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

Ulrich Kühler and Claus-Peter Wirth

FB Informatik, Universität Kaiserslautern, D-67653 Kaiserslautern, Germany
{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 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 develop admissibility conditions that guarantee the unique existence of a distinguished data model. 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, graphs etc. are essential for the design and implementation of most software systems. 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 as 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 model class is to be associated with a specification as its meaning. 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. Due to lack of space we have to refer to [16] for an inference system on the basis of the proposed specification language. See [10] for a more detailed version of this paper including proofs.

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

Example 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).

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

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

Although E yields an appropriate axiomatization of D, the specification formalisms of various first-order frameworks for ITP do not admit the axioms in E (or their respective formulations): Firstly, $spec = (sig, E)$ is not *sufficiently complete* wrt. the *constructors* true, false, $\mathsf{0}$ and s (see [12]), since neither minus nor div are completely defined by E. In the specification formalisms of [3], [4] or [8], however, each non-constructor operation must be completely defined, i.e. for a *partial* operation (such as subtraction or division) some of its total extensions must be axiomatized. Secondly, the *rewrite-based* specification languages in [3] and [9] require the left-hand side of each conditional equation to be greater than any of its other terms wrt. a reduction order. As $\mathsf{div}(x,y)$ is *smaller* than $\mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),y))$ wrt. any simplification order (see [5]), 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 [3] for instance.

Example 2 (Example 1 continued). A tail-recursive variant of div which is non-terminating but "efficient" on its domain can be axiomatized as follows:

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

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

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

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 [3], [4], [8] or [9] 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 as given in Examples 1 and 2. The language requires

constructors for each sort in a signature and provides *constructor variables* which range over data items only. Moreover, conditional equations (or rewrite rules) with positive *and* negative conditions are admitted 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). Our admissibility conditions guarantee the unique existence of a "best" data model $\mathcal{M}(spec)$ for an admissible specification *spec* – "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). Essentially, admissibility of *spec* means that the rewrite relation induced by *spec* is confluent. Since the confluence criterion presented in Sect. 5 does not require termination of the rewrite relation, we obtain easily testable admissibility conditions that are fulfilled by many relevant specifications including the ones in Examples 1 and 2. Finally, a monotonicity result (wrt. consistent extensions of specifications) is given that further demonstrates the appropriateness of our semantics (Sect. 6).

Note that the specification language proposed in this paper has its origin in the more general specification approach of [14] and [15]. 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 [7], [12] and [5].

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 $A = (A_s)_{s \in S}$, $F^{\mathcal{A}} = (f^{\mathcal{A}})_{f \in F}$ 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-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 $i = 1, \ldots, n$ implies $f^{\mathcal{A}}(a_1, \ldots, a_n) \sim_s f^{\mathcal{A}}(b_1, \ldots, b_n)$ for all $f \in F$ with $\alpha(f) = s_1 \ldots s_n s$ and $n > 0$. 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 $a_i \in A_{s_i}$, $f^{\mathcal{Q}}([a_1], \ldots, [a_n]) = [f^{\mathcal{A}}(a_1, \ldots, a_n)]$ where $\alpha(f) = s_1 \ldots s_n s$.

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, $\xrightarrow{+}$ its transitive closure and $\xrightarrow{*}$ its transitive-reflexive closure. By $\downarrow$, we denote the *joinability* relation $\xrightarrow{*} \circ \xleftarrow{*}$ of $\longrightarrow$, that is $t_1 \downarrow t_2$ if there is a t such that $t_1 \xrightarrow{*} t \xleftarrow{*} t_2$.

Moreover, $\longrightarrow$ is called *confluent* if $\xleftarrow{*} \circ \xrightarrow{*} \subseteq \downarrow$. 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 sort $s \in S$. Note that $\xleftrightarrow{*}$ 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

It is generally accepted that *initial algebra semantics* for (positive conditional) equational specifications is rarely appropriate for formalizing data types with partial or non-terminating operations (see [12], [15]). Consider e.g. the specification with those axioms from Example 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 $\mathsf{less}(\mathsf{0}, \mathsf{minus}(\mathsf{0}, \mathsf{s}^{n+1}(\mathsf{0})))$ 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. Our algebraic specification language, however, does not require sufficient completeness for adequate representations of data types with partial operations any more – mainly because we dignify the role constructors play in describing the data items of a data type (see below).

