Chapter 10

Proof in Flat Specifications[1]

Peter Padawitz

August 2, 1998

Contents

10.1 Introduction

This chapter deals with the verification issues that arise with the logic-oriented specification of data types. We put particular emphasis on

- a uniform syntax for constructor-based specifications of both visible and hidden data types,
- Gentzen clauses, rules and proofs as a uniform schema for presenting (proofs of) conjectures in a natural, flexible, structured and implementable way that keeps the gap between informal reasoning and formal deduction as small as possible,
- a simple model- and proof-theoretical basis to which all more or less advanced rules and methods can be reduced for showing their correctness,
- providing the reader with syntactical criteria for the main conditions on a specification that shall be amenable to efficient proof and prototyping methods.

Deductive aspects of specifications are also treated in other chapters of this book. Sections 2.6 through 2.9 provide basic notions and results for equational reasoning, i.e. for proving equational conjectures about specifications with unconditional equational axioms. Section 4.5 deals with entailment (= inference) relations in general. Chapter 9 handles axiomatization and proof in terms of rewrite systems. Chapter 11 presents proof systems for specifications that are given as expressions consisting of specification-building operations (see Sections 4.6 and 6.3). Chapter 15 provides design rules for specifications similarly to those that have been developed for functional and logic programs (see [MPS 93]).

[1] revised version of a contribution to: E. Astesiano, H.-J. Kreowski, B. Krieg-Brückner, eds., IFIP WG 1.3 State-of-the-Art Report, *Algebraic Foundations of System Specification*, Springer 1998, to appear

In [Pad 88] we pointed out that the theoretical background of formal specification originates in four mathematical research areas: recursion theory (REC), automated theorem proving (ATP), abstract data types (ADT) and term rewriting systems (TRS). The boundaries between these areas are about to vanish: functional and logic programming and the development of specification languages fused REC and ADT to type theory; automated reasoning systems involve rewrite methods. However, the results of REC, ATP, ADT and TRS still have little impact on software design. This situation might be remedied if the users of REC, ATP, ADT and TRS, respectively, would also look a little more beyond the special theory they are employing and take into account other foundations and results as well. So far ATP is mainly concerned with *relational* proof methods and their use in knowledge bases and classical AI applications. Its origin in algebra makes TRS view programs and data types as purely *functional* rewrite systems. REC deals with operational and denotational models of *computing and complexity*, while ADT focuses on the mere formal *description* of higher levels in software design, such as modularization, parameterization and refinement.

Only tailor-made syntactical, model- and proof-theoretic frameworks will meet the actual goals of formal software development. Although deductive and operational concepts still attract less attention in the ADT community than the model-theoretic issues, we face a great number of rule-based systems that are claimed to support the design of programs and data types. Less than thirty years ago there were only *assertion logics* for verifying imperative programs and the *fixpoint theory* of recursive functions. Nowadays the approaches concerned with deduction range from rewrite systems, type theory and frameworks dealing with the interfaces between several logics (see Chapter 4) to a large number of implemented provers and proof checkers. [ARS 96] lists more than sixty currently available automated reasoning systems.

Although this chapter is supposed to give an overview of the state of the art in its topic, we think that the non-specialized reader would not gain much from a pure list of current deduction-oriented approaches to formal specification. Instead, we intend to integrate the main concepts, notions and definitions of several approaches into a uniform framework, which is based on **many-sorted predicate logic with equality** and admits functional, relational, state-oriented ("transitional") and all kinds of mixed specification and reasoning. Moreover, the uniform framework allows us to handle **visible** structures, which are representable by unique **normal forms**, along with **hidden** structures whose identity is defined in terms of transition relations, attributes or destructors. Many-sorted predicate logic with equality is also sufficient for incorporating functional sorts, local definitions, polymorphism, parameterization and refinements. We do not start out from **order-sorted** logic (see Sections 2.10.4 and 9.5) because subtype relations can be simulated by unary predicates. Moreover, definedness predicates for expressing partiality—which is often handled with subtypes—are implicitly included in our basic notion of a **standard specification** (Def. 10.2.4). Let us start with a well-known visible type.

Example 10.1.1 The following specification consists of a **parameter specification** ENTRY and an extension LIST of ENTRY for presenting finite lists of elements taken from a model of ENTRY. While the axioms of a parameter specification may be arbitrary many-sorted first-order formulas, the extension adds only Horn or co-Horn axioms to the parameter.

ENTRY

`sorts`	$entry\ \ bool$
`funs`	$true, false :\to bool$
	$not : bool \to bool$
	$eq : entry \times entry \to bool$
`preds`	$\_ \not\equiv \_ : entry \times entry$
	$\_ \leq \_ : entry \times entry$
	$\_ > \_ : entry \times entry$

$$
\begin{array}{ll}
\texttt{vars} & b : bool \quad x, y : entry \\
\texttt{axioms} & \forall b : (b \equiv true \lor b \equiv false) \\
 & \neg(true \equiv false) \\
 & not(true) \equiv false \land not(false) \equiv true \\
 & \forall x, y : (eq(x, y) \equiv true \iff x \equiv y) \\
 & \forall x, y : (x \equiv y \lor x \not\equiv y) \\
 & \forall x, y : \neg(x \equiv y \land x \not\equiv y) \\
 & \forall x, y : (x \leq y \lor x > y) \\
 & \forall x, y : \neg(x \leq y \land x > y)
\end{array}
$$

LIST = signature of ENTRY +

$$
\begin{array}{ll}
\texttt{vissorts} & list = list(entry) \\
\texttt{consts} & nil :\rightarrow list \\
 & \_ :: \_ : entry \times list \rightarrow list \\
\texttt{defuns} & [\_] : entry \rightarrow list \\
 & \_@\_ : list \times list \rightarrow list \\
 & map : (entry \rightarrow entry) \times list \rightarrow list \\
 & filter : (entry \rightarrow bool) \times list \rightarrow list \\
 & remove : entry \times list \rightarrow list \\
\texttt{preds} & \_ \in \_ : entry \times list \\
 & \_ \notin \_ : entry \times list \\
 & sorted : list \\
 & exists, forall : (entry \rightarrow bool) \times list \\
\texttt{vars} & x, y : entry \quad L, L' : list \quad f : entry- > entry \quad g : entry- > bool \\
\texttt{Horn axioms} & [x] \equiv x :: nil \\
 & nil@L \equiv L \\
 & (x :: L)@L' \equiv x :: (L@L') \\
 & map(f, nil) \equiv nil \\
 & map(f, x :: L) \equiv f(x) :: map(f, L) \\
 & filter(g, nil) \equiv nil \\
 & filter(g, x :: L) \equiv x :: filter(g, L) \quad \Leftarrow \quad g(x) \equiv true \\
 & filter(g, x :: L) \equiv filter(g, L) \quad \Leftarrow \quad g(x) \equiv false \\
\text{(A)} & remove(x, L) \equiv filter(\lambda y.eq(x, y), L) \\
 & x \in y :: L \quad \Leftarrow \quad x \equiv y \lor x \in L \\
 & x \notin nil \\
 & x \notin y :: L \quad \Leftarrow \quad x \equiv y \land x \notin L \\
 & sorted(nil) \\
 & sorted([x]) \\
 & sorted(x :: y :: L) \quad \Leftarrow \quad x \leq y \land sorted(y :: L) \\
 & exists(g, x :: L) \quad \Leftarrow \quad g(x) \equiv true \lor exists(g, L) \\
 & forall(f, nil) \\
 & forall(g, x :: L) \quad \Leftarrow \quad g(x) \equiv true \land forall(g, L)
\end{array}
$$

LIST adds to the signature of ENTRY **visible sorts** (`vissorts`), **constructors** (`consts`), **defined functions** (`defuns`) and **predicates** (`preds`). Structured sort symbols such as $list(entry)$ stand for polymorphic types in the sense of functional programming languages. Sorts of a parameter specification such as $entry$ correspond to type variables. Predicates specified by Horn clauses will be interpreted as least relations satisfying their axioms. λ-expressions are used as short notations for additional function symbols. For instance, Axiom A is an abbreviation of two equations:

$$\begin{aligned} remove(x,L) &\equiv filter(G(x),L) \\ G(x)(y) &\equiv eq(x,y). \end{aligned}$$

Here G is a function symbol of sort $entry \rightarrow (entry \rightarrow bool)$. Bisimilarity or behavioral equivalence and other *ν-predicates* will be specified by *co-Horn clauses* and interpreted as greatest relations satisfying their axioms (Def. 10.2.3 and Section 10.3). ❏

Note the difference between predicates on the one hand and Boolean (defined) functions on the other hand. For obtaining efficient proof methods we will assume that all defined functions are total(izable), in other words: they have decidable domains. But Boolean functions with decidable domains do not have the expressive power of predicates because the latter can be regarded as Boolean functions with *semi*-decidable domains.

A pair $PSP = (PAR, SP)$ such as (ENTRY,LIST) is called a **parameterized specification** (see Section 6.4). Its semantics is given by the class of **actualizations of PSP by PAR-models**. An actualization of PSP is an unparameterized specification **SP(A)** that combines SP with a PAR-model A in the following way. The carrier elements of A are regarded as constructors of $SP(A)$. The interpretation F^A of a function or predicate $F \in PAR$ in A becomes the set $\{F(a) \equiv F^A(a) \mid a \in A\}$ resp. $\{F(a) \mid a \in F^A\}$ of axioms of $SP(A)$.

If PAR is empty, then the semantics of PSP reduces to a single model, namely the initial SP-model (Theorem 10.2.12). If PAR is non-empty, all proofs of conjectures about PSP must presuppose an *arbitrary* actualization of PSP. Then the axioms of PAR are the only properties of the parameter such proofs will refer to.

Hence even in the parametric case we reason about a flat specification (see Section 2.7) because we have a single—though arbitrary—actualization in mind. This differs from the approach taken in Chapter 11 where the proof systems yield, in the first place, a *structured operational semantics* for specification-building operations (see Sections 4.6 and 6.3). The specifications we deal with here may also *originate* in expressions over specification-building operations. But these are somewhat "flattened away" in our presentations.

Section 10.2 introduces *standard specifications* as a uniform syntactical schema for presenting visible as well as hidden types. Loose, reachable and initial semantics as well as deductive and inductive theorems of standard specifications are defined, discussed and related to each other in various ways also in Section 10.2. Section 10.3 is devoted to the special model- and proof-theoretical issues that come up with hidden sorts, infinity and behavioral equivalence. Coinductivity as a syntactical criterion for the congruence property of behavioral equivalence and the derivation of axioms for complements deserve particular attention in Section 10.3. Moreover, Sections 10.2 and 10.3 provide the model- and proof-theoretical basis mentioned above to which examples and results presented in the sequel refer: the *cut calculus*, the *initial model* and, as its natural extension to hidden types, the *final model*.

Section 10.4 introduces the most natural and flexible schema of a *Gentzen proof* and shows how goal solving, program synthesis and other verification tasks fit into this schema. Section 10.5 deals with the main properties of a standard specifications: *completeness*, *confluence*, *termination* and *consistency* with respect to base models. Syntactical criteria that ensure these properties rely on the use of the (Horn) axioms of a standard specifications as rewrite rules (see Chapter 9). Section 10.5 provides both the general notions and results that justify this use and criteria for confluence and consistency that cover more or less special situations. *Implicit induction* denotes a class of proof methods, which are based on notions discussed in Section 10.5. Criteria for proving theorems about standard specifications by implicit induction are developed in Section 10.6. Section 10.7 puts all results presented so far together and infers the main rules for building up Gentzen proofs. The rules can roughly be classified into *unfold rules* that "execute" function and predicate definitions, lemma applications and fixpoint rules. Finally, Section 10.8 presents narrowing as the extension of rewriting to a basic rule for solving goals and embeds narrowing into Gentzen proofs.

10.2 Standard specifications

For the basic notions of many-sorted logic we refer the reader to Chapter 2. Here we set up the definition of a standard specification SP, which refines that of a many-sorted one insofar as the constructors of the specified domains are separated from the functions to be defined by SP and insofar as visible domains are distinguished from hidden ones whose objects are identified only through their *behavior*.

Given a set S of sorts, $s_1, \ldots, s_n \in S$ and an S-sorted set A, $A_{s_1 \ldots s_n}$ stands for the product $A_{s_1} \times \cdots \times A_{s_n}$. An **$S$-sorted relation R on A** is an S-sorted set such that for all $s \in S$, $R_s \subseteq A_s \times A_s$. R extends to an S^+-sorted relation on A^+ as follows: $(a_1, \ldots, a_n)R(b_1, \ldots, b_n)$ iff $a_i R_{s_i} b_i$ for all $1 \leq i \leq n$.

For any expression (term or formula) e, **var(e)** denotes the set of variables occurring in e. e is **linear** in a set Y of variables if each variable of Y occurs at most once in e. e is **ground** if $var(e)$ is empty. Both **e(t)** and **e[t/u]** stand for expressions that include the subexpression t. $e[t/u]$ indicates that the subexpression u of e has been replaced by t. Set brackets $\{$ and $\}$ in a formula enclose either quantified variables or optional subexpressions. For instance, $t\{\equiv u\}$ stands for t or $t \equiv u$.

We introduce special many-sorted signatures that separate visible from hidden sorts and data constructors from defined functions and include "undefined elements" as well as particular predicates for specifying definedness and equality relations.

Definition 10.2.1 (signatures, terms and atoms) A **standard signature** $\Sigma = (S, F, P)$ consists of a set S of **sorts** and S^+-sorted sets F of **function symbols** and P of **predicates**. S splits into a set $visS$ of **visible sorts** and a set $hidS$ of **hidden sorts**. Given $w \in S^+$, $f \in F_w \cup P_w$ is **visible** if $w \in visS^+$. Otherwise f is **hidden**.

F splits into a set CO of **constructors** and a set DF of **defined functions**. For all $w \in S^*$ and $s \in S$, the elements of CO_{ws} are **s-constructors**, and $s \in visS$ implies $w \in visS^*$, i.e. *visible constructors do not have hidden arguments*. There is a **bottom constant** $\perp_s$ for each visible sort s. The S-sorted set $NF_\Sigma(X)$ of **Σ-normal forms** is defined inductively as follows:

- $X \cup \{\perp_s \mid s \in visS\} \subseteq NF_\Sigma(X)$.
- For all visible $f \in CO_{ws}$ and $t \in (NF_\Sigma(X) \setminus \{\perp\})_w$, $f(t) \in NF_\Sigma(X)_s$.
- For all hidden $f \in CO_{ws}$ and $t \in NF_\Sigma(X)_w$, $f(t) \in NF_\Sigma(X)_s$.

$T_\Sigma(X)$, T_Σ and NF_Σ denote the S-sorted sets of Σ-terms (defined as usual), ground Σ-terms and ground Σ-normal forms, respectively. Σ is **inhabited** if for all sorts s of Σ, $T_{\Sigma,s}$ is not empty. $t \in T_\Sigma(X)_s$ is **visible** resp. **hidden** if s is visible resp. hidden.

For all $w \in S^+$, P_w splits into a set $\mu\Sigma_w$ of **μ-predicates** and a set $\nu\Sigma_w$ of **ν-predicates**. For all $s \in S$, $\mu\Sigma_{ss}$ and $\nu\Sigma_{ss}$ include the **s-equality predicate** $\equiv_s$ and the **s-bisimulation predicate** $\sim_s$, respectively. All non-equality predicates are called **logical predicates**. For all $s \in visS$, $\mu\Sigma_s$ includes the **s-definedness predicate** Def_s. For all $s \in hidS$ there may be

(1) functional **s-attributes**, i.e. function symbols in $DF_{svs'}$ where $v \in S^*$ and $s' \in visS$,

(2) s-**destructors**, i.e. function symbols in $DF_{svs'}$ where $v \in S^*$ and $s' \in hidS$,

(3) relational s-**attributes with output** $w \in visS^*$, i.e. predicates in $\mu\Sigma_{svw}$ where $v \in S^*$,

(4) s-**transitions with output** $w \in visS^*$, i.e. predicates in $\mu\Sigma_{svws'}$ where $v \in S^*$ and $s' \in hidS$.

$\tau\Sigma$ denotes the set of all transitions and equality predicates for hidden sorts of Σ.

Given $r \in P_w$ and $t \in T_\Sigma(X)_w$, $r(t)$ is a Σ-**atom**. If r is an equality predicate, then $r(t)$ is an **equation**. Otherwise $r(t)$ is a **logical atom**. An equation $t \equiv t'$ is **visible** resp. **hidden** if t is visible resp. hidden. An equation between term tuples t and t' stands for the conjunction of the equations between corresponding components of t resp. t'. Given a term tuple t and a hidden term u, an atom $\delta(t,u)$ with $\delta \in \tau\Sigma$ is called a **state generator with target** u.

Σ is **visible** if $hidS$ is empty. The greatest visible subsignature of Σ is denoted by **vis**Σ. ❑

As usual, a function symbol $f \in F_{ws}$ is written as $f : w \to s$. A predicate $r \in P_w$ is written as $r : w$. Equality and definedness predicates are implicit, i.e. not listed explicitly in Σ. Attributes, destructors and transitions are used for specifying the bisimulation predicates (see Def. 10.2.4).

Note that all visible Σ-normal forms are $vis\Sigma$-terms. The purpose of ground normal forms is to represent data. A specification is *functional* if all visible data have unique normal form representations (Def. 10.2.10). Hence visible data are identified by the *structure* of their normal forms and thus functions and predicates can be specified in terms of this structure and conjectures about them can be proved by induction on the structure. Hidden data also have normal forms, but here the representation need not be unique. Instead, the identity of hidden objects is defined in terms of the bisimulation predicates of Σ. The same conceptual difference between visible and hidden data also applies to definedness: visible sorts have bottom constants in order to make undefined values visible; hidden normal forms, however, are mere names for objects, so that hidden bottom constants would not make sense.

Let Σ be a standard signature with sort set S and X be an S-sorted set of variables.

Given a further signature $\Sigma' = (S', F', P')$, a **signature morphism** $\sigma : \Sigma' \to \Sigma$ consists of a function $\sigma_{sorts} : S' \to S$ and $(S')^+$-sorted sets of functions $\sigma_{funs} = \{\sigma_w : F'_w \to F_{\sigma(w)}\}$ and $\sigma_{preds} = \{\sigma_w : P'_w \to P_{\sigma(w)}\}$ such that for all $f : w \to s \in F'$, $\sigma(f) : \sigma(w) \to \sigma(s)$ and for all $r : w \in P'$, $\sigma(r) : \sigma(w)$ (see Section 2.5).

Definition 10.2.2 (substitutions) An S-sorted function $\sigma : X \to T_\Sigma(X)$ is called a **substitution** over Σ.[2] The **domain of** σ, $dom(\sigma)$, is the set of all variables x with $x\sigma \neq x$. Given $Y \subseteq X$, σ_Y denotes the restriction of σ with $dom(\sigma) \subseteq Y$. The substitution with the empty domain is called the **identity substitution** and denoted by id. σ is **ground** resp. **ground normal** if $dom(\sigma)\sigma \subseteq T_\Sigma$ resp. $dom(\sigma)\sigma \subseteq NF_\Sigma$. T_Σ^X denotes the set of ground substitutions over Σ. The **instance** $t\sigma$ of a term or atom t **by** σ is obtained from t by replacing each variable x by $x\sigma$. A term or atom u **subsumes** t, written $u \leq t$, if t is an instance of u. t is **unifiable** with u and σ is a **unifier** of t and u if $t\sigma = u\sigma$. σ **subsumes** a substitution τ, written $\sigma \leq \tau$, if for all $x \in X$, $x\sigma \leq x\tau$. σ is **minimal** or **most general** with respect to a property P of substitutions if σ fulfils P and subsumes all substitutions fulfilling P.

