow}_{R,n_i} u_1\tau \overset{*}{\longrightarrow}_{R,n_{1-i}} u'_1$, $u'_0 \not\downarrow_{R^C} u'_1$, and $u_j\tau \overset{*}{\longrightarrow}_{R,n_i} t'$ or $u_j\tau$ is $\longrightarrow_R$-irreducible. By Lemma 69(4) of [Wir95] and the ω-shallow confluence of R, V^{G} up to $\omega{+}n_0{+}n_1$ this implies $u'_0 \downarrow_{R,n_i} u'_1$, which by Lemma 4.3(3) implies the contradictory $u'_0 \downarrow_{R^C} u'_1$.
Case 2. There are $t, \hat{u}, \hat{v} \in \mathcal{T}(sig, V)$ such that $\hat{u}, \hat{v}$ are distinct $\longrightarrow_R$-irreducible ground terms, $t = \hat{u}$ or $\hat{u} = t$ occurs in $\Delta_0\sigma$ and $t = \hat{v}$ or $\hat{v} = t$ occurs in $\Delta_1\sigma$:
In this case we have $\hat{u} \overset{*}{\longleftarrow}_{R,n_0} t\tau \overset{*}{\longrightarrow}_{R,n_1} \hat{v}$. By the ω-shallow confluence of R, V^{G} up to $\omega{+}n_0{+}n_1$ this implies the contradictory $\hat{u} \overset{*}{\longrightarrow}_{R,n_1} \circ \overset{*}{\longleftarrow}_{R,n_0} \hat{v}$. □

Proof of Theorem 6.2. Let $\mathcal{A} = ((A_s)_{s \in S_1}, (f^{\mathcal{A}})_{f \in F_1})$ be a data model of $spec_1$, and let φ be a valuation of V_1 in $\mathcal{A}$. We have to show that $\mathcal{A}$ satisfies Γ with φ. Let $\mathcal{B} = ((B_s)_{s \in S_0}, (f^{\mathcal{B}})_{f \in F_0})$ be the sig_0-reduct of $\mathcal{A}$, i.e. $B_s = A_s$ for each $s \in S_0$ and $f^{\mathcal{B}} = f^{\mathcal{A}}$ for each $f \in F_0$. Now $\hat{\varphi} := \varphi|_{V_0}$ is a valuation of V_0 in $\mathcal{B}$ such that $\mathrm{eval}^{\mathcal{A}}_{\varphi}(t) = \mathrm{eval}^{\mathcal{B}}_{\hat{\varphi}}(t)$ for every $t \in \mathcal{T}(sig_0, V_0)$. Due to condition (4) in Definition 6.1, $\mathcal{GT}(sig_1^{C_1}) = \mathcal{GT}(sig_0^{C_0})$ for each $s \in S_0$, and so we have $A_s^{C_1} = B_s^{C_0}$ for each $s \in S_0$. Moreover, Γ is a clause over sig_0 and V_0. Thus, we obtain: (*) $\mathcal{A}$ satisfies Γ with φ iff $\mathcal{B}$ satisfies Γ with $\hat{\varphi}$. Assume that $\mathcal{B} \in \mathrm{DMod}(spec_0)$. Since $\mathrm{DMod}(spec_0) \models \Gamma$, $\mathcal{B}$ satisfies Γ with $\hat{\varphi}$. Hence, by (*), $\mathcal{A}$ satisfies Γ with φ.

We still have to show that $\mathcal{B}$ is a data model of $spec_0$. Since $\mathcal{A}$ is a (sig_1)-model of $spec_1$, it follows with an argument similar to (*) that each rewrite rule in R_0 is valid in $\mathcal{B}$. Thus, $\mathcal{B}$ is a (sig_0)-model of $spec_0$. Let $t_1, t_2 \in \mathcal{GT}(sig_0^{C_0})$ such that $t_1^{\mathcal{B}} = t_2^{\mathcal{B}}$. Then $t_1^{\mathcal{A}} = t_2^{\mathcal{A}}$, which entails $t_1 \downarrow_{R_1^{C_1}} t_2$ by Proposition 4.5(2). Using condition (5) in Definition 6.1 we obtain $t_1 \downarrow_{R_0^{C_0}} t_2$. This implies that $\mathrm{Mod}(spec_0) \models t_1 = t_2$ as is easily shown. Hence, $\mathcal{B}$ is a data model of $spec_0$. $\square$

References

[AM95] J. Avenhaus and K. Madlener. Theorem proving in hierarchical clausal specifications. SEKI-Report SR-95-14, FB Informatik, Universität Kaiserslautern, 1995.

[BK86] J. A. Bergstra and J. W. Klop. Conditional rewrite rules: Confluence and termination. *J. Computer and System Sci.*, 32:323–362, 1986.

[BM79] R. Boyer and J Moore. *A Computational Logic*. Academic Press, 1979.

[BR95] A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *J. Automated Reasoning*, 14:189–235, 1995.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science*, pages 243–320. Elsevier Science Publ. B. V., 1990.

[DOS88] N. Dershowitz, M. Okada, and G. Sivakumar. Confluence of conditional rewrite systems. In 1^{st} *CTRS*, volume 308 of *LNCS*, pages 31–44. Springer, 1988.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification*. Springer, 1985.

[HS96] D. Hutter and C. Sengler. INKA: The next generation. In 13^{th} *CADE*, volume 1104 of *LNAI*, pages 288–292. Springer, 1996.

[KS96] D. Kapur and M. Subramaniam. New uses of linear arithmetic in automated theorem proving by induction. *J. Automated Reasoning*, 16:39–78, 1996.

[Pla85] D. A. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.

[WG94a] C.-P. Wirth and B. Gramlich. A constructor-based approach to positive/negative-conditional equational specifications. *J. Symbolic Computation*, 17:51–90, 1994.

[WG94b] C.-P. Wirth and B. Gramlich. On notions of inductive validity for first-order equational clauses. In 12^{th} *CADE*, volume 814 of *LNAI*, pages 162–176. Springer, 1994.

[Wir90] M. Wirsing. Algebraic specification. In *Handbook of Theoretical Computer Science*, pages 675–788. Elsevier Science Publishers B. V., 1990.

[Wir95] C.-P. Wirth. Syntactic confluence criteria for positive/negative-conditional term rewriting systems. SEKI-Report SR-95-09, FB Informatik, Universität Kaiserslautern, 1995.

[WK95] C.-P. Wirth and U. Kühler. Inductive theorem proving in theories specified by positive/negative-conditional equations. SEKI-Report SR-95-15, FB Informatik, Universität Kaiserslautern, 1995.