3.1 Syntax of Specifications with Constructors

For a data type D to be adequately represented by certain model classes of a specification of D (see below), we require 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 $sig^C = (S, C, \alpha|_C)$ 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 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 $\sigma\colon V \to \mathcal{T}(sig, V)$ 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 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*.

Definition 3. 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).

3.2 Model Semantics of Specifications with Constructors

We first 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 4. 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 $s \in S$, $A_s^C = \{\, t^{\mathcal{A}} \in A_s \mid t \in \mathcal{GT}(sig^C)_s \,\}$; and

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

It is easily seen that the data reduct $\mathcal{A}^C$ is in fact a sig^C-algebra.[1] Given $X \subseteq V$, 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, as is stated in the following lemma.

[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 $\mathcal{A}$.

Lemma 5. *Let $\mathcal{A}$ and $\mathcal{B}$ be sig-algebras, and let h: $\mathcal{A} \to \mathcal{B}$ be a sig-homomorphism. Let $h^C = (h_s^C)_{s \in S}$ be the family of functions defined by $h_s^C = h_s|_{A_s^C}$ for all $s \in S$. Then h^C: $\mathcal{A}^C \to \mathcal{B}^C$ is a sig^C-epimorphism.*

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

Definition 6. 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 φ: $X \to \mathcal{A}$ such that $\varphi(x) \in A_s^C$ for every $x \in X_s \cap V_s^{\mathrm{C}}$ and $\varphi(x) \in A_s$ for every $x \in X_s \cap V_s^{\mathrm{G}}$. $\mathrm{eval}_\varphi^{\mathcal{A}}$ denotes *the sig*-homomorphism from $\mathcal{T}(sig, X)$ to $\mathcal{A}$ that extends φ.

(ii) Let φ be a valuation of V in $\mathcal{A}$. If $\mathrm{eval}_\varphi^{\mathcal{A}}(t_1) = \mathrm{eval}_\varphi^{\mathcal{A}}(t_2)$ then $\mathcal{A}$ *satisfies* $t_1 = t_2$ *with* φ. $\mathcal{A}$ satisfies $\mathrm{def}(t)$ for $t \in \mathcal{T}(sig, V)_s$ with φ if $\mathrm{eval}_\varphi^{\mathcal{A}}(t) \in A_s^C$. $\mathcal{A}$ satisfies a negative literal $\neg\lambda$ with φ if $\mathcal{A}$ does not satisfy λ with φ. $\mathcal{A}$ satisfies a clause Γ with φ if there is a literal in Γ that $\mathcal{A}$ 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 7. *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 8. 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 any conditional equation is valid in the trivial *sig*-algebra, $\mathrm{Mod}(spec)$ is not empty for any specification with constructors *spec*. Still, we do not really consider $\mathrm{Mod}(spec)$ an appropriate inductive semantics of *spec*, as $\mathrm{Mod}(spec)$ includes also those *sig*-models of *spec* which *confuse* data items, i.e. these models satisfy equations between constructor ground terms that are not valid in all *sig*-models of *spec*.

3.3 Data Models

In eliminating those models from $\mathrm{Mod}(spec)$ 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 9. 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 $\mathrm{DMod}(spec)$ denote the class of all data models of $spec$.

Note that $\mathrm{DMod}(spec)$ may be empty:

Example 10. Let $sig = (S, F, \alpha)$ be the signature with $S = \{\mathsf{any}\}$, $F = C \cup \{\mathsf{d}\}$, $C = \{\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3\}$, and $\alpha(o) = \mathsf{any}$ for each $o \in F$. Let $E = \{\, \mathsf{d} = \mathsf{c}_1 \leftarrow \mathsf{c}_1 \neq \mathsf{c}_3,\ \mathsf{d} = \mathsf{c}_2 \leftarrow \mathsf{c}_1 \neq \mathsf{c}_3 \,\}$, and let $spec = (sig, C, E)$.