Let $Y, Z \subseteq X$ and σ, τ be substitutions. The composition $\sigma\tau$ is defined by $x(\sigma\tau) = (x\sigma)\tau$ for all $x \in X$. σ is a **renaming of** Y **away from** Z if $dom(\sigma)\sigma \subseteq X$, $|Y\sigma| = |Y|$ and $Y\sigma \cap Z = \emptyset$. σ is a **skolemization** if for all $x \in dom(\rho)$, $x\sigma$ is a "new" constant. Skolemizations prevent variables of an expression e from being replaced when a further substitution is applied to $e\rho$ (Def. 10.2.13). ❑

Definition 10.2.3 (formulas) A **goal** is a finite conjunction G of Σ-atoms. The empty conjunction is

[2]Both substitutions and signature morphisms are usually denoted by small Greek letters.

called **empty goal** and denoted by $\emptyset$ or $TRUE$.

A **goal set** is a finite disjunction of formulas $\exists Y G$ where $Y \subseteq X$ and G is a goal. The empty disjunction is called **empty goal set** and denoted by $FALSE$. Given goal sets GS and HS, the formula $GS \Leftarrow HS$ is called a **Gentzen clause**.

A goal G is a **state goal** resp. **μ-goal** resp. **ν-goal** if G consists of atoms $r(t)$ with $r \in \Sigma \setminus \tau\Sigma$ resp. $r \in \mu\Sigma$ resp. $r \in \nu\Sigma$. A goal G of the form $\delta_1(t_1, x_1) \wedge \cdots \wedge \delta_n(t_n, x_n)$ is a **state generator goal with output** $Y = \{x_1, \ldots, x_n\}$ if $\delta_1, \ldots, \delta_n \in \tau\Sigma$ and G is linear in Y.

The set of **modal formulas ϕ with output** $Y \subseteq var(\phi)$ is defined inductively as follows:

(1) A state goal is a modal formula with empty output.

(2) A state generator goal with output Y is a modal formula with output Y.

(3) If ϕ and ψ are modal formulas with output Y resp. $\emptyset$, then $\phi \wedge \psi$ is a modal formula with output Y.

(4) If ϕ and ψ are modal formulas with output Y resp. Z, then $\phi \vee \psi$ is a modal formula with output $Y \cup Z$.

(5) If G is a state generator goal with output Y, ϕ is a modal formula with output Z and $Z \cap var(G) = \emptyset$, then $\exists Y(G \wedge \phi)$ is a modal formula with output Z.

(6) If G is a state generator goal with output Y and ϕ is a modal formula with empty output, then $\forall Y(G \Rightarrow \phi)$ is a modal formula with empty output.

(7) If ϕ is a first-order formula and ψ is a modal formula such that for all Σ-structures A and $b : X \rightarrow A$, $A \models_b \phi$ iff $A \models_b \psi$ (cf. 10.2.6), then ϕ is also modal.

Given an atom $r(t)$ and a goal G, the formula $r(t) \Leftarrow G$ is a **Horn clause**. If r is logical, then $r(t) \Leftarrow G$ is a Horn clause **for** r. A Horn clause of the form $f(t) \equiv u \Leftarrow G$ is a Horn clause **for** f. Given a logical atom $r(t)$, a goal H and a goal set GS, the formula $r(t) \Rightarrow (H \Rightarrow GS)$ is a **co-Horn clause for** r. ❑

Gentzen clauses are as expressive as arbitrary first-order formulas.[3] Moreover, functional-logic programs are merely Horn clauses and most conjectures about them, such as pre/post-conditions, invariants, etc., are given as Gentzen clauses. As being implications Gentzen clauses can be passed over directly to tableau (top-down, backward) proof calculi. They need not be transformed into space-consuming normal forms before a theorem prover can deal with them. Skolemization and other normalizations often destroy the original structure (and easy-to-understand meaning) of conjectures so that user interaction with a prover that processes normalized formulas is mostly impossible.

While conjectures may be presented as arbitrary Gentzen clauses, the axioms of a standard specification are restricted to Horn and co-Horn clauses. This complies with usual syntactic schemas adopted by functional, logic and even state- or object-oriented programs. Semantically, the restriction to Horn and co-Horn clauses guarantees the existence of initial and final models and thus of concrete implementations. These models also entail a number of “meta-theorems” that lead to proof rules, which the theorem prover can apply automatically (see Section 10.3).

Definition 10.2.4 **A specification** $SP = (\Sigma, AX)$ consists of a standard signature Σ and a set AX of Horn and co-Horn clauses over Σ, called the **axioms** of SP. SP is a **standard specification** if there is a chain of specifications

$$SP_1 = (\Sigma_1, AX_1) \subseteq SP_2 = (\Sigma_2, AX_2) \subseteq \cdots \subseteq SP_5 = (\Sigma_5, AX_5) = SP$$

[3] The lack of explicit negation is compensated by the fact that predicates often induce axioms for their complements by simply negating the axioms (see Sections 10.3 and 10.4).

that satisfy the following conditions:

(1)	*SP_1 is the visible level*
	SP_1 consists of all visible sorts, visible constructors c, visible equality predicates $\equiv$ and visible definedness predicates Def of SP, the following Horn axioms, called **constructor axioms**:[4] $c(x_1,\dots,x_{i-1},\bot,x_{i+1},\dots,x_n) \;\equiv\; \bot$ for all $1 \le i \le n$, $Def(c(x_1,\dots,x_n)) \;\Leftarrow\; Def(x_1)\wedge\cdots\wedge Def(x_n)$, a set FP of defined functions and μ-predicates and modal Horn axioms $f(t)\equiv u \Leftarrow H$ resp. $r(t)\Leftarrow H$ for FP such that (a) $var(u)\subseteq var(t,H)$, (b) t is a normal form, (c) for all visible $x\in var(t)$, $Def(x)\in H$,[5] (d) H is modal.

(2)	*SP_2 is the hidden constructor level*
	$SP_2\setminus SP_1$ consists of all hidden sorts, hidden constructors, the set FP of destructors, transitions and attributes of SP and simple and modal Horn axioms $f(t)\equiv u\Leftarrow H$ resp. $r(t)\Leftarrow H$ for FP such that (a)-(c) hold true.

(3)	*SP_3 is the hidden extension level of SP*
	$SP_3\setminus SP_2$ consists of a set FP of defined functions and μ-predicates and simple and modal Horn axioms $f(t)\equiv u\Leftarrow H$ resp. $r(t)\Leftarrow H$ for FP such that (a)-(c) hold true.

(4)	*SP_4 is the bisimulation level of SP*
	$SP_4\setminus SP_3$ consists of all bisimulation predicates $\sim$ of SP and the co-Horn axioms $x\sim y \;\Rightarrow\; f(x,z)\sim f(y,z)$ for all attributes and destructors $f: sw\to s'$ of SP, $x\sim y \;\Rightarrow\; (r(x,z) \;\Rightarrow\; r(y,z))$ for all attributes $r: sw$ of SP, $x\sim y \;\Rightarrow\; (\delta(x,z,x') \;\Rightarrow\; \exists y'(\delta(y,z,y')\wedge x'\sim y'))$ for all transitions $\delta: sws'$ of SP, $x\sim y \;\Rightarrow\; y\sim x$,[4] $x\sim_s y \;\Rightarrow\; x\equiv y$ for all visible sorts s of SP.[4]

(5)	*SP_5 is the ν-level*
	$SP_5\setminus SP_4$ consists of a set P of ν-predicates and co-Horn axioms $r(t)\Rightarrow(H\Rightarrow GS)$ for P such that (b) holds true and H is a goal over Σ_4.

SP is **visible** if Σ is visible (cf. 10.2.1). SP has a visible subspecification **visSP** that consists of $vis\Sigma$ and the (Horn) axioms for $vis\Sigma$. ❑

The requirement that all axioms are modal is crucial for the class of SP-models to be closed under quotients by bisimulations (cf. Thm. 10.2.8) or, in other words, for SP to be **behaviorally consistent** (cf. [BHW 95]).

[4] These axioms are omitted in examples.
[5] Def-atoms in the premise of an axiom are omitted in examples.

Modality involves a constraint on hidden equations in Horn axiom premises (see 10.2.3) that is less restrictive than the condition required by other behavioral-specification approaches, which forbid all hidden equations in axiom premises (cf., e.g., [BW 82], Cor. 4; [Wir 90], Thm. 5.4.5; [BHW 95], Ex. 3.24). Since we will reason about SP with respect to the final model, which is the quotient by bisimilarity of the initial model (see Section 3), behavioral consistency is crucial.

The hierarchy of a standard specification SP ensures that ν-predicates are defined in terms of μ-predicates, but not vice versa. The hierarchy will be reflected by the construction of the standard model of SP: the initial model of SP's hidden extension level is extended by greatest solutions of the co-Horn axioms AX, which exists because the corresponding AX-*step function* is monotonic (cf. Thm. 10.3.1). The construction resembles the construction of *stable* models of stratified logic programs (cf. [ABW 88]).

The restriction on the visible level of SP still allows us to specify any partial function that is continuous in the sense of recursive function theory. Though the constructor axioms imply that constructors are strict, i.e. preserve bottom constants, defined functions may be non-strict, i.e. map $\perp$ to non-bottom normal forms.

Example 10.2.5 Let NAT and LIST be visible specifications of natural number arithmetic and finite lists of elements of a sort *entry*. We extend NAT+LIST by a specification of **infinite streams**:

INFSEQ = NAT + LIST +

`hidsorts`	$stream$	
`consts`	$\_\&\_ : entry \times stream \to stream$	
	$01 :\to stream$	
	$nats : nat \to stream$	
	$odds : stream \to stream$	
	$zip : stream \times stream \to stream$	
	$map : (entry \to entry) \times stream \to stream$	
`defuns`	$head : stream \to entry$	
	$tail, evens : stream \to stream$	
	$10 :\to stream$	
	$\_\#\_ : list \times stream \to stream$	
`preds`	$\_ \sim \_ : stream \times stream$	
	$exists, forall, forallExists : (entry \to bool) \times stream$	
`vars`	$n : nat \quad x, y : entry \quad L : list \quad s, s' : stream$	
	$f : entry \to entry \quad g : entry \to bool$	
`Horn axioms`	$head(x\&s) \equiv x$	
	$tail(x\&s) \equiv s$	
	$head(01) \equiv 0$	
	$tail(01) \equiv 10$	
	$head(nats(n)) \equiv n$	$head(zip(s, s')) \equiv head(s)$
	$tail(nats(n)) \equiv nat(n + 1)$	$tail(zip(s, s')) \equiv zip(s', tail(s))$
	$head(odds(s)) \equiv head(s)$	$head(map(f, s)) \equiv f(s)$
	$tail(odds(s)) \equiv odds(tail(tail(s))$	$tail(map(f, s)) \equiv map(f, tail(s))$
	$evens(s) \equiv odds(tail(s))$	
	$10 \equiv 1\&01$	
	$nil\#s \equiv s$	
	$(x :: L)\#s \equiv x\&(L\#s)$	
	$exists(g, s) \;\Leftarrow\; g(head(s)) \equiv true \vee exists(g, tail(s))$	

`co-Horn axioms`	$s \sim s' \;\Rightarrow\; head(s) \equiv head(s')$
	$s \sim s' \;\Rightarrow\; tail(s) \sim tail(s')$
	$forall(g,s) \;\Rightarrow\; (g(head(s)) \equiv true \wedge forall(g, tail(s)))$
	$forallExists(g,s) \;\Rightarrow\; (exists(g,s) \wedge forallExists(g, tail(s)))$
(A)	$fair(s) \;\Rightarrow\; forallExists(\lambda x.eq(x,0), s)$
(B)	$fair(s) \;\Rightarrow\; \exists L, s' : (0 \notin L \wedge s \sim L\#(0\&s') \wedge fair(s'))$

We expect a model that interprets INFSEQ as follows. & appends an entry to a stream. 01 and 10 denote the streams whose elements alternate between zeros and ones. $nats(n)$ generates the stream of all numbers starting from n. $odds(s)$ returns the stream of all elements of s that have odd-numbered positions in s. zip merges two streams into a single stream by alternatively appending an element of one stream to an element of the other stream. # concatenates a list and a stream into a stream. $head$, $tail$, map, $exists$ and $forall$ have the same meaning as stream functions as they have as list functions. $\sim$ is stream equality. $forallExists(g,s)$ is valid iff $g(x) = true$ for infinitely many $x \in s$. Hence $fair(s)$ holds true iff s contains infinitely many zeros. This results from Axiom A as well as from Axiom B. ❏

ν-predicates often express *safety* or *invariance* conditions, while μ-predicates often describe *liveness* or *reachability* conditions (cf., e.g., [Lar 88]). As being specified by Horn clauses a μ-predicate on infinite structures like streams or processes can often be formulated in terms of their finite prefixes, while ν-predicates mostly deal with an "infinite future" and thus cannot be reduced to approximating relations on finite structures.

Definition 10.2.6 (semantics) Let S be the set of sorts of Σ. A **Σ-structure** A consists of an S-sorted set, the **carrier** of A, also denoted by A, a function $f^A : A_w \to A_s$ for each function symbol $f : w \to s$ and a relation $r^A \subseteq A_w$ for each predicate $r : w$ of Σ.[6] Given two Σ-structures A and B, a **Σ-homomorphism** $h : A \to B$ is a homomorphism in the sense of Section 2.3 such that $h(r^A) \subseteq r^B$ for all predicates r of Σ. A and B are **Σ-isomorphic**, written: $A \cong B$, if $g \circ h = id_A$ and $h \circ g = id_B$ hold true for some Σ-homomorphism $g : B \to A$.

Let $\approx$ be an S-sorted equivalence relation on A. $\approx$ is **A-compatible** with a function symbol $f : w \to s \in \Sigma$ if for all $a, b \in A_w$, $a \approx b$ implies $f^A(a) \approx f^A(b)$. $\approx$ is **A-compatible** with a predicate $r \in \Sigma$ if for all $a \in r^A$, $a \approx b$ implies $b \in r^A$. $\approx$ is a **Σ-congruence** if $\approx$ is A-compatible with (all function symbols and predicates of) Σ. $\approx$ is a **Σ-bisimulation** if

- for all $a, b, c, a' \in A$ and $\delta \in \tau\Sigma$, $(a, b, c) \in \delta^A$ and $a \approx a'$ imply $(a', b, c') \in \delta^A$ and $c \approx c'$ for some $c' \in A$,
- for all visible $s \in S$ and $a, b \in A_s$, $a \approx b$ implies $a = b$.

If $\approx$ is A-compatible with all function symbols of Σ, then the **quotient** $B = A/\!\approx$ **of** A **by** $\approx$ is the Σ-structure that interprets $s \in S$ as the quotient set $A_s/\!\approx$ and a function symbol $f \in \Sigma$ by $f^B([a]) = [f^A(a)]$ where $[a]$ is the equivalence class of a, i.e. the set of all $b \in A$ with $a \approx b$.[7] B interprets each predicate $r \in \Sigma$ as the set of equivalence classes $[a]$ with $b \in r^A$ for some $b \approx a$.

Given a Σ-structure A and a signature morphism $\sigma : \Sigma' \to \Sigma$, the definition of the **σ-reduct** $A|_\sigma$ **of** A (see Section 2.5) is extended to predicates: for all $r \in \Sigma'$, $r^{A|_\sigma}$ is defined as $\sigma(r)^A$. If σ is an inclusion, i.e. $\Sigma' \subseteq \Sigma$, we write $A_{\Sigma'}$ instead of $A|_\sigma$ and call $A_{\Sigma'}$ the **Σ'-reduct of** A.

The interpretation of Σ-terms in a Σ-structure A depends on a **valuation** of variables in A, i.e. an S-sorted function $b : X \to A$. The unique Σ-homomorphism extending b to a function from $T_\Sigma(X)$ to A is denoted by b^*. Given that $x_1, \ldots, x_n$ are the variables of t, the function $t^A : A^n \to A$ is defined by $t^A(b(x_1), \ldots, b(x_n)) = b^*(t)$.

[6] If Σ has no predicates, a Σ-structure is a Σ-algebra (see Section 2.2).

[7] If $a = (a_1, \ldots, a_n)$, then $[a]$ stands for $([a_1], \ldots, [a_n])$.

In particular, $t^A = b^*(t)$ if t is a ground term. A is **reachable** if for all $a \in A$ there is $t \in T_\Sigma$ with $t^A = a$. Each Σ-structure A has a unique reachable substructure, denoted by **gen(A)** (see Section 2.2).

A valuation $b : X \to A$ **solves** an equation $t \equiv t'$ **in** A if $b^*(t) = b^*(t')$. b **solves** a logical atom $r(t)$ in A if $b^*(t) \in r^A$. The notion of a solution is extended from atoms and equations to complex first-order formulas as usual. If b solves a formula F in A, we write $A \models_b F$. A **satisfies** F or F is **valid** in A, written $A \models F$, if all valuations in A solve F in A.

Let $SP = (\Sigma, AX)$ be a standard specification. A Σ-structure A is an SP**-model** if A satisfies all axioms of SP. The classes of all resp. all reachable SP-models are denoted by **Mod(SP)** resp. **Gen(SP)** and called the **loose** resp. **reachable semantics** of SP. ❏

Proposition 10.2.7 Let $\Sigma' \subseteq \Sigma$, A be a reachable Σ'-structure and B be a reachable Σ-structure such that for all ground atoms p over Σ', $A \models p$ iff $B \models p$. Then $A \cong B_{\Sigma'}$. ❏

The main problem caused by the requirement that SP is behaviorally consistent (see above) comes with the set $\tau\Sigma$ of transitions (Def. 10.2.1). The point is that a bisimulation is usually not compatible with transitions, at least not with respect to their "targets" (Def. 10.2.4).[8] Fig. 13 illustrates both properties schematically. A formula ϕ that is valid in a Σ-structure A may not hold in the quotient $A/\approx$ if $\approx$ is not A-compatible with some predicate occurring in ϕ. The following theorem says that at least the class of valid *modal* formulas is closed under quotients by bisimulations. This is the reason why the axioms of a standard specification SP are supposed to be modal. The condition that SP is *coinductive* (Def. 10.3.6) will then ensure the bisimulation property of $\sim^A$ (Thm. 10.3.7).

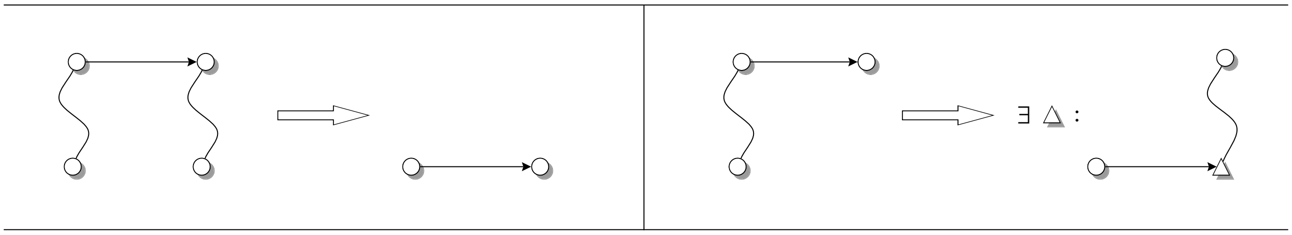

Figure 1. Congruence versus bisimulation

Theorem 10.2.8 [Pad 97a] Let Σ be a standard signature, A be a Σ-structure and $\approx$ be a Σ-bisimulation that is A-compatible with $\Sigma \setminus \tau\Sigma$. $B = A/\approx$ becomes a Σ-structure by interpreting the function symbols of Σ, all predicates $r \in \Sigma \setminus \tau\Sigma$ as usual and all $\delta \in \tau\Sigma$ as follows: For all $a, b, c \in A$,

$$([a],[b],[c]) \in \delta^B \iff_{def} \exists\, c' \approx c : (a,b,c') \in \delta^A.$$

Moreover, for all modal formulas ϕ, $A \models \phi$ implies $B \models \phi$. In particular, for all standard specifications $SP = (\Sigma, AX)$, B is an SP-model if A is an SP-model and $\sim^A$ is A-compatible with $\Sigma \setminus \tau\Sigma$. ❏

Let us now turn to the particular models induced by a standard specification.

Definition 10.2.9 Given a class $\mathcal{C}$ of Σ-structures, $I \in \mathcal{C}$ is **initial** in $\mathcal{C}$ if for all $A \in \mathcal{C}$ there is a unique Σ-homomorphism $ini^A : I \to A$. $T \in \mathcal{C}$ is **final** in $\mathcal{C}$ if for all $A \in \mathcal{C}$ there is a unique Σ-homomorphism $fin^A : A \to T$. ❏

Each two initial (resp. final) Σ-structures are Σ-isomorphic. The initial model of SP is usually presented as the quotient of T_Σ by the equivalence relation consisting of all ground equations that are derivable from the

[8] Hence most behavioral-semantics approaches did not admit transition relations.

axioms of SP via the *cut calculus*:[9]

Definition 10.2.10 Let $SP = (\Sigma, AX)$ be a standard specification. The **cut calculus for** SP consists of all Horn (!) axioms of SP, the following axioms stating that $\equiv$ is a Σ-congruence and compatible with all predicates of Σ:

$$\begin{array}{rcll}
x \equiv x & & & \\
y \equiv x & \Leftarrow & x \equiv y & \\
f(x) \equiv f(y) & \Leftarrow & x \equiv y & \text{for all function symbols } f : w \to s \text{ of } \Sigma \\
r(y) & \Leftarrow & r(x) \wedge x \equiv y & \text{for all predicates } r \text{ of } \Sigma \\
(x_1, \dots, x_n) \equiv (x_1, \dots, x_n) & \Leftarrow & x_1 \equiv y_1 \wedge \dots \wedge x_n \equiv y_n & \text{for all } n > 1,
\end{array}$$

and the following inference rules. Let p be an atom and G, H be goals.[10]

instantiation $\quad \dfrac{p \Leftarrow H}{p\sigma \Leftarrow H\sigma} \Downarrow \quad$ for all substitutions σ over Σ

cut $\quad \dfrac{p \Leftarrow G \wedge H, \quad H}{p \Leftarrow G} \Downarrow$

$\wedge$-generation $\quad \dfrac{G, \quad H}{G \wedge H} \Downarrow$

Given a formula ϕ, we write $SP \vdash_{cut} \phi$ if there is a derivation of ϕ employing only axioms and rules of the cut calculus for SP. The set of derivable equations $t \equiv t'$ induces the SP**-equivalence** relation $\equiv_{SP}$:

$$t \equiv_{SP} t' \quad \Longleftrightarrow_{def} \quad SP \vdash_{cut} t \equiv t'.$$

Given $t \in T_\Sigma$ and $u \in NF_\Sigma$, u is a **normal form of** t if t and u are SP-equivalent. SP is **complete** if each ground Σ-term has a normal form. SP is **consistent** if each two SP-equivalent ground normal forms are equal. SP is **functional** if SP is complete and consistent. If SP is functional, we denote the unique normal form of a ground Σ-term (tuple) t by $\mathbf{nf(t)}$.

SP-equivalence yields an interpretation of $\equiv$ on T_Σ. If the logical μ-predicates r of SP are interpreted accordingly, i.e.

$$r_{SP}(t) \quad \Longleftrightarrow_{def} \quad SP \vdash_{cut} r(t),$$

then T_Σ becomes a $(\Sigma \setminus \nu\Sigma)$-structure and SP-equivalence becomes a Σ-congruence. The quotient of T_Σ by $\equiv_{SP}$ is called the **initial** SP**-model** and denoted by $\mathbf{Ini(SP)}$. A first-order formula satisfied by $Ini(SP)$ is called an **initial theorem** of SP. ❑

Proposition 10.2.11 (definedness) Let SP be a functional specification.

(1) For all $t \in T_\Sigma$, $SP \vdash_{cut} Def(t)$ iff $t \not\equiv_{SP} \perp$.

(2) For all $\sigma : X \to NF_\Sigma$ and Horn axioms $f(t) \equiv u \Leftarrow H$ or $r(t) \Leftarrow H$ of SP, $SP \vdash_{cut} H\sigma$ implies that $t\sigma$ is normal. ❑

Theorem 10.2.12 Let $SP = (\Sigma, AX)$ be a standard specification. Each SP-model A with $A_{\Sigma \setminus \nu\Sigma} \cong Ini(SP)$ is an initial Σ-structure in $Mod(SP)$. ❑

[9] See Theorem 2.23 for the case of specifications whose axioms are unconditional equations.

[10] Arrows attached to rules indicate the direction(s) in which the rules provide implications. Hence rules with $\Downarrow$ are used for forward reasoning, while rules with $\Uparrow$ build up top-down proofs.

The choice of $Ini(SP)$ as the standard model of a (Horn) specification has been motivated thoroughly in the literature (cf., e.g., [MG 85], [EM 85], [Pad 88]). Initial models reduce the reasoning about data types to inductive theorem proving. Initial semantics neatly complies with functional sorts, polymorphism and parameter specifications (see Examples 10.1.1 and 10.2.5). Functional sorts like $s \to s'$ are implicilty associated with an apply-operator that extends the inductive definition of $T_\Sigma(X)$ as follows:

$$t \in T_\Sigma(X)_{s\to s'} \wedge u \in T_\Sigma(X)_s \quad \Rightarrow \quad t(u) \in T_\Sigma(X)_{s'}.$$

As, for each Σ-structure A, $A_{s_1\dots s_n}$ is defined as the product $A_{s_1} \times \dots \times A_{s_n}$, $A_{s\to s'}$ is defined as the function space $[A_s \to A_{s'}]$. Thm. 10.2.12 remains valid if $Mod(SP)$ is replaced by $Gen(SP)$ [MTW 88].

Definition 10.2.13 (Gentzen clause theorems) Let $SP = (\Sigma, AX)$ be a standard specification. A Horn clause $p \Leftarrow H$ is a **deductive theorem** of SP if there is a skolemization ρ with $dom(\rho) = var(H)$ (Def. 10.2.2) such that $(\Sigma, AX \cup H\rho) \vdash_{cut} p\rho$.

A Gentzen clause $GS \Leftarrow HS$ is an **inductive theorem** of SP if for all goals H of HS and $\sigma \in T_\Sigma^X$, $SP \vdash_{cut} H\sigma$ implies $SP \vdash_{cut} G\tau$ for some goal G of GS and $\tau \in T_\Sigma^X$ such that $\tau_V = \sigma_V$ where V is the set of free variables of $GS \Leftarrow HS$.[11]

DTh(SP) and **ITh(SP)** denote the sets of deductive and inductive theorems of SP, respectively. ❑

The initial SP-model interprets μ-predicates as the **least relations** satisfying their axioms. This follows from *Kleene's fixpoint theorem*, which is also used for defining the semantics of pure logic programs. *Tarski's fixpoint theorem* imposes weaker conditions on the axioms for a predicate and will be employed for interpreting ν-predicates. Let us recapitulate (the set-theoretical versions of) both theorems.

Theorem 10.2.14 (fixpoint theorems) (cf., e.g., [LNS 82]) Let A be a (sorted) set and $\Phi : \wp(A) \to \wp(A)$ be a monotonic function (with respect to set inclusion). $B \subseteq A$ is a **fixpoint** of Φ if $\Phi(B) = B$. $Phi^\infty = \cup_{i\in\mathbb{N}}\Phi^i(\emptyset)$ and $\Phi_\infty = \cap_{i\in\mathbb{N}}\Phi^i(A)$ are called the **Kleene closures of** Φ.

Tarski's theorem $\mu\Phi = \cap\{B \subseteq A \mid \Phi(B) \subseteq B\}$ is the least fixpoint of Φ and a superset of Φ^∞.
$\nu\Phi = \cup\{B \subseteq A \mid B \subseteq \Phi(B)\}$ is the greatest fixpoint of Φ and a subset of Φ_∞.

Φ is **upward continuous** if for all increasing chains $B_1 \subseteq B_2 \subseteq B_3 \subseteq \dots$ of subsets of A, $\Phi(\cup_{i\in\mathbb{N}}B_i)$ is a subset of $\cup_{i\in\mathbb{N}}\Phi(B_i)$. Φ is **downward continuous** if for all decreasing chains $B_1 \supseteq B_2 \supseteq B_3 \supseteq \dots$ of subsets of A, $\cap_{i\in\mathbb{N}}\Phi(B_i)$ is a subset of $\Phi(\cap_{i\in\mathbb{N}}B_i)$.

Kleene's theorem Φ upward continuous $\Rightarrow$ $\Phi(\Phi^\infty) \subseteq \Phi^\infty$ $\Rightarrow$ $\mu\Phi = \Phi^\infty$.
Φ downward continuous $\Rightarrow$ $\Phi_\infty \subseteq \Phi(\Phi_\infty)$ $\Rightarrow$ $\nu\Phi = \Phi_\infty$. ❑

Definition 10.2.15 (relational version) Let $SP = (\Sigma, AX)$ be a specification such that AX consists of simple Horn axioms for functions and logical predicates of Σ. Suppose that all axioms have been **flattened**, i.e. for each equation $t \equiv u$ and each logical atom $r(v)$ occurring in an axiom, u, v are normal forms and either t is a normal form, too, or $t = f(t')$ for some defined function f and normal form t'. Obviously, each Horn axiom for a function or logical predicate can be flattened such that the initial model of the flattened specification is isomorphic to the original one.

The **relational version** of SP, $rel(SP)$, is the specification obtained from (a flattened version of) SP by replacing each defined function $f : w \to s \in \Sigma$ by the **input-output predicate** $r_f : ws$ of f and each equation $f(t) \equiv u$ occurring in AX by the atom $r_f(t, u)$. ❑

[11] For equational Gentzen clauses without quantifiers, this characterization of inductive theorems agrees with Def. 9.27.

Theorem (see 10.5.5) Let $SP = (\Sigma, AX)$ be the hidden extension level of a functional specification and $rel(SP) = (\Sigma', AX')$. Σ' consists of the sorts, constructors, bottom constants and predicates of Σ. Moreover, for all defined functions $f \in \Sigma$, logical predicates $r \in \Sigma$ and $t, u \in T_{\Sigma'}$,

$$\begin{array}{rcl} Ini(SP) \models f(t) \equiv u & \iff & Ini(rel(SP)) \models r_f(t,u), \\ Ini(SP) \models r(t) & \iff & Ini(rel(SP)) \models r(t). \ ❑ \end{array}$$

Each carrier element of $Ini(rel(SP))$ is an equivalence class $[t]$ consisting of constructor terms. Since SP is functional, $[t]$ is either a singleton and t is a normal form or all terms of $[t]$ contain $\perp$.

Let $\mathcal{C}$ be the class of Σ-structures that interpret sorts, function symbols and equality predicates the same as $Ini(SP)$ does. Since the cut calculus is sound with respect to $\mathcal{C}$, $SP \vdash_{cut} r(t)$ implies $\mathcal{C} \models r(t)$ for all ground atoms $r(t)$. In other words, $Ini(SP)$ interprets all μ-predicates as the least relations satisfying their axioms. This yields the correctness of *fixpoint induction* for proving initial SP-theorems of the form $\phi \Leftarrow r(x)$ (see Section 3). We may also prove conjectures of the form $\phi \Leftarrow f(x) \equiv y$ by *fixpoint induction* if SP is functional because then, by Thm. 10.5.5, a proof that $\phi \Leftarrow r_f(x,y)$ is an initial theorem of $rel(SP)$ is also a proof that $\phi \Leftarrow f(x) \equiv y$ is an initial theorem of SP.

It remains to interpret the ν-predicates of a standard specification. This will be done in Section 10.3.

Theorem 10.2.16 (validity and provability) Let $SP = (\Sigma, AX)$ be an inhabited standard specification, $\sigma : \Sigma' \to \Sigma$ be a signature morphism and $\approx$ be a Σ-bisimulation that is $Ini(SP)$-compatible with $\Sigma \setminus \tau\Sigma$. For all $t, t' \in T_{\Sigma'}$,

$$[t] \approx_\sigma [t'] \quad \Longleftrightarrow_{def} \quad [\sigma(t)] \approx [\sigma(t')]$$

defines a Σ'-bisimulation on $Ini(SP)|_\sigma$ such that $gen(Ini(SP)|_\sigma)/\approx_\sigma$ agrees with $gen(Ini(SP)/\approx|_\sigma)$. Given a predicate $r \in \Sigma'$, $\approx_\sigma$ is compatible with r if $\approx$ is compatible $\sigma(r)$ (see 10.2.6). Let Σ_σ consist of Σ and a **representation predicate** $rep_s : \sigma(s)$ for each sort (tuple) $s \in \Sigma'$. Let AX_σ consist of AX and a Horn axiom

$$rep(\sigma(f)(x_1, \ldots, x_n)) \quad \Leftarrow \quad rep(x_1) \wedge \cdots \wedge rep(x_n)$$

for each function symbol $f \in \Sigma'$. Let $SP_\sigma = (\Sigma_\sigma, AX_\sigma)$.

(1) For all Horn clauses $C = (p \Leftarrow H)$ over Σ,

$$\begin{array}{rcl} Mod(SP) \models C & \iff & C \in DTh(SP), \\ Gen(SP) \models C & \iff & \forall \sigma \in T_\Sigma^X : (\Sigma, AX \cup H\sigma) \vdash_{cut} p\sigma. \end{array}$$

(2) For all Gentzen clauses C over Σ,

$$Ini(SP) \models C \iff C \in ITh(SP).$$

(3) For all Gentzen clauses C over Σ',

$$\begin{array}{rcl} Ini(SP)|_\sigma \models C & \iff & \sigma(C) \in ITh(SP), \\ gen(Ini(SP)|_\sigma) \models C(x) & \iff & Ini(SP_\sigma) \models \sigma(C)(x) \Leftarrow rep(x), \\ gen(Ini(SP)/\approx|_\sigma) \models C(x) & \iff & Ini(SP_\sigma)/\approx \models \sigma(C)(x) \Leftarrow rep(x) \end{array}$$

where x is a tuple consisting of all variables of C. ❑

From now on, we assume that the signature of a non-parameter specification is inhabited. This is crucial for the $\Leftarrow$-directions of (1), which establish soundness properties of the cut calculus. Without the assumption of inhabitedness the free variables of axioms and derived goals had to be universally quantified as it in the equational calculus of Section 2.8. Otherwise one may infer invalid goals from valid ones whenever empty sorts are involved. For instance, a supposedly invalid equation $a \equiv b$ were derivable from the axioms $F(x) \equiv a$ and $F(x) \equiv b$ if $sort(x)$ is empty. Apart from the assumption of inhabitedness, *equational* deductive theorems of a standard specification with only *equational* axioms are exactly those formulas that are derivable via the equational calculus of Section 2.8.

(1) and (2) imply $Ini(SP) \models g$ *iff* $Gen(SP) \models g$ for all goals g and $Ini(SP) \models g$ *iff* $Mod(SP) \models g$ for all *ground* goals g. For *conditional* Horn clauses C, however, the hierarchy is strict: C may hold in $Ini(SP)$, but not in $Gen(SP)$, and C may hold in $Gen(SP)$, but not in $Mod(SP)$. Invariance properties of the form $r(f(x)) \Leftarrow r(x))$, stating that f preserves r, are usually valid only in $Ini(SP)$, in particular if r is defined inductively on the structure of normal forms (see, e.g., [Pad 88], Ex. 4.3.4). This is one of the reasons for choosing initial semantics as *the* semantics and the inductive theory as *the* theory of a (visible) data type.

Functionality is defined for any standard specification SP, but actually concerns only its visible subspecification because our standard model of SP will be a final one where $\equiv_{SP}$ defines only the equality of visible data. Completeness seems to forbid the specification of partial functions. For instance, the classical theory of partial-recursive functions (REC; see Section 10.1) admits *divergence*, while completeness demands convergence into normal forms, at least into bottom constants. However, when it comes to proof methods, REC hides divergence by modelling a partial function as the supremum of its *total* approximations. But approximations and their suprema may be specified as *total* defined functions, i.e. within a complete specification (see, e.g., [Pad 91], [Pad 92]). Alternatively, certain partial functions may be modelled as hidden constructors along with a ν-predicate expressing divergence (see [Gor 95]).

10.3 Final semantics and bisimilarity

Final semantics was introduced for modelling **permutative types** such as finite sets, finite bags (multisets) and functions with a finite domain (stores, arrays, indexed lists) (see, e.g., [GGM 76], [Wan 79], [Kam 83]). These types are still constructor-based, but *constructor equations* are needed to axiomatize data equality. Hence specifications of permutative types are complete, but not consistent. From a model-theoretic viewpoint, initial semantics is sufficient for handling permutative types. Constructor equations are Horn axioms and thus we obtain an initial model as in the case of standard specifications where the constructor equations are fixed (see 10.2.4(1)). From a proof-theoretic viewpoint, however, initial semantics is less appropriate. Efficient resolution- and rewriting-oriented proof methods treat constructor equations CE on a lower level than other axioms (see, e.g., [Plo 72], [Sti 85], [JKi 86]). Normal forms are replaced by equivalence classes of normal forms modulo the equivalence relation $\equiv_{CE}$ induced by CE. Resolution and rewriting modulo $\equiv_{CE}$ work well if CE is restricted to particular axioms such as associativity, commutativity, idempotence, etc. Otherwise corresponding proof rules are difficult to handle.

With respect to final semantics, constructor equations become *theorems* that can be derived, e.g., by **context induction** (see [Hen 91]). Studies in category theory and modal logic on coalgebras, coinduction and greatest fixpoints suggested both subsuming permutative types under hidden types and adopting final semantics for **object-oriented** as well as **infinite types** such as streams and processes (see, e.g., [AW 89], [GD 94], [Rei 95], [Rut 96], [GM 97], [JR 97]). Actually, even infinite types can be presented as *functional*, i.e. complete and consistent specifications (Def. 10.2.10). Consistency is not restrictive here because hidden data equality is defined in terms of bisimulation and not equality predicates. Completeness, however, seems to be restrictive. How can uncountably many streams be represented by countably many normal forms? The answer is that not all elements of a carrier set need to be covered by normal forms as long as the theory of a specification SP is preserved whenever further normal forms are added to SP. Since a theorem about a hidden carrier cannot be proved by structural induction (because the structure is hidden), it will mostly be valid in extensions of SP by further normal forms. For instance, many streams are not represented as normal forms of INFSEQ, while a single stream may have several normal forms (see Ex. 10.2.5). Whereas *visible* data are considered to be equal if they have the same normal forms, bisimulation predicates take over the rôle of identifying terms representing the same *hidden* objects.