One easily sees that neither $\mathrm{Mod}(spec) \models \mathsf{c}_1 = \mathsf{c}_3$ nor $\mathrm{Mod}(spec) \models \mathsf{c}_1 = \mathsf{c}_2$. Let $\mathcal{A} \in \mathrm{Mod}(spec)$. If $\mathsf{c}_1^{\mathcal{A}} = \mathsf{c}_3^{\mathcal{A}}$ then $\mathcal{A}$ is not a data model of $spec$. Otherwise, due to E, 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 $spec$ either.

The data reducts of any two data models in $\mathrm{DMod}(spec)$ are isomorphic:

Lemma 11. *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 12. *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 $\mathcal{B} \models t_1 = t_2$. In section 4 we will show that all "defined" terms are uniformly evaluated in all data models of *admissible* specifications (see Theorem 20).

4 Admissibility Conditions

So far we have placed no restrictions on the set of positive/negative conditional equations in a specification with constructors. However, Example 10 shows 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 unique existence of a distinguished data model $\mathcal{M}(spec)$ for any admissible specification $spec$. Hence, $\mathrm{DMod}(spec) \neq \emptyset$.

Our admissibility conditions strongly rely on 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).

Definition 13. 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 $spec = (sig, C, R)$ 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 [1], [14]). 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 20 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 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 hold. 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 holds if $u\sigma \overset{*}{\longrightarrow}_R \hat{u}$ for some constructor ground term $\hat{u}$.

Definition 14. 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 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 15.

(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 10 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 10 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, because then $t_1 \xleftrightarrow{*}_R t_2$ is equivalent to $t_1 \downarrow_R t_2$ for $t_1, t_2 \in \mathcal{GT}(sig^C)$, and applying Lemma 15(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 [7], [1]).

A further admissibility condition is needed to ensure that the sig-algebra $\mathcal{T}(sig, V^{\mathrm{G}})/\xleftrightarrow{*}_R$ is a (data) model of $spec$. 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 16. 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 $\mathcal{M}(spec)$

We now show that, for any admissible specification *spec*, $\mathcal{T}(sig, V^{\mathrm{G}})/\overset{*}{\longleftrightarrow}_R$ is a "best" data model of *spec* – "best" in the sense that its algebraic properties resemble those of the initial model of a usual equational specification. This gives rise to two appropriate kinds of semantics for specifications with constructors of data types with partial operations: (i) DMod(*spec*) and (ii) the (isomorphism) class of the standard data model $\mathcal{M}(spec) = \mathcal{T}(sig, V^{\mathrm{G}})/\overset{*}{\longleftrightarrow}_R$.

Proposition 17. *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 a 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 $\longrightarrow_R$ is a sound and complete operationalization of equality in all data models of *spec*. It is of central importance in the proofs of Theorems 19 and 20.

Lemma 18. *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 $DMod(spec) \models t_1 = t_2$.*

Recall (e.g. from [7]) that a sig-algebra $\mathcal{F}(X)$ is *free* over $X \subseteq V$ in a class K of sig-algebras 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 $h\colon \mathcal{F}(X) \to \mathcal{A}$ such that $\varphi = h \circ u$. 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.

Theorem 19. *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. The significance of $\mathcal{M}(spec)$ 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 20. *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 an admissible specification *spec*.

Note that Theorems 19, 20 and 23 have counterparts in the more general specification approach of [14] and [15] from which the specification approach 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 [6] for a definition of "decreasing") or even non-terminating (see Example 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 can even happen when termination is given (combined with left-linearity and normality; see [6], p. 36, Example B). 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 [6], p. 36, Example A; taken from [2]). Therefore, syntactic confluence criteria for non-terminating rewrite systems need to require 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 22) 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 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 $\longrightarrow_R$. Note that different criteria may be applied to handle these subproblems. 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 termi-

[2] See Theorems 68 and 71 of [13]. Theorem 68 is the version for ω-shallow confluence, Theorem 71 for ω-level confluence.

nation preconditions are practically feasible in general. One reason for this is 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 [6]). 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 [13]. Sometimes, however, confluence of $\longrightarrow_{R^C}$ can only be shown using the semantic knowledge of the specifier: either with semantic confluence criteria in the style of [11] (see Theorem 6.5 of [15]) or in a way that only works for the specific set of constructor rules at hand.

Before we can formally present our syntactic confluence criterion, we have to introduce some more notions.