Besides separating visible from hidden sorts we have distinguished μ- from ν-predicates. Initial models interpret μ-predicates as least relations satisfying their (Horn) axioms. Dually, ν-predicates are interpreted as greatest relations satisfying their (co-Horn) axioms. Horn axioms for a visible symbol f usually entail an inductive definition of f in the initial model. Category theory set up the duality between inductively defined algebras and coinductively defined coalgebras (see, e.g., [AM 82], [Gor 95], [Rei 95], [JR 97]). Initial algebras and final coalgebras are isomorphisms built up from constructors in one direction and destructors in the other. For instance, the final coalgebra of infinite streams (see Ex. 10.2.5) splits into the constructor &, the attribute *head* and the destructor *tail*. We keep the duality in mind, but follow more algebraic lines by constructing final models as quotients of initial ones.

Theorem 10.3.1 (fixpoint semantics, bisimilarity) ?? Let $SP = (\Sigma, AX)$ be a functional specification and $\mathcal{C}$ and $\mathcal{D}$ be the classes of Σ-structures that interpret sorts, function symbols and equality predicates the same as $Ini(SP)$ does $\Sigma \setminus \nu\Sigma$ the same as $Ini(SP)$ does. The **step function** $\Psi : \mathcal{C} \to \mathcal{C}$ is defined as follows: For all $r \in \nu\Sigma$, $b : X \to A$ and $A \in \mathcal{C}$,

$$b(x) \in r^{\Psi(A)} \quad \Longleftrightarrow_{def} \quad \forall (r(x) \Rightarrow C) \in AX : A \models_b C.$$

Ψ is monotonic with respect to set inclusion on $\mathcal{C}$. Hence by Tarski's fixpoint theorem, Ψ has the greatest fixpoint $\nu\Psi = \cup\{A \in \mathcal{C} \mid A \subseteq \Psi(A)\}$.

SP is called **downward continuous** if Ψ is downward continuous. In this case Kleene's fixpoint theorem implies that $\nu\Psi$ agrees with $\Psi_\infty = \cap_{i\in\mathbb{N}}\Psi^i(\top)$ where $\top$ is the greatest element of $\mathcal{C}$.

The interpretation of $\sim$ in $\nu\Psi$ is called SP**-bisimilarity** and denoted by $\sim_{SP}$. SP is **behaviorally congruent** if $\sim_{SP}$ is $\nu\Psi$-compatible with $\Sigma \setminus \tau\Sigma$. ❑

SP-bisimilarity is an equivalence relation. For all visible sorts s, $\sim_{SP,s}$ coincides with the equality on $Ini(SP)_s$.

SP is downward continuous if for all co-Horn axioms $r(x) \Rightarrow (H \Rightarrow GS)$ of SP, the goal set GS does not involve existential quantifiers. Otherwise the downward continuity of SP is not guaranteed. If, for instance, AX includes a co-Horn clause of the form $r(x) \Rightarrow \exists y q(x,y)$, then SP is downward continuous only if $r^{\cap_i \Psi(B_i)}$ is a subset of $r^{\Psi(\cap_i B_i)}$. But this means that $\forall i \exists y : q^{B_i}(x,y)$ implies $\exists y \forall i : q^{B_i}(x,y)$, which does *not* always hold true.

The μ-calculus of modal logic (see [Sti 92]) provides the idea for a condition on co-Horn axioms that guarantees the continuity of Ψ. Given a predicate r and a modal formula $F(r)$, $\mu r.F(r)$ and $\nu r.F(r)$ denote the least resp. greatest interpretation of r that satisfies $r(x) \Leftrightarrow F(r)$. First a certain *polarity condition* on the structure of F ensures that the step function induced by F is monotonic and thus Tarski's fixpoint theorem implies that there least and greatest interpretations do exist. F is even (upward and downward) continuous if the transition system that underlies any modal-formula interpretation is *finitely branching*. This property can be generalized to a criterion for the downward continuity of SP as defined in Theorem 10.3.1 and thus for the iterative constructability of $\nu\Psi$ even if co-Horn axioms involve existential quantifiers. We do not elaborate this criterion here, but note that it holds true for all sample specifications presented in this chapter.

Similarly to the function approximations used in recursive-function theory, one may introduce a μ-predicate $r' : nat \times w$ that approximates a ν-predicate $r : w$. Then SP is turned into its **Horn clause completion** SP_∞ by specifying each $r \in \nu\Sigma$ and its approximation r' in terms of generalized Horn axioms whose premises may contain negated μ-predicates and universal quantifiers:

$$\begin{array}{rcl} r(x) & \Leftarrow & \forall i : r'(i,x) \\ r'(0,x) & & \\ r'(i+1,x) & \Leftarrow & \forall Y(\neg H \vee \exists X_1 G_1 \vee \cdots \vee \exists X_n G_n)[q'(i,u)/q(u) \mid q : v \in \nu\Sigma,\ u \in T_{\Sigma,v}(X)] \end{array}$$

where $r(x) \Rightarrow (H \Rightarrow (\exists X_1 G_1 \vee \cdots \vee \exists X_n G_n))$ is an axiom of SP and $Y = var(H, G_1, \ldots, G_n) \setminus X_1 \setminus \cdots \setminus X_n \setminus \{x\}$. For instance, axioms of INFSEQ$_\infty$ for $fair$ and $fair'$ (see Ex. 10.2.5) read as follows:

$$\begin{array}{rcl} fair(s) & \Leftarrow & \forall i : fair'(i,s) \\ fair'(0,s) & & \\ fair'(i+1,s) & \Leftarrow & 0 \notin L \wedge s \sim_i L\#(0\&s') \wedge fair'(i,s'). \end{array}$$

$fair'(i,s)$ is an initial theorem of INFSEQ$_\infty$ iff, intuitively, s contains at least i zeros.

A proof by induction on $i \in \mathbb{N}$ shows that for all $r : w \in \nu\Sigma$ and $t \in T_{\Sigma,w}$, $\Psi^i(\top)$ satisfies $r(t)$ iff $Ini(SP_\infty)$ satisfies $r'(i,t)$. Hence for all ground atoms p, the Kleene closure Ψ_∞ satisfies p iff $Ini(SP_\infty)$ satisfies p, and we conclude from Theorem 10.3.1 and Prop. 10.2.7 that the greatest fixpoint $\nu\Psi$ of Ψ is isomorphic to $Ini(SP_\infty)$ if SP is downward continuous.

Due to the axioms of SP_∞, proving statements about $\nu\Sigma$ in terms of SP_∞ usually amounts to inducing on the nat-argument of approximating predicates. Such proofs can often be replaced by simpler ones using **coinduction** presented in Section 10.7. At least, it is up to future work to find out the sort of valid statements, which can be proved easier by employing SP_∞.

Definition 10.3.2 (final semantics) Let SP be a functional and behaviorally congruent specification and $\nu\Psi$ be defined as in Theorem 10.3.1. The **final SP-model Fin(SP)** is the quotient of $\nu\Psi$ by $\sim_{SP}$ that interprets the predicates of Σ as follows:

- For all $r \in \mu\Sigma \setminus \tau\Sigma \cup \nu\Sigma$, $\quad [a] \in r^{Fin} \iff_{def} a \in r^{\nu\Psi}$.
- For all $\delta \in \tau\Sigma$, $\quad ([a],[b],[c]) \in \delta^{Fin} \iff_{def} \exists\, c' \sim_{SP} c : (a,b,c') \in \delta^{\nu\Psi}$.

A first-order formula satisfied by $Fin(SP)$ is called a **final theorem** of SP. ❑

By Theorem 10.2.8, the final SP-model is an SP-model. $Fin(SP)$ coincides with $Ini(SP)$ if SP has neither hidden sorts nor ν-predicates.

Example 10.3.3 Note that $\sim_{SP,s}$ is the entire product $Ini(SP)^2_s$ if there are no s-transitions and all s-destructors are functions into hidden sorts. To illustrate this consider the following standard specification of integer numbers:

INT

`hidsorts`	int
`consts`	$0 :\to int$
	$pred, succ : int \to int$
	$\_+\_ : int \times int \to int$
	$\_-\_ : int \times int \to int$

`defuns`	$\_+1 : int \rightarrow int$	
	$\_-1 : int \rightarrow int$	
`preds`	$is0 : int$	
	$\_\sim\_ : int \times int$	
`vars`	$x, y : int$	
`Horn axioms`	$0+1 \equiv succ(0)$	
	$0-1 \equiv pred(0)$	
	$succ(x)+1 \equiv succ(x+1)$	$(x+y)+1 \equiv ((x+1)+y)$
	$succ(x)-1 \equiv x$	$(x+y)-1 \equiv ((x-1)+y)$
	$pred(x)+1 \equiv x$	$(x-y)+1 \equiv ((x+1)-y)$
	$pred(x)-1 \equiv pred(x-1)$	$(x-y)-1 \equiv ((x-1)-y)$
	$is0(0)$	
`co-Horn axioms`	$x \sim y \;\Rightarrow\; x+1 \sim y+1$	
	$x \sim y \;\Rightarrow\; x-1 \sim y-1$	
	$x \sim y \;\Rightarrow\; (is0(x) \Rightarrow is0(y))$	

The final INT-model $Fin(\mathrm{INT})$ (see below) is isomorphic to $\mathbb{Z}$. For instance, $Fin(\mathrm{INT})$ satisfies the equations $succ(pred(x)) \equiv x$ and $pred(succ(x)) \equiv x$. INT-bisimilarity would identify *all* ground INT-terms if the last axiom for $\sim$ were omitted. ❑

Suppose that SP is downward continuous. Then $\nu\Psi$ is isomorphic to the initial model of the Horn clause completion of SP (see above). Consequently, $Fin(SP) = \nu\Psi/\sim_{SP} \;\cong\; Ini(SP_\infty)/\sim_{SP}$. This means, for instance, that $Fin(SP)$ satisfies $r(t)$ with $r \in \nu\Sigma$ if $\forall i : r'(i,t)$ is an inductive theorem of SP_∞. Proving $\forall i : r'(i,t)$ by induction on i means to induce on the length of a (cut calculus) derivation of $r'(i,t)$ insofar as induction steps from i to $i+1$ are (forward) applications of axioms for r' or other approximating predicates.

If SP has no transitions, then SP-bisimilarity can be characterized as *contextual* equivalence. Given a hidden sort s, an **s-context** is a visible term or atom consisting of visible symbols, destructors and a single s-sorted variable. Based on the set Con of all contexts, the **contextual SP-equivalence** relation $\approx_{SP}$ is defined as follows:

$$[t] \approx_{SP} [t'] \quad \Longleftrightarrow_{def} \quad \forall c \in Con_{sort(t)}\; c(t) \equiv_{SP} c(t') \text{ resp. } SP \vdash_{cut} c(t) \Leftrightarrow c(t')$$

(see Section 5.5). For instance, it is easy to see that contextual INFSEQ-equivalence coincides with INFSEQ-bisimilarity (Ex. 10.2.5).

Definition 10.3.4 (hierarchical models) Let Σ and Σ' be standard signatures with $\Sigma' \subseteq \Sigma$. Let A and B be reachable Σ'- resp. Σ-structures (see 10.2.6).

B is **complete wrt** A if $B_{\Sigma'} \cong gen(B_{\Sigma'})$ or, equivalently, for all sorts $s \in \Sigma'$ and $t \in T_{\Sigma,s}$ there is $t \in T_{\Sigma'}$ such that B satisfies $t \equiv t'$. B is **consistent wrt** A if $gen(B_{\Sigma'}) \cong A$ or, equivalently, for all ground Σ'-atoms p, $B \models p$ implies $A \models p$. Given a standard specification SP with signature Σ, an SP-model B is **hierarchical over** A if B is complete and consistent wrt A (and thus $B_{\Sigma'} \cong A$) and if for all logical μ-predicates $r \in \Sigma \setminus \Sigma'$ and $t \in T_\Sigma$, $B \models r(t)$ implies $Ini(SP) \models r(t)$. ❑

Note that SP is functional (see 10.2.10) iff $Ini(SP)$ is hierarchical over $Ini(SP')$ where SP' consists of the constructors, bottom constants, definedness predicates and the axioms of SP. Since visible Σ-normal forms are $vis\Sigma$-terms, all reachable SP-models are complete wrt $Ini(visSP)$ whenever SP is complete. We summarize the main properties of the final SP-model:

Theorem 10.3.5 Let SP be a functional and behaviorally congruent specification.

(1) For all μ-predicates $r : w$ of SP, $r^{Fin(SP)}$ is the least subset of $Fin(SP)_w$ that satisfies the Horn axioms of SP.

(2) For all defined functions $f : w \to s$ of SP, the input-output predicate $r_f : ws$ of f is the least subset of $Fin(SP)_{ws}$ that satisfies the Horn axioms of the relational version of SP.

(3) For all ν-predicates $r : w$ of SP, $r^{Fin(SP)}$ is the greatest subset of $Fin(SP)_w$ that satisfies the co-Horn axioms of SP.

(4)) If SP is downward continuous (cf. 10.3.1), then $Fin(SP)$ and $Ini(SP_\infty)/{\sim_{SP}}$ are isomorphic.

(5) $Fin(SP)$ is final in the class of hierarchical SP-models over $Ini(visSP)$ [Pad 97a]. ❑

(1)-(3) imply that the fixpoint rules presented in Section 10.7 are sound. [Pad 97a] provides (almost) syntactical conditions on Horn axioms for hidden symbols of SP that guarantee behavioral congruence:

Definition 10.3.6 (coinductivity) Given hidden (empty or singleton) term tuples t, u and visible term tuples a, b, an atom $p(t,a,b,u)$ is **oriented** if t and a are normal forms and $p(t,a,b,u)$ has one of the following forms:

(1) $\delta(t,a,b,u)$ where $\delta \in \tau\Sigma$,

(2) $r(t,a,b)$ where $r \in \mu\Sigma \setminus \tau\Sigma$ is hidden,

(3) $f(t,a) \equiv b$ where f is a hidden defined function f with visible range sort,

(4) $f(t,a) \equiv u$ where f is a hidden defined function f with hidden range sort.

We identify p with δ, r or f, respectively. A Horn clause

$$p(t_0,a_0,b_0,u_0) \quad \Leftarrow \quad G_0 \wedge p_1(t_1,a_1,b_1,u_1) \wedge G_1 \wedge \cdots \wedge p_n(t_n,a_n,b_n,u_n) \wedge G_n \qquad (*)$$

is **deterministic** if for all $0 \le i \le n$, G_i is a goal over $vis\Sigma$, $p_i(t_i,a_i,b_i,u_i)$ is oriented, $b_i, u_i \in NF_{vis\Sigma}(X)$[12] if $i > 0$, $var(t_i,a_i) \subseteq V_{i-1}$ and $var(b_0,u_0) \subseteq V_n$ where $V_0 = var(t_0,a_0,G_0)$ and $V_i = V_{i-1} \uplus var(b_i,u_i) \cup var(G_i)$. If, in addition,

(5) $t_0, a_0 \in NF_{vis\Sigma}(X)$ and $b_0, u_0 \in NF_\Sigma(X)$ **or**

(6) p_0 is a destructor or transition, $a_0 \in NF_{vis\Sigma}(X)$ and there are $t \in NF_{vis\Sigma}(X)$, $c, d \in NF_\Sigma(X)$ and a term v over $vis\Sigma \cup DF$ such that $c \notin X$, $t_0 = c(t)$ and $(b_0,u_0) = d(v(t))$,

then the clause is called **coinductive**.

SP is **coinductive** if all axioms for hidden defined functions or μ-predicates of SP are coinductive. ❑

Coinductive clauses are modal (Def. 10.2.3). With respect to destructors, coinductivity was inspired by *coinductive coalgebra definitions* (see [Rei 95], [JR 97]). With respect to transitions, it generalizes typical formats of transition system specifications used in process logics (see [Plo 83], [GV 92]).

a and b comprise all visible arguments of an oriented atom $p(t,a,b,u)$. a collects the "input", b the "output" arguments. In Cases (3) and (4), the distinction between input and output is fixed. In Cases (1) and (2), it is determined by the requirement that the axioms for r are coinductive. Intuitively, $(*)$ is deterministic if all variables "flow" only from left to right, i.e. from t_0, a_0, G_0 to t_i, a_i, from b_j, u_j, G_j to t_{j+k}, a_{j+k}, $k > 0$, or right to left from b_m, u_m, G_m to b_0, u_0.

Theorem 10.3.7 [Pad 97a] A functional and coinductive specification SP is behaviorally congruent if SP is downward continuous. ❑

[12] The only *hidden* normal forms of $NF_{vis\Sigma}(X)$ are variables!

Example 10.3.8 INFSEQ (see Ex. 10.2.5) is coinductive. Suppose we want to define stream comprehension analogously to list comprehension by the following axioms (see *filter* in Ex. 10.1.1):

$$head(filter(g,s)) \equiv head(s) \quad \Leftarrow \quad g(head(s)) \equiv true \tag{1}$$
$$head(filter(g,s)) \equiv head(filter(g,tail(s))) \quad \Leftarrow \quad g(head(s)) \equiv false. \tag{2}$$

(2) is not coinductive! In terms of the clause in Def. 10.3.6, $t_0 = filter(g,s)$, $a_0 = u_0 = \varepsilon$ and $b_0 = head(filter(g,tail(s)))$. 10.3.6(5) is violated because t_0 is not a variable. 10.3.6(6) does not hold because the occurrence of the defined function *head* in b_0 *precedes* the constructor *filter*. In fact, INFSEQ-bisimilarity cannot be compatible with *filter* already for intuitive reasons: *filter* may return finite streams, while the destructors *head* and *tail* only allow us to specify infinite streams. *filter* can only be part of a specification STREAM that comprises both finite and infinite streams (see [Pad 97a]). STREAM replaces the functional destructors *head* and *tail* by a transition $\longrightarrow$: $stream \times nat \times stream$ and a relational destructor $empty : stream$. ❑

Similarly, CCS-like process types can be specified coinductively in a quite straightforward way.

For the rest of this section suppose that $SP = (\Sigma, AX)$ is functional and behaviorally congruent. Hence SP has all the—proof-theoretically significant—properties stated in Theorem 10.3.5. As a first consequence we obtain a couple of initial or final "meta-theorems" representing background knowledge about SP.

Lemma 10.3.9 Let r be a logical μ-predicate and f be a defined function of SP, $AX_r = \{r(t_i) \Leftarrow H_i \mid 1 \leq i \leq n\}$ and $AX_f = \{f(t_i) \equiv u_i \Leftarrow H_i \mid 1 \leq i \leq n\}$ be the sets of axioms of SP for r resp. f, $Y_i = var(r(t_i) \Leftarrow H_i)$ resp. $Y_i = var(f(t_i) \equiv u_i \Leftarrow H_i)$ and x, y be (tuples of) variables that do not occur in Y_i. The formulas

$$r(x) \iff \exists Y_1(x \equiv t_1 \wedge H_1) \vee \cdots \vee \exists Y_n(x \equiv t_n \wedge H_n),$$
$$f(x) \equiv y \iff \exists Y_1(x \equiv t_1 \wedge y \equiv u_1 \wedge H_1) \vee \cdots \vee \exists Y_n(x \equiv t_n \wedge y \equiv u_n \wedge H_n),$$

called the **iff-completions** of r and f, respectively, are both initial and final theorems of SP. ❑

For instance,

$$sorted(L) \iff L \equiv nil \ \vee \ \exists\{x\}(L \equiv [x]) \ \vee \ \exists\{x,y,L'\}(L \equiv x :: y :: L' \wedge x \leq y \wedge sorted(y :: L'))$$

is the iff-completion of $sorted : list$ (see Ex. 10.1.1).

Analogously, ν-predicates are completed with respect to the greatest fixpoint $\nu\Psi$ of the step function Ψ (see Theorem 10.3.1):

Lemma 10.3.10 Let r be a ν-predicate of SP and $AX_r = \{r(x) \Rightarrow (H_i \Rightarrow \exists X_i GS_i) \mid 1 \leq i \leq n\}$ be the set of axioms of SP for r and $Y_i = var(GS_i) \setminus X_i \setminus \{x\}$. The formula

$$r(x) \iff \forall Y_1(H_1 \Rightarrow \exists X_1 GS_1) \wedge \cdots \wedge \forall Y_n(H_n \Rightarrow \exists X_n GS_n),$$

called the **iff-completion** of r, is valid in $\nu\Psi$ and a final theorem of SP. ❑

For instance,

$$s \sim s' \iff head(s) \equiv head(s') \wedge tail(s) \sim tail(s') \tag{3}$$

is the iff-completion of $\sim$: $stream \times stream$ (see Ex. 10.2.5).

By negating the iff-completion of a predicate r one often obtains the complement of r.

Definition 10.3.11 Given predicates $r : w$ and $\overline{r} : w$ of SP, $\overline{r}$ is called the SP-**complement** of r if $\overline{r}^{Fin(SP)}$ is the complement of $r^{Fin(SP)}$ i.e. $\overline{r}^{Fin(SP)} = Fin(SP)_w \setminus r^{Fin(SP)}$. ❑

On the one hand, clause (3) suggests that the *least* interpretation of $\not\sim$: *stream* $\times$ *stream*, specified by the Horn clause

$$s \not\sim s' \quad \Leftarrow \quad head(s) \not\equiv head(s') \vee tail(s) \not\sim tail(s'), \tag{4}$$

yields the INFSEQ-complement of $\sim$. On the other hand, $\nu\Psi$ interprets $\sim$ as the *greatest* solution of (3). When does the *negation* of the iff-completion of a predicate r coincide with the iff-completion of $\neg r$? Is $\not\sim$ really the complement of $\sim$?

We first note that the step functions Φ and Ψ of Theorem 10.3.1 can be defined in terms of iff-completions: given $r \in \mu\Sigma$ and $q \in \nu\Sigma$ with iff-completions $r(x) \Leftrightarrow C_r$ resp. $q(x) \Leftrightarrow C_q$, we obtain

$$b(x) \in r^{\Phi(A)} \iff A \models_b C_r \qquad \text{and} \qquad b(x) \in q^{\Psi(A)} \iff A \models_b C_q$$

for all $b : X \to Ini(SP)$ and Σ-structures A whose $(\Sigma \setminus \nu\Sigma)$-reduct agrees with $Ini(SP)$.

Given $r \in P$, let $\overline{r}$ be another predicate of SP that is supposed to be the SP-complement of r. Let $r(x) \Leftrightarrow C_r$ and $\overline{r}(x) \Leftrightarrow C_{\overline{r}}$ be the iff-completions of r and $\overline{r}$, respectively. Let $r : w \in \mu\Sigma$ and $q : w \in \nu\Sigma$. Suppose that $\overline{r} \in \nu\Sigma$, $\overline{q} \in \mu\Sigma$ and for all $i \in \mathbb{N}$,

$$\overline{r}^{\Psi^i(\top)} = Ini(SP)_w \setminus r^{\Phi^i(\bot)}, \tag{5}$$

$$\overline{q}^{\Phi^i(\bot)} = Ini(SP)_w \setminus q^{\Psi^i(\top)}. \tag{6}$$

Then we obtain

$$\overline{r}^{\Psi_\infty} = \cap_{i\in\mathbb{N}}\overline{r}^{\Psi^i(\top)} = \cap_{i\in\mathbb{N}}(Ini(SP)_w \setminus r^{\Phi^i(\bot)}) = Ini(SP)_w \setminus \cup_{i\in\mathbb{N}} r^{\Phi^i(\bot)} = Ini(SP)_w \setminus r^{\Phi^\infty}, \tag{7}$$

$$\overline{q}^{\Phi^\infty} = \cup_{i\in\mathbb{N}}\overline{q}^{\Phi^i(\bot)} = \cup_{i\in\mathbb{N}}(Ini(SP)_w \setminus q^{\Psi^i(\top)}) = Ini(SP)_w \setminus \cap_{i\in\mathbb{N}} q^{\Psi^i(\top)} = Ini(SP)_w \setminus q^{\Psi_\infty}. \tag{8}$$

Hence $\overline{r}^{\Psi_\infty}$ and $\overline{q}^{\Phi^\infty}$ are the complements of r^{Φ^∞} and q^{Ψ_∞}, respectively. From this we can conclude easily that $\overline{r}$ and $\overline{q}$ are the SP-complements of r resp. q:

Theorem 10.3.12 (axioms for complements) Given a μ- resp. ν-predicate r and a ν- resp. μ-predicate $\overline{r}$ such that $r(x) \Leftrightarrow C_r$ and $\overline{r}(x) \Leftrightarrow C_{\overline{r}}$ are the iff-completions of r and $\overline{r}$, respectively, and $C_{\overline{r}}$ is equivalent to $\neg C_r[\overline{p}(t)/\neg p(t) \mid p \in P]$. If SP is downward continuous, then $\overline{r}$ is the SP-complement of r.

Proof. By Theorem 10.3.1, $Ini(SP) = \mu\Phi = \Phi^\infty$ and $\nu\Psi = \Psi_\infty$. Let $r : w$ be a predicate of SP and $[a] \in Fin(SP)_w$. If $r \in \mu\Sigma \setminus \tau\Sigma$, then

$$[a] \in \overline{r}^{Fin} \iff a \in \overline{r}^{\nu\Psi} \iff a \in \overline{r}^{\Psi_\infty} \overset{(7)}{\iff} a \notin r^{\Phi^\infty} \iff a \notin r^{Ini} \iff [a] \notin r^{Fin}.$$

If $r \in \nu\Sigma$, then

$$[a] \in \overline{r}^{Fin} \iff a \in \overline{r}^{\mu\Phi} \iff a \in \overline{r}^{\Phi^\infty} \overset{(8)}{\iff} a \notin r^{\Psi_\infty} \iff a \notin r^{\nu\Psi} \iff [a] \notin r^{Fin}.$$

If $r \in \tau\Sigma$, then $[a] = ([b],[c],[d])$ for some b, c, d and

$$[a] \in \overline{r}^{Fin} \iff (b,c,d) \in \overline{r}^{\nu\Psi} \iff \forall d' \sim_{SP} d : (b,c,d') \in \overline{r}^{\Psi_\infty}$$
$$\overset{(7)}{\iff} \forall d' \sim_{SP} d : (b,c,d') \notin r^{\Phi^\infty} \iff \forall d' \sim_{SP} d : (b,c,d') \notin r^{Ini} \iff [a] \notin r^{Fin}.$$

Hence in all three cases, $\overline{r}$ is the SP-complement of r.

It remains to show (5) and (6). We show both equations by induction on i. They hold true for $i = 0$ because for all $p \in P$. $p^\top = Ini(SP)_w$ and $p^\bot = \emptyset$ (see Theorem 10.3.1). Let $i > 0$ and (5) and (6) be valid for $i-1$. Let $b : X \to Ini(SP)$. (5) and (6) for $i-1$ imply

$$\Psi^{i-1}(\top) \models_b \overline{r}(t) \iff \Phi_{i-1}(\bot) \not\models_b r(t) \iff \Phi_{i-1}(\bot) \models_b \neg r(t)$$

and thus for all formulas C,

$$\Psi^{i-1}(\top) \models_b C[\overline{p}(t)/\neg p(t) \mid p \in P] \iff \Phi_{i-1}(\bot) \models_b C. \tag{9}$$

Hence

$$b(x) \in \overline{r}^{\Psi^i(\top)} \iff b(x) \in \overline{r}^{\Psi(\Psi^{i-1}(\top))} \iff \Psi^{i-1}(\top) \models_b C_{\overline{r}} \overset{assumption}{\iff} \Psi^{i-1}(\top) \models_b \neg C_r[\overline{p}(t)/\neg p(t) \mid p \in P]$$
$$\overset{(9)}{\iff} \Phi_{i-1}(\bot) \models_b \neg C_r \iff \Phi_{i-1}(\bot) \not\models_b C_r \iff b(x) \notin r^{\Phi(\Phi^{i-1}(\bot))} \iff b(x) \notin r^{\Phi^i(\bot)}.$$

Hence (5) holds true for i. (6) can be proved analogously. ❑

Theorem 10.3.12 implies that $\not\sim$ as specified by (4) is indeed the SP-complement of $\sim$: $stream \times stream$. Similarly, the iff-completion of the μ-predicate $sorted$ (see above) leads to the following co-Horn axioms for the ν-predicate $unsorted$:

$$\begin{array}{lll} unsorted(L) & \Rightarrow & (L \equiv nil \;\Rightarrow\; FALSE) \\ unsorted(L) & \Rightarrow & (L \equiv [x] \;\Rightarrow\; FALSE) \\ unsorted(L) & \Rightarrow & ((L \equiv x :: y :: L' \wedge x \leq y) \;\Rightarrow\; unsorted(y :: L')). \end{array}$$

Both—alternative—axioms for the ν-predicate $fair$ of INFSEQ (Ex. 10.2.5) yield generalized[13] Horn axioms for $unfair$:

$$\begin{array}{lll} unfair(s) & \Leftarrow & existsForall(\lambda x.not(eq(x,0)), s) \\ unfair(s) & \Leftarrow & \forall\{L, s'\}((0 \notin L \wedge s \sim L\#(0\&s')) \Rightarrow unfair(s')). \end{array}$$

Lemmas 10.3.9 and 10.3.10, Theorem 10.3.12 and Lemma 10.4.1 given below are crucial for the correctness of inference rules presented in Section 10.7. Let us now turn to the general proof schema for reasoning about standard specifications.

10.4 Gentzen proofs and program synthesis

In the previous section we have defined the basic theories of a standard specification that capture its loose, reachable, initial and final semantics, respectively (see Defs. 10.2.13, 10.3.2). We discussed the dominant rôle of initial semantics because it reflects the intuition behind *free*-constructor-based types and entails efficient proof rules for reasoning about such *visible* types. However, a data type DT that involves non-free constructors, hidden sorts or predicates "dealing with infinity" is better given a final semantics. We also mentioned the Horn clause completion of SP, which, on the one hand, allows us to reduce final to initial models, and thus, on the other hand, does not entail special proof rules for hidden types. As proofs about the visible part of SP are mainly based on induction and least fixpoint properties, so can proofs about the hidden part of SP be based on greatest fixpoint properties. Moreover, "meta-theorems" such as Lemmas 10.3.9 and 10.3.10 and Theorem 10.3.12, which hold true under general requirements to SP, provide for **simplification rules** that speed up proofs considerably because they can be applied—in contrast to fixpoint rules—almost automatically (see Section 10.7).

Previous examples should have justified our assertion that Gentzen clauses are sufficient for presenting not only a data type DT itself, but also all interesting conjectures about DT, such as invariants and other conditions on functions or predicates of DT that one might demand in order to call them "correct". In contrast to the formulas used in classical ATP approaches (see Section 10.1) Gentzen clauses are not the result of a normalization

[13]see above

process that often hides the meaning of the original formulas considerably and thus makes proofs difficult to follow, let alone to control interactively. In particular, inductive proofs *require* interaction for guiding them towards induction hypotheses or setting up generalizations. Hence we focus on **Gentzen rules**, i.e. proof rules that relate (finite) sets (= conjunctions) of Gentzen clauses to each other. A derivation consisting of Gentzen rule applications is called a **Gentzen proof**. A Gentzen proof is **successful** if it terminates with the Gentzen clause $TRUE$. In contrast to a derivation with the cut calculus (Def. 10.2.10), a Gentzen proof proceeds

- **top-down** or **backward** because it starts with the conjecture to be proved and argues backwards by establishing inverse implications between successively derived clause sets;
- **linear** since a Gentzen rule is always applied to the clause set derived at latest, in other words, the inference relation built up of a set of Gentzen rules is the transitive closure of the set of rule instances.

A Gentzen clause $GS \Leftarrow HS$ is an implication (first level) such that both the conclusion GS and the premise HS are goal sets (second level), which are disjunctions of goals (third level), which are conjunctions of atoms (fourth level). Most steps of a Gentzen proof work on the second level, i.e. modify GS or HS separately. On the first level, this means applying the following two Gentzen rules: Let SP be a standard specification with final model $Fin(SP)$.

head expansion	$\dfrac{GS_0 \Leftarrow HS}{GS_1 \Leftarrow HS} \Uparrow$	if GS_0 **expands** to GS_1, i.e. $Fin(SP) \models GS_0 \Leftarrow GS_1$
body contraction	$\dfrac{GS \Leftarrow HS_0}{GS \Leftarrow HS_1} \Uparrow$	if HS_0 **contracts** to HS_1, i.e. $Fin(SP) \models HS_0 \Rightarrow GS_1$

Rules for expanding resp. contracting formulas are called **expansion** resp. **contraction rules**. Contractions establish implications in the direction in which the proof proceeds. Expansions yield implications in the opposite direction.[14] A rule is **correct** if the implication is valid in the final model of any specification for which such a model exists (see Theorem 10.3.5). Since Gentzen proofs always proceed backwards, all Gentzen rules are expansions. The arrow attached to a rule indicates whether it is an expansion ($\Uparrow$), a contraction ($\Downarrow$) or both ($\Updownarrow$). In the last case the rule establishes an equivalence between two formulas and is called a **simplification**, which appeals to the fact that it often simplifies a formula in some respect. Most Gentzen rules are presented in Section 10.7.

Besides proving the correctness of a specification SP by showing the final validity of selected conjectures, SP may be tested by **solving goals**, in particular equations, in (existentially quantified) variables. In functional languages, solving is restricted to ground term evaluation, i.e. to solving equations of the form $t \equiv x$ with $x \in X \setminus var(t)$. In logic languages, a goal to be solved is called a query and provides the input to a logic program. Most queries can be answered by applying Gentzen rules. A goal H is a **solved goal** if H is a conjunction $x_1 \equiv t_1 \wedge \cdots \wedge x_n \equiv t_n$ where $x_1, \ldots, x_n$ are variables and $t_1, \ldots, t_n$ are normal forms. Then a clause $G \Leftarrow H$ holds true iff $G[t_1/x_1, \ldots, t_n/x_n]$ holds true. Hence G is solved by expanding G to a solved goal. In pure logic programming, solved goals are the only presentations of solutions. CLP (= constraint logic programming) accepts more general forms of solutions, like the goal $x > 5$, which covers an infinite set of solved goals.

Conversely, G is unsolvable if G contracts to $FALSE$. In Section 10.8, we deal with particular rules for deriving solved goals and discuss their completeness: does a normal form substitution σ solve a goal G only if the rules expand G to the solved goal $\bigwedge_{x \in var(t)} x \equiv x\sigma$?

For obtaining solved goals one needs to stepwise remove logical predicates and defined functions from the

[14] *Inductive logic programming* calls expansions *inductive* rules and contractions *deductive* rules [MR 94].

goal to be solved. These rules, which are called **unfold rules**, are direct consequences of Lemmas 10.3.9 and 10.3.10 and will be presented in Section 10.7. If all logical predicates and defined functions have been removed, we might still be left with goals consisting of equations between normal forms. The following lemma allows us to reduce such goals to solved ones.

Lemma 10.4.1 Let SP be a functional specification. c, d be different visible constructors of SP, $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$. Then the following equivalences are final theorems of SP:

$$\begin{array}{rcl} c(x) \equiv c(y) & \Longleftrightarrow & (x_1 \equiv y_1 \wedge \cdots \wedge x_n \equiv y_n) \vee (c(x) \equiv c(y) \equiv \bot) \\ c(x) \equiv d(y) & \Longleftrightarrow & c(x) \equiv d(y) \equiv \bot \\ c(x) \equiv \bot & \Longleftrightarrow & x_1 \equiv \bot \vee \cdots \vee x_n \equiv \bot \\ c \equiv \bot & \Longleftrightarrow & FALSE \\ x \equiv \bot \wedge Def(t) & \Longleftrightarrow & FALSE \end{array}$$

Moreover, if for all different visible constructors c, d of SP, SP is extended by the Horn axioms

$$\begin{array}{rcll} c(x) \not\equiv c(y) & \Leftarrow & (x_i \not\equiv y_i \wedge c(x) \not\equiv \bot) \vee (x_i \not\equiv y_i \wedge c(y) \not\equiv \bot) & \text{for all } 1 \leq i \leq n \\ c(x) \not\equiv d(y) & \Leftarrow & c(x) \not\equiv \bot \vee d(y) \not\equiv \bot & \\ c(x) \not\equiv \bot & \Leftarrow & x_1 \not\equiv \bot \wedge \cdots \wedge x_n \not\equiv \bot & \\ c \not\equiv \bot, & & & \end{array}$$

then for all visible sorts s of SP, $\not\equiv^{Ini(SP)}: ss$ is the complement of SP-equivalence (Def. 10.3.11). ❑

As Lemma 10.3.10 and Theorem 10.3.12 lead to inference rules for handling hidden equations and inequations, Lemma 10.4.1 provides us with rules for visible equations and inequations. If they are applied as soon as possible, they usually reduce the search space of a Gentzen proof significantly. Therefore, many theorem provers realize them as part of a built-in simplifier that works in the background and applies rules automatically (see Section 10.7).

The functionality of a specification includes its completeness (Def. 10.2.10). Besides concluding completeness from termination (Def. 10.5.6) one may use the following characterization: SP is complete iff for each defined function $f : s_1 \ldots s_n \to s$ of SP, $Fin(SP)$ satisfies the Horn clause

$$hasNF_s(f(x)) \quad \Leftarrow \quad hasNF_w(x)$$

where the predicate $hasNF_s$ is specified by the following axioms:

$$\begin{array}{rcll} hasNF_s(c(x)) & \Leftarrow & hasNF_v(x) & \text{for all hidden constructors } c : v \to s \\ hasNF_s(x) & \Leftarrow & Def(x) \vee x \equiv \bot & \text{if } s \text{ is visible} \\ hasNF_{s_1 \ldots s_n}(x_1, \ldots, x_k) & \Leftarrow & hasNF_{s_1}(x_1) \wedge \cdots \wedge hasNF_{s_n}(x_n) & \text{for all sorts } s_1, \ldots, s_n. \end{array}$$

Given an arbitrary subspecification SP' of SP, $Ini(SP)$ is consistent wrt $Ini(SP')$ iff for all ground Σ'-atoms p, $SP \vdash_{cut} p$ implies $SP' \vdash_{cut} p$, intuitively: iff the step from SP' to SP does not lead to "confusion", i.e. properties of SP'-data, for example equivalences, that did not already hold in SP'. Consistency is a crucial proof obligation when specifications are developed hierarchically. But it also occurs when one designs *in-the-small*, namely when a defined function or μ-predicate is redefined by replacing its axioms by equivalent ones. Then consistency is accompanied by a change of signatures and thus replaced the more general property of inductive equivalence along signature morphisms:

Definition 10.4.2 (inductive equivalence) Given two specifications $SP_i = (\Sigma_i, AX_i)$, $i = 1, 2$, and signature morphisms $\sigma : \Sigma' \to \Sigma_1$ and $\tau : \Sigma' \to \Sigma_2$, SP_1 and SP_2 are **inductively equivalent along** (σ, τ) if

$gen(Ini(SP_1)|_\sigma)$ and $gen(Ini(SP_2)|_\tau)$ are isomorphic or, equivalently, for all ground Σ'-atoms p, $SP_1 \vdash_{cut} \sigma(p)$ iff $SP_2 \vdash_{cut} \tau(p)$. ❑

Σ' comprises the functions and predicates to be redefined together with those that remain the same, while $\Sigma_1 \setminus \Sigma'$ resp. $\Sigma_2 \setminus \Sigma'$ are the auxiliary functions and predicates used in SP_1- resp. SP_2-axioms for Σ'. The proof that SP_1 and SP_2 are inductively equivalent splits into three steps.