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 21. 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 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 $\hat{u}, \hat{v}$ 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: 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 $\big((0, y{\neq}0 \wedge \mathsf{less}(x,y){=}\mathsf{true}, 1), (\mathsf{s}(\mathsf{div}(\mathsf{minus}(x,y),y)), y{\neq}0 \wedge \mathsf{less}(x,y){=}\mathsf{false}, 1)\big)$. It is complementary according to Part (b) of Definition 21 (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 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 obviously is the case according to Part (a) of Definition 21.

In addition, two further notions are used in our confluence criterion for 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.

Note that the rewrite systems in Examples 1 and 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 of Examples 1 and 2.

Theorem 22. *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 [2] (also cited in [6]) 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 again could 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 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 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 admit negative conditions in conditional equations. Thus, 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 div1 in Sect. 1).

The appropriateness of the *semantics* of our specification language can also be demonstrated with Examples 1 and 2. Consider the class of data models of the specification *spec'*. Since *spec'* is admissible (by Theorem 22), the standard data model $\mathcal{M}(spec')$ exists and DMod(*spec'*) is not empty. Obviously, every data model of *spec'* has the two truth values and the natural numbers as its data reduct. Hence, in spite of "junk terms", clauses like $\mathsf{plus}(x,y) = \mathsf{plus}(y,x)$ or clause (1) (see Example 2) are valid in DMod(*spec'*) if 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 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" extensions of the specification. For DMod(*spec*), this monotonicity property is given as is shown in the following.

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 $V_i = (V_{i,s})_{s \in S_i}$ is a variable system for sig_i. We say that $spec_1$ is a *constructor-consistent extension* of $spec_0$ if the following holds: (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 23. *Let $spec_0 = (sig_0, C_0, R_0)$ and $spec_1 = (sig_1, C_1, R_1)$ be admissible specifications. 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 20 and 23 imply a similar monotonicity for the validity in $\mathcal{M}(spec)$ that, 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 $\mathcal{M}(spec)$ for the specification of Example 1 (but not in DMod(*spec*)). After a constructor-consistent extension of *spec* with the rewrite rule $\mathsf{minus}(\mathsf{0},\mathsf{s}(y)) = \mathsf{0}$, however, 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 23 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) which makes 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). 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 [3] and [9]: We can e.g. easily axiomatize operations that are naturally defined by *destructor recursion* (such as div or mergesort).

Note that the reduction order given by a terminating rewrite relation is usually not necessary for rewrite-based ITP as is shown in [16], 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.

References

1. J. Avenhaus and K. Madlener. Theorem proving in hierarchical clausal specifications. SEKI-Report SR-95-14, FB Informatik, Universität Kaiserslautern, 1995.
2. J. A. Bergstra and J. W. Klop. Conditional rewrite rules: Confluence and termination. *J. Computer and System Sci.*, 32:323–362, 1986.
3. A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *J. Automated Reasoning*, 14:189–235, 1995.
4. R. Boyer and J Moore. *A Computational Logic*. Academic Press, 1979.
5. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science*, pages 243–320. Elsevier Science Publishers B. V., 1990.
6. 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.
7. H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification*. Springer, 1985.
8. D. Hutter and C. Sengler. INKA: The next generation. In 13^{th} *CADE*, volume 1104 of *LNAI*, pages 288–292. Springer, 1996.
9. D. Kapur and M. Subramaniam. New uses of linear arithmetic in automated theorem proving by induction. *J. Automated Reasoning*, 16:39–78, 1996.
10. U. Kühler and C.-P. Wirth. Conditional equational specifications of data types with partial operations for inductive theorem proving. SEKI-Report SR-96-11, FB Informatik, Universität Kaiserslautern, 1996.
11. D. A. Plaisted. Semantic confluence tests and completion methods. *Information and Control*, 65:182–215, 1985.
12. M. Wirsing. Algebraic specification. In *Handbook of Theoretical Computer Science*, pages 675–788. Elsevier Science Publishers B. V., 1990.
13. C.-P. Wirth. Syntactic confluence criteria for positive/negative-conditional term rewriting systems. SEKI-Report SR-95-09, FB Informatik, Universität Kaiserslautern, 1995.
14. C.-P. Wirth and B. Gramlich. A constructor-based approach to positive/negative-conditional equational specifications. *J. Symbolic Computation*, 17:51–90, 1994.
15. 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.
16. 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.