(1) For all functions f and logical predicates r of Σ', $\sigma(f)(x) \equiv \tau(f)(x)$ and $\sigma(r)(x) \Leftrightarrow \tau(r)(x)$ are inductive theorems of $SP_1 \cup SP_2$.

(2) $Ini(SP_1 \cup SP_2)$ is consistent wrt $Ini(SP_1)$.

(3) $Ini(SP_1 \cup SP_2)$ is consistent wrt $Ini(SP_2)$.

Indeed, (1)-(3) imply

$$SP_1 \vdash_{cut} \sigma(p) \overset{(2)}{\Longleftrightarrow} SP \vdash_{cut} \sigma(p) \overset{(1)}{\Longleftrightarrow} SP \vdash_{cut} \tau(p) \overset{(3)}{\Longleftrightarrow} SP_2 \vdash_{cut} \tau(p)$$

for all ground Σ'-atoms p. Roughly said, Condition (1) states that certain functions and predicates occurring in SP_1 and SP_2 are equivalent with respect to the inductive theory of the *union* of these specifications. Conditions (2) and (3) ensure that the equivalences are preserved when SP_1 (resp. SP_2) is *replaced* by SP_2 (resp. SP_2).

Consistency is also a proof obligation when a "concrete" specification CSP is supposed to implement an "abstract" specification ASP. The implementation itself is presented as a further specification ISP and related to CSP and ASP via signature morphisms $\alpha : ASP \to ISP$ and $\kappa : CSP \to ISP$ (see Chapter 7). In terms of Section 10.2, one may call ISP a **correct implementation of** ASP **by** CSP if $gen(Fin(ISP)|_\alpha)$ is a hierarchical ASP-model over $Ini(visASP)$ and if $Fin(ISP)|_\kappa$ is consistent wrt $Fin(CSP)$ (Def. 10.3.4). Both conditions involve consistency requirements, which can often be satisfied with the help of Theorem 10.5.16.

We have defined the semantics of a parameterized specification $PSP = (PAR, SP)$ as the class of actualizations of PSP by PAR-models where an actualization $SP(A)$ combines a PAR-model A with SP in the way described in Section 10.1. A may be given as a structure whose signature, say, Σ, is different from the signature of PAR, say, Σ'. Then we need a signature morphism $\sigma : \Sigma' \to \Sigma$ and it is not A, but the σ-reduct $A|_\sigma$ of A, that is combined with SP. The actualizations of PSP are often described as values of a **functor** $F_{PSP} : Mod(PAR) \to Mod(SP)$ (see [EM 85], Chapter 7). Here F_{PSP} maps A to $Fin(SP(A))$. The question whether, for a single PAR-model A, $Fin(SP(A))$ is consistent wrt A is generalized to the question on which subclass $\mathcal{C}$ of $Mod(PAR)$ F_{PSP} is **persistent**, i.e. maps *each* $A \in \mathcal{C}$ to an SP-model that is consistent wrt A.

Deductive program synthesis and **transformation** deals with rules that relate inductively equivalent specifications to each other (see, e.g., Chapter 15 and [MPS 93]). The rules were mostly derived from proofs of program equivalences. For instance, let axioms for two functions f, g be given and suppose the task is to replace the axiom

$$(f;g)(x) \equiv g(f(x)) \tag{1}$$

for their composition by new axioms that do not refer to f and g. We may start an expansion of (1) using the axioms for f, g as well as functions and predicates f or g refers to. At some stage of the expansion we may obtain a clause that includes a term of the form $g(f(t))$. We rewrite $g(f(t))$ into $(f;g)(t)$, i.e. perform a **fold step**, which is actually an induction step (see Section 10.7). We proceed with the expansion until—hopefully—we are left with Horn clauses that satisfy the syntactic conditions of a standard specification on Horn axioms for $(f;g)$. We exchange them for (1) and conclude that (1) is a final *theorem* of the modified specification. The expansion can also be regarded as the derivation of a solution of (1) in the function variable $(f;g)$. In the functional programming and term rewriting communities, rule-based systems for solving goals in function variables are called **higher-order unification** procedures. The solutions come as λ-expressions, though never involve fixpoint operators, which means that these procedures do not (yet?) synthesize *recursive* programs. The restriction is due to the purely functional treatment of formulas and proofs in those communities. Traditional approaches to program synthesis got further here because they are more logic-oriented. In particular, fold steps realize induction steps (see above) and these are indispensible for deriving recursive programs.

In order to show how classical methods of program synthesis fit into the functional-logic approach let us reformulate the well-known **assertion method** for verifying iterative programs in terms of a Gentzen rule, called **Hoare induction**. Given the input-output relation $r : A \times C$ of a function $f : A \to C$ to be defined, the task is to construct an iterative program for f. An **iterative program for** f is a set of axioms of the following form:

$$\begin{array}{l} f(x) \equiv iter(in(x)) \\ iter(y) \equiv iter(loop_1(y)) \;\Leftarrow\; G_1(y) \\ \dots \\ iter(y) \equiv iter(loop_n(y)) \;\Leftarrow\; G_n(y) \\ iter(y) \equiv out(y) \;\Leftarrow\; G_{n+1}(y) \end{array}$$

where the **iterator** $iter : B \to C$, $in : A \to B$, $out : B \to C$, $loop_1, \dots, loop_n : B \to B$ and the *guards* $G_1, \dots, G_{n+1}$ are auxiliary functions and goals, respectively. The program for f and the axioms for r are supposed to be part of a specification SP such that $r(x,z) \Leftarrow f(x) \equiv z$ is valid in the $Fin(SP)$. This clause is the premise of Hoare induction:

Hoare induction

$$\frac{r(x,z) \;\Leftarrow\; f(x) \equiv z}{\begin{array}{ll} & inv(x, in(x)) \\ \wedge & inv(x, loop_1(y)) \;\Leftarrow\; inv(x,y) \wedge G_1(y) \\ & \dots \\ \wedge & inv(x, loop_n(y)) \;\Leftarrow\; inv(x,y) \wedge G_n(y) \\ \wedge & r(x, out(y)) \;\Leftarrow\; inv(x,y) \wedge G_{n+1}(y) \end{array}} \Uparrow$$

The conclusion of the rule "generates" the predicate $inv : A \times B$, called the **Hoare invariant**. inv may be regarded as a generalization of r. While r relates the input domain A of f to the output domain C of f, inv relates A to the intermediate domain B. A symmetric rule captures **subgoal induction** [MW 77]. Instead of the Hoare invariant, this rule generates a **subgoal invariant** $sgi : B \times C$, which relates the intermediate domain B to the output domain of f and actually coincides with the input-output relation of $iter$ [Pad 97]. sug can be defined in terms of inv and vive versa:

$$\begin{array}{lcl} sgi(y,z) & \Longleftrightarrow & \forall x : r(x,z) \Leftarrow inv(x,y), \\ inv(x,y) & \Longleftrightarrow & \forall z : r(x,z) \Leftarrow sgi(y,z). \end{array}$$

Example 10.4.3 The following specification includes an iterative program for a function *split* that, given $n \in \mathbb{N}$ and a list L, decomposes L into sublists $L_1, \dots, L_k$ such that $L_1, \dots, L_{k-1}$ have length n.

```
PARTITION = NAT + LIST +
     defuns      split:nat*list->list(list)
                 iter:nat*list*nat*list(list)->list(list)
                 flatten:list(list)->list
                 length:list->nat
```

```
preds      lengthOK:nat*list(list)
vars       x:entry  n,k:nat  L,L':list  P:list(list)
Horn axioms
           split(n,L) = iter(n,L,0,nil,nil)
           iter(n,x::L,k,L',P) = iter(n,L,k+1,L'@[x],P)  <==  k < n
           iter(n,x::L,k,L',P) = iter(n,x::L,0,nil,P@[L'])  <==  k = n
           iter(n,nil,k,L',P) = P@[L']
           flatten(nil) = nil
           flatten(L::P) = L@flatten(P)
           lengthOK(n,nil)
           lengthOK(n,[L])  <==  length(L) <= n
           lengthOK(n,L::P)  <==  length(L) = n /\ lengthOK(n,P)
           length(nil) = 0
           length(x::L) = length(L)+1
```

Let $A = nat \times list$ and $C = list(list)$. *split* is supposed to satisfy the following input-output relation $r : A \times C$:

$$r(n, L, P) \quad \Longleftrightarrow_{def} \quad L \equiv flatten(P) \wedge lengthOK(n, P).$$

To prove $r(n, L, P) \Leftarrow split(n, L) \equiv P$, we set $B = A \times A \times C$, apply Hoare induction and obtain the following requirements to a Hoare invariant $inv : A \times B$:

$$\begin{array}{lcl}
inv(n, L, n, L, 0, nil, nil) & & \\
inv(n, L, n', L', k+1, L''@[x], P) & \Leftarrow & inv(n, L, n', x :: L', k, L'', P) \wedge k < n' \\
inv(n, L, n', x :: L', 0, nil, P@[L'']) & \Leftarrow & inv(n, L, n', x :: L', k, L'', P) \wedge k = n' \\
r(n, L, flatten(P@[L''])) & \Leftarrow & inv(n, L, n', nil, k, L'', P).
\end{array}$$

In the course of expanding these clauses to $TRUE$ (using rules presented in Section 10.7) one "derives" the following definition of inv:

$$\begin{array}{rcl}
inv(n, L, n', L', k, L'', P) & \Longleftrightarrow_{def} & L \equiv flatten(P@[L''@L']) \wedge n \equiv n' \wedge k \equiv length(L'') \\
& & \wedge\ lengthOK(n, P@[L'']). \ \square
\end{array}$$

10.5 Confluence and consistency

In this section we deal with confluence, which is the crucial condition on a standard specification SP that allows us to switch back and forth between SP and its relational version $rel(SP)$ (Def. 10.2.15). Confluence ensures that SP and $rel(SP)$ are equivalent in the sense of Theorem 10.5.5. As already pointed out in Section 10.2 this implies that $Ini(SP)$ interprets not only the logical predicates, but also the input-output predicate of all defined functions of SP as the least relations satisfying their axioms (Theorem 10.3.5(2)). In turn, this guarantees the correctness of most important proof rules that involve the functional part of SP, namely unfolding, normal form handling, fixpoint induction (Section 10.7) and its derivatives such as Hoare induction (see above). Moreover, confluence provides us with simple criteria for the consistency of SP (Def. 10.2.10) and the consistency of $Ini(SP)$ with respect to the initial models of subspecifications of SP (Def. 10.3.4). Recall that the consistency of SP is also needed for showing that $Fin(SP)$ is final in the class of hierarchical SP-models over $Ini(visSP)$ (Theorem 10.3.5(5)). Morever, we have seen in the previous section how the consistency of the initial model of $SP_1 \cup SP_2$ is used for proving that SP_1 and SP_2 are inductively equivalent.

The notion of confluence is based on the **rewrite relation** $\longrightarrow_{SP}$ that is derived from the axioms of SP. For dealing with predicates as well as with functions we first define the **goal reduction calculus** for SP that yields the basic rules $\longrightarrow_{SP}$ is built upon. Both the cut calculus (Def. 10.2.10) and the reduction calculus are used for proving goals. However, the former produces bottom-up proofs starting out from axioms of SP, while the latter proves a goal top-down by transforming it into the empty goal. Confluence is characterized as the equivalence of both calculi with respect to ground goals (Theorem 10.5.3).

A **fresh** or **extra variable** of a Horn clause $C = (t\{\equiv u\} \Leftarrow H)$[15] is a variable that occurs in u or H, but not in t. **fresh**(C) denotes the set of fresh variables of the clause. Note that Condition 10.2.4(a) on a standard specification implies that all fresh variables of C occur in the premise H.

Definition 10.5.1 Let $SP = (\Sigma, AX)$ be a standard specification. The **reduction calculus for** SP consists of the following rules for reducing goals. Let G be a goal and σ be a substitution over Σ.

rewriting $\dfrac{G(t\sigma)}{G(u\sigma) \wedge H\sigma} \Uparrow$ if $C = (t \equiv u \Leftarrow H) \in AX$ and $fresh(C)\sigma \subseteq NF_\Sigma(X)$

resolution $\dfrac{r(t\sigma) \wedge G}{H\sigma \wedge G} \Uparrow$ if $r \not\equiv \equiv$, $C = (r(t) \Leftarrow H) \in AX$ and $fresh(C)\sigma \subseteq NF_\Sigma(X)$

reflection $\dfrac{t \equiv t \wedge G}{G} \Uparrow$

A sequence $G_1, \ldots, G_n$ of goals such that for all $1 \leq i < n$, G_{i+1} is obtained from G_i by applying one of the above rules, is called an SP**-reduction of** G_1 **into** G_n and we write $G_1 \vdash_{SP} G_n$. A goal G is SP**-convergent** if $G \vdash_{SP} \emptyset$.

The SP**-rewrite relation** is the binary relation on terms and goals that is defined as follows: Let r be any unary predicate of Σ. For all $t, u \in T_\Sigma(X)$ and goals G,

$$G(t) \longrightarrow_{SP} G(u) \iff_{def} t \longrightarrow_{SP} u \iff_{def} \exists \text{ a rewrite step } r(t) \vdash_{SP} r(u) \wedge H \text{ such that } H \vdash_{SP} \emptyset.$$

A term or goal u is an SP**-reduct** of a term or goal t if $t \stackrel{*}{\longrightarrow}_{SP} u$. t is SP**-reduced** if $t \stackrel{*}{\longrightarrow}_{SP} u$ implies $t = u$. A substitution τ is an SP**-reduct** of a substitution σ if $x\sigma \stackrel{*}{\longrightarrow}_{SP} x\tau$ for all $x \in X$. A goal G is **strongly** SP**-convergent** if all SP-reducts of G are SP-convergent.

Two terms are SP**-joinable** if they have a common SP-reduct. SP is (ground) **confluent** if for all ground terms t, each two SP-reducts of t are SP-joinable. ❑

If SP has no logical predicates, then $\stackrel{*}{\longrightarrow}_{SP}$ coincides with *join conditional rewriting* in the sense of Section 9.2.3. By Def. 10.2.4, all Σ-normal forms are SP-reduced. $\vdash_{SP}$ and $\longrightarrow_{SP}$ are correct with respect to $Ini(SP)$, i.e. for all goals G, H and terms t, u,

$$G \vdash_{SP} H \;\Rightarrow\; Ini(SP) \models G \Leftarrow H, \text{ in particular, } G \vdash_{SP} \emptyset \;\Rightarrow\; SP \vdash_{cut} G, \tag{1}$$

$$t \stackrel{*}{\longrightarrow}_{SP} u \;\Rightarrow\; t \equiv_{SP} u. \tag{2}$$

Moreover,

$$G(t) \stackrel{*}{\longrightarrow}_{SP} G(u) \vdash_{SP} \emptyset \;\Rightarrow\; G(t) \vdash_{SP} \emptyset, \tag{3}$$

$$t \text{ and } u \text{ are } SP\text{-joinable} \iff t \equiv u \vdash_{SP} \emptyset. \tag{4}$$

Lemma 10.5.2 SP is confluent iff all SP-convergent ground goals are strongly SP-convergent.

Proof. "$\Leftarrow$": Let u, u' be reducts of a ground term t. Since $t \stackrel{*}{\longrightarrow}_{SP} u$ and $u \equiv u$ is convergent, (3) implies $t \equiv u \vdash_{SP} \emptyset$. Since $t \stackrel{*}{\longrightarrow}_{SP} u'$ and convergent ground goals are strongly convergent, we conclude $u' \equiv u \vdash_{SP} \emptyset$. Hence by (4), u' and u are joinable.

[15] Recall that set brackets in a formula enclose optional subexpressions. $t\{\equiv u\}$ stands for t or $t \equiv u$.

"⇒": Let $r(t)$ be a ground atom and u be a ground term such that $r(t) \vdash_{SP} \emptyset$ and $t \xrightarrow{+}_{SP} u$. We show $r(u) \vdash_{SP} \emptyset$ by induction on a shortest reduction $r(t), G_1, \ldots, G_n, \emptyset$. Let the step from $r(t)$ to G_1 be a rewrite step. Then $G_1 = (r(t') \wedge H)$ and $t \longrightarrow_{SP} t'$ for some v, H. Since SP is confluent, t' and u have a common reduct v. By induction hypothesis, $r(t') \vdash_{SP} \emptyset$ and $t' \xrightarrow{*}_{SP} v$ imply $r(v) \vdash_{SP} \emptyset$. Since $u \xrightarrow{*}_{SP} v$, (3) implies $r(u) \vdash_{SP} \emptyset$.

Let the step from $r(t)$ to G_1 be a resolution step. Then there is an axiom $r(v) \Leftarrow H$ in SP such that $t = v\sigma$ and $G_1 = H\sigma$ for some σ. By Conditions 10.2.4(b) and (c), v is a tuple $(v_1, \ldots, v_n)$ of normal forms such that for all $1 \leq i \leq n$, $v_i \in X \cup \{\perp\}$ or $Def(v_i) \in H$. Moreover, there are $1 \leq k \leq n$ and a term u_k such that $v_k\sigma \xrightarrow{+}_{SP} u_k$ and $u = (v_1\sigma, \ldots, u_k, \ldots, v_n\sigma)$. Hence $v_k \in X$ or $Def(v_k) \in H$.

Suppose that there is τ with $\sigma \xrightarrow{*}_{SP} \tau$ and $u \xrightarrow{*}_{SP} v\tau$. By induction hypothesis, $H\sigma \vdash_{SP} \emptyset$ implies $H\tau \vdash_{SP} \emptyset$. Hence $r(v\tau)$ and thus $r(u)$ are convergent. If $v_k \in X$, we define τ by $v_k\tau = u_k$ and $x\tau = x\sigma$ for all $x \in X \setminus \{v_k\}$ and indeed obtain $\sigma \xrightarrow{*}_{SP} \tau$ and $u = (v_1\sigma, \ldots, u_k, \ldots, v_n\sigma) \xrightarrow{*}_{SP} (v_1\tau, \ldots, u_k, \ldots, v_n\tau) = v\tau$.

Let $Def(v_k) \in H$. Since v_k is a normal form and $Def(v_k\sigma)$ is a subgoal of $H\sigma$ and thus convergent, the axioms for Def (see Def. 10.2.4) imply that there is σ' with $\sigma \xrightarrow{*}_{SP} \sigma'$ and $Def(x\sigma') \vdash_{SP} \emptyset$ for all $x \in var(v_k)$. Hence $Def(x\sigma) \vdash_{SP} \emptyset$ and thus $v_k\sigma \xrightarrow{+}_{SP} u_k$ implies $u_k = v'_k\tau'$ and $v'_k\rho = v_k$ for a term v'_k, a substitution τ' and a variable renaming ρ such that v'_k is linear in X and for all $x, y \in var(v'_k)$, $x\rho = y\rho$ implies $x\tau' \xleftarrow{*}_{SP} x\rho\sigma = y\rho\sigma \xrightarrow{*}_{SP} y\tau'$. Since SP is confluent, we conclude $v'_k\tau' \xrightarrow{*}_{SP} v_k\tau$ for some τ with $x\tau' \xrightarrow{*}_{SP} x\rho\tau$ for all $x \in var(v'_k)$. Hence $x\rho\sigma \xrightarrow{*}_{SP} x\tau' \xrightarrow{*}_{SP} x\rho\tau$ for all $x \in var(v'_k)$ and thus $x\sigma \xrightarrow{*}_{SP} x\tau$ for all $x \in var(v_k)$. Moreover, $u_k = v'_k\tau' \xrightarrow{*}_{SP} v_k\tau$. W.l.o.g. we define $x\tau = x\sigma$ for all $x \in X \setminus var(v_k)$ and obtain—again—$\sigma \xrightarrow{*}_{SP} \tau$ and $u = (v_1\sigma, \ldots, u_k, \ldots, v_n\sigma) \xrightarrow{*}_{SP} v\tau$.

Let the step from $r(t)$ to G_1 be a reflection step. Then $r(t) = (t' \equiv t')$, $t' \xrightarrow{*}_{SP} u'$ and $r(u) = (u' \equiv t')$ for some t', u'. Since $u' \equiv u'$ is convergent, (3) implies $r(u) \vdash_{SP} \emptyset$. ❏

By (1), $\vdash_{SP}$ is correct with respect to $\vdash_{cut}$. The completeness coincides with confluence:

Theorem 10.5.3 (Church-Rosser Theorem)[16] A complete specification SP is confluent iff for all ground goals G, $SP \vdash_{cut} G$ implies $G \vdash_{SP} \emptyset$.

Proof. "⇐": By Lemma 10.5.2, it is sufficient to show that convergent goals are strongly convergent. So let $G \vdash_{SP} \emptyset$ and $G \xrightarrow{*}_{SP} G'$. By (1) and (2), $SP \vdash_{cut} G'$. Hence by assumption, $G' \vdash_{SP} \emptyset$.

The ⇒-part is proved by induction on the length of a shortest cut calculus proof of G (see [Pad 92], Theorem 6.5(2)). ❏

Corollary 10.5.4 (consistency criterion I) A complete specification SP is consistent and thus functional (Def. 10.2.10) iff SP is confluent.

Proof. "⇐": Let t, t' be two SP-equivalent ground normal forms. By the Church-Rosser Theorem, $t \equiv t' \vdash_{SP} \emptyset$. By (4), t and t' are joinable. By Def. 10.2.4, normal forms are joinable iff they are equal.

"⇒": Let u, u' be reducts of a ground term t. Since SP is complete, there are normal forms v, v' of u and u', respectively. By (2), u and u' and thus v and v' are SP-equivalent. Since SP is consistent, v and v' are equal and thus a common reduct of t. ❏

The next result is one of the main consequences of Theorem 10.5.3. It is used, for instance, in the proofs of Theorem 10.3.5(2) and Lemma 10.3.9.

Theorem 10.5.5 Let $SP = (\Sigma, AX)$ be the hidden extension level of a functional specification and

[16] This generalizes the classical Church-Rosser Theorem 9.1 for unconditional equational rewrite systems to specifications whose axioms are arbitrary Horn clauses.

$rel(SP) = (\Sigma', AX')$. For all defined functions $f \in \Sigma$, logical predicates $r \in \Sigma$ and $t, u \in T_{\Sigma'}$,

$$Ini(SP) \models f(t) \equiv u \iff Ini(rel(SP)) \models r_f(t,u), \tag{5}$$

$$Ini(SP) \models r(t) \iff Ini(rel(SP)) \models r(t). \tag{6}$$

Proof. Without loss of generality suppose that SP is flattened. Since SP is functional, $rel(SP)$ is functional, too, and thus Theorem 10.5.3 implies that (5) and (6) are equivalent to: for all defined functions $f \in \Sigma$, logical predicates $r \in \Sigma$ and $t, u \in T_{\Sigma'}$,

$$f(t) \equiv u \vdash_{SP} \emptyset \iff r_f(t,u) \vdash_{rel(SP)} \emptyset, \tag{7}$$

$$r(t) \vdash_{SP} \emptyset \iff r(t) \vdash_{rel(SP)} \emptyset. \tag{8}$$

Since SP is flattened, (7) and (8) can be shown easily by induction on the length of the respective reductions. ❏

For semantical as well as deductive reasons the confluence of a standard specification is as important as its behavioral congruence. For ensuring the latter property we have the criterion of coinductivity (Def. 10.3.6). The development of criteria for confluence has a long tradition in term rewriting (see Chapter 9). The general approach is to localize the property insofar as joinability needs to be shown only for *critical pairs*, i.e. for reducts that stem from axiom superpositions, and then to lift the joinability of critical pairs to full confluence by arguing inductively along a *reduction ordering*.

Definition 10.5.6 A standard specification SP with signature Σ is **terminating** if all SP-reduced terms are normal forms and there is a **reduction ordering** $>$ **for** SP, i.e. a well-founded and transitive relation on terms and goals[17] such that the following two conditions hold true:

axiom compatibility $r(t\sigma) > H\sigma$ resp. $v(t\sigma) > (v(u\sigma) \wedge H\sigma)$ for all Horn axioms $C = (r(t) \Leftarrow H)$ resp. $C = (t \equiv u \Leftarrow H)$ of SP, terms and atoms $v(x)$ and $\sigma \in T_\Sigma^X$ such that $fresh(C)\sigma \subseteq NF_\Sigma$ and $H\sigma$ is SP-convergent.

subterm compatibility $t > u$ for all $t \in T_\Sigma$ and all proper subterms u of t.

Let $>$ be a reduction ordering for SP. An SP-reduction $G_1, \ldots, G_n$ is **decreasing** if $G_i > G_{i+1}$ for all $1 \leq i < n$. Given a goal G, $\mu(\mathbf{G})$ and $\nu(\mathbf{G})$ denote the sets of atoms $r(t)$ of G such that r is a μ- resp. ν-predicate. Let R be a set of ν-predicates of SP and HC be a set of Horn clauses $r(t) \Leftarrow H$ such that $r \in R$, $live(H) \subseteq R$ and $\nu\Psi \models HC$ (see Theorem 10.3.1). HC is **$>$-reductive** if for all $r \in R$, $(r(t) \Leftarrow H) \in HC$ and $\sigma \in T_\Sigma^X$,

$$fresh(r(t) \Leftarrow H)\sigma \subseteq NF_\Sigma \ \wedge \ \mu(H)\sigma \vdash_{SP} \emptyset \quad \Rightarrow \quad r(t\sigma) > H\sigma. \ ❏.$$

The axiom compatibility of $>$ implies that all SP-reductions into the empty goal are decreasing. Moreover, rewrite steps decrease ground terms, i.e. $t \longrightarrow_{SP} t'$ implies $t > t'$. Moreover, the assumptions that $>$ is well-founded and all SP-reduced terms are normal forms imply that each ground term has a normal SP-reduct (Def. 10.5.1). Hence a terminating specification is always complete (Def. 10.2.10). The subterm compatibility of $>$ is needed for the above-mentioned induction step from critical pairs to the contexts in which they may occur.

The assumption "$H\sigma$ is SP-convergent" in axiom compatibility makes termination a rather weak requirement. The assumption says that a rewrite or resolution step must decrease a ground goal only if the premise of the applied axiom instance is convergent.

[17] Here a goal is regarded as the multiset of its atoms and an equation as the multiset of its two sides.

If HC is $>$-reductive, then the co-Horn axioms for R can be replaced by Horn axioms such that the resulting specification HSP is terminating in a slightly stronger sense: in axiom compatibility, it reads $\mu(H)\sigma$ instead of $H\sigma$. Let us show that the final model of HSP and the final model of SP are isomorphic:

Theorem 10.5.7 (Horn axioms for ν-predicates) Suppose that $SP = (\Sigma, AX)$ is terminating, confluent and downward continuous and R and HC are as in Def. 10.5.6. Construct HSP from SP by replacing the co-Horn axioms for R with HC. If HC is $>$-reductive, then $Fin(HSP)$ and $Fin(SP)$ are isomorphic.

Proof. By Theorem 10.3.1, it is sufficient to show $r^{Ini(HSP)} = r^{\nu\Psi}$ for all $r : w \in R$. Since $\nu\Psi$ satisfies HC and $r^{Ini(HSP)}$ is the least interpretation of r that satisfies HC, $r^{Ini(HSP)}$ is a subset of $r^{\nu\Psi}$ and thus it remains to show $r^{\nu\Psi} \subseteq r^{Ini(HSP)}$ or, equivalently,

$$\forall a \in Ini(HSP)_w \setminus r^{Ini(HSP)} : a \notin r^{\nu\Psi}. \tag{9}$$

So let $a \in Ini(HSP)_w \setminus r^{Ini(HSP)}$. Then there is $t \in T_\Sigma$ such that $[t] = a$ and $HSP \not\vdash_{cut} r(t)$. Hence by (1), $r(t)$ is not HSP-convergent. We say that a goal G **fails after i steps** iff each decreasing HSP-reduction of G consists of at most i goals and the last one of these is not the empty goal. Since $>$ is well-founded, $r(t) \not\vdash_{HSP} \emptyset$ implies that there is $i \in \mathbb{N}$ such that $r(t)$ fails after i steps. Hence (9) reduces to:

$$\exists i \in \mathbb{N} : r(t) \text{ fails after } i \text{ steps} \quad \Rightarrow \quad [t] \notin r^{\nu\Psi}. \tag{10}$$

Since SP is downward continuous, $\nu\Psi = \Psi_\infty = \cap_{i\in\mathbb{N}}\Psi^i(\top)$ by Theorem 10.3.1. Hence (10) follows from:

$$r(t) \text{ fails after } i \text{ steps} \quad \Rightarrow \quad [t] \notin r^{\Psi^{i+1}(\top)}. \tag{11}$$

We show (11) by induction on i.

Suppose that $r(t)$ fails after 0 steps. Then $r(t) \not> H\sigma$ for all $(r(u) \Leftarrow H) \in HC$ and $\sigma \in T_\Sigma^X$ such that $u\sigma = t$ and $fresh(r(u) \Leftarrow H)\sigma \subseteq NF_\Sigma$. Since HC is $>$-reductive, $\mu(H)\sigma$ is not SP-convergent. Since SP is terminating, SP is complete. Since SP is confluent, Theorem 10.5.3 implies $SP \not\vdash_{cut} \mu(H)\sigma$. Hence $Ini(SP)$ does not satisfy $\mu(H)\sigma$ and thus $\top$ does not satisfy $H\sigma$ (see 10.3.1). Hence $[t] = [u\sigma] \notin r^{\Psi(\top)}$ by the definition of Ψ.

Suppose that $r(t)$ fails after $i > 0$ steps. Let $(r(u) \Leftarrow H) \in HC$ and $\sigma \in T_\Sigma^X$ such that $u\sigma = t$ and $fresh(r(u) \Leftarrow H)\sigma \subseteq NF_\Sigma$. If $\mu(H)\sigma$ is not SP-convergent, we can conclude as above that $\Psi^i(\top)$ does not satisfy $H\sigma$ and thus $[t] = [u\sigma] \notin r^{\Psi^{i+1}(\top)}$ by the definition of Ψ.

Let $\mu(H)\sigma$ be SP-convergent. Then $r(t) > H\sigma$ because HC is $>$-reductive. Hence $H\sigma$ fails after $i-1$ steps. By induction hypothesis, $[v\sigma] \notin q^{\Psi^i(\top)}$ for all $q(v) \in \nu(H)\sigma$. Hence, if $\nu(H)$ is not empty, we conclude $[t] = [u\sigma] \notin r^{\Psi^{i+1}(\top)}$ from the definition of Ψ. If $\nu(H)$ is empty, then $H = \mu(H)$ and thus $H\sigma$ is SP-convergent. Hence there is a decreasing SP-reduction of $H\sigma$ into the empty goal. Since all SP-reductions are HSP-reductions, we conclude that $H\sigma$ does *not* fail, which contradicts the above assertion that $H\sigma$ fails after $i-1$ steps. ❏

If the assumptions of Theorems 10.3.12 and 10.5.7 hold true, even the complement of a μ-predicate can be specified by Horn axioms and thus both predicates may be used in Horn axioms for further μ-predicates or defined functions. For instance, we turn the co-Horn axioms for *unsorted* (see Section 10.3) into the following set HC of (generalized) Horn axioms by inverting the outer implication signs:

$$\begin{array}{lcl} unsorted(L) & \Leftarrow & (L \equiv nil \;\Rightarrow\; FALSE) \\ unsorted(L) & \Leftarrow & (L \equiv [x] \;\Rightarrow\; FALSE) \\ unsorted(L) & \Leftarrow & \forall\{x, y, L'\}((L \equiv x :: y :: L' \wedge x \leq y) \;\Rightarrow\; unsorted(y :: L')). \end{array}$$

Since $\nu\Psi$ satisfies HC and there is a suitable reduction ordering $>$ such that HC is $>$-reductive, we conclude from Theorem 10.5.7 that HC provides an equivalent specification of *unsorted*. Some ideas involved in the proof of Theorem 10.5.7 (especially Implication (10)) are quite similar to arguments that justify **negation as failure** in logic programming (see [Apt 90], Section 5).

Termination is assumed in the Superposition Theorem given below, which characterizes confluence as *subjoinability* of *critical clauses*:

Definition 10.5.8 (critical clause)[18] Let $SP = (\Sigma, AX)$ be a standard specification, v be a term or an atom with a single occurrence of a variable x,

$$C \;=\; t\{\equiv u\} \Leftarrow H \quad \text{and} \quad C' \;=\; t' \equiv u' \Leftarrow H'$$

be Horn axioms of SP and σ, τ be minimal substitutions (see 10.2.2) such that $t\sigma = v[t'\tau/x]$ and the position of x in v agrees with the position of a function symbol in t. Then the Horn clause

$$CC \;=\; v[u'\tau/x]\{\equiv u\sigma\} \;\Leftarrow\; H\sigma \wedge H'\tau$$

is the SP**-critical clause induced by** C **on** C' **at** $t\sigma$. CC is an SP**-overlay** if $v = x$.

SP is **weakly orthogonal**[19] if for each SP-critical clause $t\{\equiv t'\} \Leftarrow H$, $t = t'$ or $Ini(SP) \models \neg H$. ❏

By Def. 10.2.4, each SP-critical clause is either an overlay or induced by a non-constructor axiom on a constructor axiom. Moreover, each overlay is induced by two (conditional) equations. If SP has no Horn axioms with fresh variables and no critical clause is induced by two different clauses C, C', then SP is weakly orthogonal. If, however, C has fresh variables, then C may induce a critical clause $t \equiv t' \Leftarrow H$ *on C itself* such that t is different from t' (see Ex. 10.5.12).

Definition 10.5.9 (subjoinability)[20] Let $SP = (\Sigma, AX)$ be a standard specification. Given a binary relation $>$ on terms and goals, a Horn clause $p \Leftarrow H$ is **subjoinable below** a term or atom t if

$$H\sigma \vdash_{SP} \emptyset \;\;\Rightarrow\;\; p\sigma \vdash_{SP} \emptyset$$

for all $\sigma \in T_\Sigma^X$ such that all SP-convergent ground goals $G < t\sigma$ are strongly SP-convergent (Def. 10.5.1). ❏

Subjoinability is a variant of inductive validity (Def. 10.2.13) where the cut calculus is replaced by the reduction calculus.

Theorem 10.5.10 (Superposition Theorem I)[21] A terminating specification is confluent iff for all terms and atoms t, each SP-critical clause induced at t is subjoinable below t.

Proof. "only-if": Let SP be confluent, $>$ be a reduction ordering for SP, C, C', CC be as in Def. 10.5.8 and $\rho \in T_\Sigma^X$ such that $H\sigma\rho \wedge H'\tau\rho$ is SP-convergent and all SP-convergent ground goals $G < t\sigma\rho$ are strongly SP-convergent. Since $>$ is axiom compatible, $t\sigma\rho > H\sigma\rho \wedge H'\tau\rho$. Hence $H\sigma\rho$ and $H'\tau\rho$ are strongly SP-convergent. Since $>$ is well-founded, there is a normal SP-reduct ρ' of ρ. Hence $H\sigma\rho \stackrel{*}{\longrightarrow}_{SP} H\sigma\rho'$, $H'\tau\rho \stackrel{*}{\longrightarrow}_{SP} H'\tau\rho'$ and thus $H\sigma\rho'$ and $H'\tau\rho'$ are SP-convergent. Since C and C' are axioms of SP, $t\sigma\rho'\{\equiv u\sigma\rho'\}$ is SP-convergent and $t\sigma\rho' = v[t'\tau/x]\rho' \longrightarrow_{SP} v[u'\tau/x]\rho'$. Since SP is confluent, $v[u'\tau/x]\rho'\{\equiv u\sigma\rho'\}$ and thus $v[u'\tau/x]\rho\{\equiv u\sigma\rho\}$ are SP-convergent.

[18] See Def. 9.22 for the equational case.
[19] See Def. 9.12 for the equational case.
[20] See Def. 9.23 for the equational case.
[21] In the equational case this theorem combines the *Buchberger-Newman Lemma* (see Lemma 9.1) with the *Knuth-Bendix Theorem* (see Theorem 9.9). In the ground equational case it agrees with Lemmas 7 and 8 of [Küc 89]. In the case of conditional-equational axioms without fresh variables it agrees with [DOS 88], Theorem 2, which, however, assumes termination and joinability in a stronger sense.

The "if"-part of the proof is carried out in [Pad 92] (Theorem 6.10). It is shown by induction on p along $>$ that all SP-convergent ground atoms p are strongly SP-convergent. Moreover, the proof depends on the fact that all Horn axioms of SP are subjoinable below their left-hand sides. ❑

Note that the axiom compatibility of $>$ is needed in this proof only for axioms with fresh variables. If, for instance, $fresh(C)$ were empty, we could choose $x\rho'$ as $x\rho$ for all $x \in var(C)$ and would get the same result.

One immediately obtains

Corollary 10.5.11 (confluence criterion I)[22] A terminating and weakly orthogonal specification is confluent. ❑

For instance, this criterion implies that the constructor subspecification of any standard specification is confluent.

In contrast to pure equational logic and rewriting we employ here rather complicated notions. This is necessary because standard specifications should cover non-trivial data types and functional-logic programs built upon them. Consequently, we must deal with *conditional* axioms and take into account *fresh variables* from the very begininning. We would cheat the user of the theory if we would restrict ourselves to the simple equational case and consider the incorporation of axiom premises and fresh variables a "straightforward" extension. Take a look at research papers on conditional rewriting with or without extra variables and you will face quite various notions of reduction, termination, critical pair, joinability, confluence and normal form (see, e.g., [BK 86], [JW 86], [Kap 87], [DOS 87], [DOS 88], [BG 89], [GW 93], [AL 94], [SMI 95]). The variety is due to the number of application areas for rewriting, which range from theorem proving and prototyping to programming and compiler construction.

Example 10.5.12 The following visible specification comprises typical functional-logic programs and thus should be functional in order to be amenable to efficient proof methods.

```
DIV&REP = ENTRY + NAT +
     vissorts       tree
     consts         leaf:entry->tree
                    _#_:tree*tree->tree
     defuns         div:nat*nat->(nat*nat)
                    rep&min:tree*nat->tree*nat
                    min:entry*entry->entry
     vars           m,n,q,r:nat  x,y,z:entry  T1,T2,U1,U2:tree
     Horn axioms    div(m,n) = (0,m)  <==  m < n
                    div(m,n) = (q+1,r)  <==  m >= n > 0 /\ div(m-n,n) = (q,r)
                    div(m,0) = bottom
                    rep&min(leaf(x),y) = (leaf(y),x)
                    rep&min(T1#T2,x) = (U1#U2,min(y,z))  <==  rep&min(T1,x) = (U1,y) /\
                                                              rep&min(T2,x) = (U2,z)
                    min(x,y) = x  <==  x <= y
                    min(x,y) = y  <==  y <= x

 EXT = DIV&REP +
     defuns         repByMin:tree->tree
     vars           x:entry  T1,T2:tree
     Horn axioms    repByMin(T1) = T2  <==  rep&min(T1,x) = (T2,x)
```

$div(m,n)$ returns the quotient of m and n and the remainder of the division. $repByMin(T)$ replaces all entries of the tree T by their minimum. DIV&REP can be implemented directly in a functional language such as

[22] See Theorem 9.6 for the equational case.

Standard ML [Pau 96]:

```
exception bottom
fun div(m,n) = if m < n then (0,m)
               else if m >= n > 0 then let val (q,r) = div(m-n,n)
                                       in (q+1,r) end
                                   else raise bottom
infix #
datatype tree = leaf of nat | # of tree*tree
fun rep&min(leaf(x),y) = (leaf(y),x)
|   rep&min(T1#T2,x) = let val (U1,y) = rep&min(T1,x)
                           val (U2,z) = rep&min(T2,y)
                       in (U1#U2,min(y,z)) end
```

$repByMin$ can be implemented directly only in a logic language that provides a mechanism for solving equations. In order to apply the axiom the premise equation $rep\&min(T_1,x) \equiv (T_2,x)$ must be solved in x and T_2, but no solution is obtained by simply rewriting $rep\&min(T_1,x)$. Pure logic programming would enforce us to replace defined functions by their input-output predicates:

$$\begin{array}{rcl} r_{rep\&min}(leaf(x),y,leaf(y),x) & & \\ r_{rep\&min}(T_1\#T_2,x,U_1\#U_2,min(y,z)) & \Leftarrow & r_{rep\&min}(T_1,x,U_1,y) \wedge r_{rep\&min}(T_2,x,U_2,z) \\ r_{repByMin}(T_1,T_2) & \Leftarrow & rep\&min(T_1,x,T_2,x). \end{array}$$

Then we could compute the solution. However, this means throwing out the baby with the bathwater. Reasoning about $repByMin$ on the basis of *these* axioms would prevent us from exploiting the otherwise obvious functional dependency between arguments of $r_{repByMin}$. The axioms for $r_{rep\&min}$ are a purely relational *implementation* of $repByMin$ as Pettorossi's program using λ-abstraction is a purely functional *implementation* of $repByMin$ [PS 87].

Given a ground term t, the equation $rep\&min(t,x) \equiv (T_2,x)$ has unique solution in the initial DIV&REP-model. This fact should be sufficient for concluding that EXT is functional if DIV&REP is so. ❑

The axioms for div and $rep\&min$ illustrate a syntactical criterion for confluence, which is similar to coinductivity (Def. 10.3.6). Axiom variables may "flow" only from the conclusion's left-hand side to left-hand sides of premise equations and from right-hand sides of premise equations to *subsequent* premise equations or "back" to the conclusion's right-hand side:

Definition 10.5.13 A Horn clause

$$f(t)\{\equiv u\} \quad \Leftarrow \quad t_1 \equiv u_1 \wedge \cdots \wedge t_n \equiv u_n \wedge H$$

is **deterministic up to** $Y \subseteq X$ if for $V_0 = var(t)$ and all $1 \le i \le n$, u_i is a normal form,

- $Def(u_i) \in H$ or $u_i \in X \cup \{\perp\}$,
- $var(t_i) \subseteq V_{i-1}$, $var(u) \subseteq V_n$ and $Y = var(H) \setminus V_n$ for $V_i = V_{i-1} \uplus var(u_i)$. ❑

A Horn clause without fresh variables is deterministic up to the empty set of variables: set n=0. Hence determinism generalizes the usual, but too restrictive (see above), assumption in conditional rewriting theory. In contrast to similar notions in [BG 89], [GW 93], [AL 94] and [SMI 95], our notion of a deterministic clause $f(t) \equiv u \Leftarrow G$ takes into consideration only those fresh variables that occur in u. A non-empty Y indicates that $f(t)$ rewrites to u if H is solvable in Y, but the solution is not part of the "result" u. This case must be admitted, in particular if H includes an atom $r(v,w)$ and, given an instance v' of v, the rewrite step should only depend on the *existence* of an instance w' of w such that $r(v',w')$ holds true.

The axioms for div, $rep\&min$ and min (see Ex. 10.5.12) are deterministic up to $\emptyset$, but

$$repByMin(T_1) \equiv T_2 \quad \Leftarrow \quad rep\&min(T_1, x) \equiv (T_2, x) \tag{1}$$

is not deterministic, not even up to $\{x\}$, as one might think at first sight. Although T_2 is "used", but x is not, both variables are "generated" simultaneously and thus the premise of (1) can be regarded—in terms of Def. 10.5.13—neither as $t_1 \equiv u_1$ nor as H. Clause (1) is a simple, but typical **network specification**.[23] While T_1 denotes an *input port* and T_2 denotes an *output port*, x is *fed back* from output to input. Fortunately, the function $repByMin$, which is to be defined by solutions of the premise of (1), does not occur there, too. Hence we can employ the fact that unique solutions are obtained already with respect to the *base* specification DIV&REP. In other words, the overlay induced by Clause (1) on itself and given by:

$$T_2 \equiv T_3 \quad \Leftarrow \quad rep\&min(T_1, x) \equiv (T_2, x) \wedge rep\&min(T_1, y) \equiv (T_3, y) \tag{2}$$

is an inductive DIV&REP-theorem.

Theorem 10.5.14 (confluence criterion II) [Pad 97] Let SP be terminating, AX be the set of Horn axioms of SP and SP' be a confluent subspecification of SP with signature Σ' and Horn axiom set AX' such that $NF_{\Sigma,s} = NF_{\Sigma',s}$ for all sorts $s \in \Sigma'$. SP is confluent if

(a) $f \in \Sigma \setminus \Sigma'$ for all $(f(t)\{\equiv u\} \Leftarrow H) \in AX \setminus AX'$,

(b) for all $C \in AX \setminus AX'$, C is deterministic or the overlay induced by C on itself is an inductive theorem of SP',

(c) all SP-overlays induced by two different clauses of $AX \setminus AX'$ are inductive theorems of SP'. ❑

To apply the criterion to Ex. 10.5.12 we first assume that NAT and (the instances of) ENTRY are confluent and set $SP' =$ ENTRY+NAT and $SP =$ DIV&REP. Then (a) holds true trivially. (b) is valid because the axioms for for div, $rep\&min$ and min are deterministic. The only two different axioms of $SP \setminus SP'$ that induce an overlay are the two axioms for min. It reads as follows:

$$x \equiv y \quad \Leftarrow \quad x \leq y \wedge y \leq x.$$

Since this is an axiom of ENTRY, we conclude (c). Hence Theorem 10.5.14 implies that DIV&REP is confluent. Secondly, we set $SP' =$ DIV&REP and $SP =$ EXT. Again, (a) holds true trivially. (b) is valid because (13) is an inductive SP'-theorem, which can be shown easily, e.g. by expanding induction (Section 10.7). (c) holds true trivially because no overlay is induced by two different axioms of $SP \setminus SP'$. Hence confluence criterion II implies that EXT is confluent.

Given a standard specification SP, the subspecification SP' consisting of the constructors, bottom constants, definedness predicates and constructor axioms of SP is always confluent. Hence this may always be the first choice for SP' when applying Theorem 10.5.14.

The Superposition Theorem and its corollaries 10.5.11 and 10.5.14 are the main reasons for requiring that SP is terminating. Though termination also implies completeness, it is mostly easier to show this condition directly. As a side-effect one may already use completeness when defining a reduction ordering, which is usually done inductively on the structure of terms and goals (see, e.g., Def. 9.10). If the definition splits into several cases, the transitivity of $>$ is often harder to prove than the well-foundedness. If SP is complete, then the following combination of an ordering on Σ with a term resp. proof ordering yields a reduction ordering for most standard specifications.

[23] See [Pad 92], Ex. 8.7, for a more complex example.

Theorem 10.5.15 (termination criterion) (see [Pad 92], Section 6.2) Suppose that SP is complete and all SP-reduced terms are normal forms. A binary relation on the set of defined functions and logical μ-predicates of Σ is defined as follows:

$$f >_\Sigma g \iff_{def} \exists : (f(t)\{\equiv t\} \Leftarrow H) \in AX \wedge g \text{ occurs in } u \text{ or } H.$$

SP is terminating if for all $C = (f(t)\{\equiv t\} \Leftarrow H) \in AX$, atoms or subterms $g(v)$ of u or H, $\sigma \in T_\Sigma^X$ and normal forms t', v' of $t\sigma$ resp. $v\sigma$ such that $g >_\Sigma^* f$, $H\sigma$ is convergent and $fresh(C)\sigma$ consists of normal forms, one of the following conditions holds true:

- f and g are function symbols and v' is smaller than t' with respect to a fixed well-founded ordering on NF_Σ.
- f and g are logical predicates and $g(v')$ has a shorter cut calculus proof than $f(t')$. ❑

We leave it to the reader to show that the termination criterion applies to the specification EXT of Ex. 10.5.12. The reduction ordering $>$ built up from the signature ordering $>_\Sigma$ satisfies $f(t) > u$ for all defined functions and logical μ-predicates f and all ground normal forms u. Note that this is necessary for $>$ to be compatible with axioms that involve fresh variables (see Def. 10.5.6).

To sum up, Corollary 10.5.11 and Theorems 10.5.14 and 10.5.15 provide us with criteria for confluence resp. termination and thus—via the Church-Rosser Theorem—for consistency (Corollary 10.5.4) and the equivalence of SP and $rel(SP)$ (Theorem 10.5.5).

Confluence criteria are also useful for proving the consistency of $Ini(SP)$ wrt $Ini(SP')$ where SP' is an arbitrary subspecification of SP (Def. 10.3.4). In Section 10.4, we pointed out that the more general consistency property comes up as proof obligation when two specifications are shown to be inductively equivalent or when implementations are shown to be correct. Hence we close this section by presenting four criteria for the consistency of $Ini(SP)$ wrt $Ini(SP')$, which impose increasingly weaker assumptions on SP.

Theorem 10.5.16 (consistency criteria II) [Pad 97] Let AX be the set of Horn axioms of SP and SP' be a subspecification of SP with signature Σ' and Horn axiom set AX'.

(1) **(proof by consistency)** Let $\Sigma = \Sigma'$.
$Ini(SP)$ is consistent wrt $Ini(SP')$
iff (by definition) $SP \vdash_{cut} p$ implies $SP' \vdash_{cut} p$ for all ground atoms p over SP'
iff $AX \setminus AX'$ consists of inductive theorems of SP'.

$Ini(SP)$ is consistent wrt $Ini(SP')$ if

(2) $s \in \Sigma \setminus \Sigma'$ for all function symbols $f : w \to s \in \Sigma \setminus \Sigma'$ and $r \in \Sigma \setminus \Sigma'$ for all $(r(t) \Leftarrow H) \in AX \setminus AX'$.

$Ini(SP)$ is consistent wrt $Ini(SP')$ if SP is functional, $f \in \Sigma \setminus \Sigma'$ for all $(f(t)\{\equiv u\} \Leftarrow H) \in AX \setminus AX'$ and

(3) $s \in \Sigma \setminus \Sigma'$ for all constructors $c : w \to s \in \Sigma \setminus \Sigma'$ **or**

(4) SP' is complete and all axioms of SP' are deterministic up to $\emptyset$. ❑

10.6 Implicit induction

Theorem 10.5.16(1) forbids signature extensions ($\Sigma = \Sigma'$). Hence it is less useful for showing consistency than for proving inductive theorems. But even in Case $\Sigma = \Sigma'$ consistency can be reduced to functionality—if $AX \setminus AX'$ is SP'-*reducible*:

Definition 10.6.1 A set CS of Horn clauses is (ground) SP**-reducible** if for all $(t\{\equiv u\} \Leftarrow H) \in CS$ and $\sigma \in T_\Sigma^X$ such that $H\sigma$ is $(SP \cup CS)$-convergent, $t\sigma$ is not SP-reduced. ❑

Lemma 10.6.2[24] Let CS be a set of Horn clauses such that $SP \cup CS$ is terminating.

(1) If $SP \cup CS$ is confluent and CS is SP-reducible, then $Ini(SP \cup CS)$ is consistent wrt $Ini(SP)$.

(2) If SP is confluent and $Ini(SP \cup CS)$ is consistent wrt $Ini(SP)$, then CS is SP-reducible. ❑

Theorem 10.5.10 reduces confluence to the subjoinability of critical clauses. Lemma 10.6.2 suggests a variant that characterizes confluence and *reductive* consistency simultaneously (Theorem 10.6.6). Given a subspecification SP' of SP, consistency of $Ini(SP)$ wrt $Ini(SP')$ becomes reductive consistency if $\vdash_{cut}$ is replaced by convergence:

Definition 10.6.3 Given a subspecification SP' of SP, SP is **reductively consistent wrt** SP' if for all ground atoms p over SP', $p \vdash_{SP} \emptyset$ implies $p \vdash_{SP'} \emptyset$. ❑

Proposition 10.6.4 (confluence criterion III) Given a confluent subspecification SP' of SP, SP is confluent if SP is reductively consistent wrt SP'.

Proof. Let u, u' be SP-reducts of a ground term t. Hence $t \equiv u$ and $t \equiv u'$ are SP-convergent. Since SP is reductively consistent wrt SP', both equations are SP'-convergent and thus SP'-joinable (Def. 10.5.1), i.e. $u \xrightarrow{*}_{SP'} v \xleftarrow{*}_{SP'} t \xrightarrow{*}_{SP'} v' \xleftarrow{*}_{SP'} u'$ for some v, v'. Since SP' is confluent, v and v' and thus u and u' are SP'-joinable. ❑

Secondly, we modify the definition of subjoinability (Def. 10.5.9):

Definition 10.6.5 (SP'-subjoinability) Given a subspecification SP' of SP and a binary relation $>$ on goals, a Horn clause $p \Leftarrow H$ is SP'**-subjoinable below** a term or atom t if

$$H\sigma \vdash_{SP} \emptyset \quad \Rightarrow \quad p\sigma \vdash_{SP} \emptyset$$

for all $\sigma \in T_\Sigma^X$ such that all SP-convergent ground goals $G < t\sigma$ are strongly SP-convergent and SP'-convergent.

Given two subsets A, B of AX, (A, B) is (SP', SP)**-convergent** if for all terms and atoms t, each SP-critical clause $u\{\equiv v\} \Leftarrow H$ induced by some $C \in A$ on some $C' \in B$ at t is SP'-subjoinable below $t\{\equiv v\}$. ❑

Theorem 10.6.6 (Superposition Theorem II) Let SP be terminating with reduction ordering $>$ and CS be a set of Horn clauses over Σ.

(a) $SP \cup CS$ is confluent and reductively consistent wrt SP if $>$ is compatible with CS and $(AX \cup CS, AX \cup CS)$ is $(SP, SP \cup CS)$-convergent.

(b) Let $SP \cup CS$ be confluent. If the $>$ is compatible with CS or CS has no fresh variables[25], then $(AX \cup CS, AX \cup CS)$ is $(SP, SP \cup CS)$-convergent.

Proof. Quite similar to the proof of Theorem 10.5.10. For (a), one shows by induction on p along $>$ that all $(SP \cup CS)$-convergent ground atoms p are strongly $(SP \cup CS)$-convergent and SP-convergent. ❑

The criterion for the confluence of $SP \cup CS$ in (a) can be weakened if SP is already confluent:

Lemma 10.6.7 [Pad 95] Let SP be confluent and CS be an SP-reducible set of Horn clauses such that $SP \cup CS$ is terminating. $(AX \cup CS, AX \cup CS)$ is $(SP, SP \cup CS)$-convergent if (CS, AX) is $(SP, SP \cup CS)$-convergent. ❑

[24] This generalizes [JKo 86], Theorem 1, to conditional axioms.
[25] See the remark following Theorem 10.5.10.

These results lead to a method for proving inductive theorems without explicit induction steps, provided that the conjectures are Horn clauses and there is a reduction ordering that is compatible with AX *and* CS (Def. 10.5.6):

Theorem 10.6.8 (implicit induction I) Let SP be confluent and terminating and CS be a set of Horn clauses over Σ.

(a) If CS is SP-reducible, (CS, AX) is $(SP, SP \cup CS)$-convergent and the reduction ordering for SP is compatible with CS, then CS consists of inductive theorems of SP.

(b) Let CS consist of inductive theorems of SP. If the reduction ordering for SP is compatible with CS or CS has no fresh variables, then (CS, AX) is $(SP, SP \cup CS)$-convergent.

Proof. (a) By Lemma 10.6.7, $(AX \cup CS, AX \cup CS)$ is $(SP, SP \cup CS)$-convergent. Hence by Theorem 10.6.6(a), $SP \cup CS$ is confluent and reductively consistent wrt SP and thus by the Church-Rosser Theorem, $Ini(SP \cup CS)$ is consistent wrt $Ini(SP)$. Hence Theorem 10.5.16(1) implies $CS \subseteq ITh(SP)$.

(b) By Theorem 10.5.16(1), $Ini(SP \cup CS)$ is consistent wrt SP. Since SP is confluent, the Church-Rosser Theorem implies that $SP \cup CS$ is reductively consistent wrt SP. Hence by Prop. 10.6.4, $SP \cup CS$ is confluent and thus by Theorem 10.6.6(b), (CS, AX) is $(SP, SP \cup CS)$-convergent. ❑

By (b), the previous theorem also allows us to disprove inductive conjectures. In contrast to forbidding fresh variables in AX, it might be reasonable to exclude them from CS. (b) says that in this case the reduction ordering need not be compatible with the conjectures.

Critical clauses of conjectures on axioms correspond to cases proofs by explicit induction split into. Consequently, the implicit-induction approaches of [Red 90], [KR 90] and [BKR 95] replace the notion of a critical clause of CS on AX by that of a cover or test set for CS, which, roughly said, subsumes all solutions of CS. The conjectures are proved by induction on the test set along a reduction ordering for SP (see, e.g., Section 9.6). One may construct the test set from the minimal substitutions that generate critical clauses of CS on AX. In contrast to Theorem 10.6.8(a), the corresponding validity criterion (Theorem 10.6.10) does not require the reduction ordering for SP to be compatible with CS. Instead, CS and all lemmas used in proofs that critical clauses are SP-subjoinable must be *bounded*:

Definition 10.6.9 Given a reduction ordering $>$ for SP, a clause $t\{\equiv u\} \Leftarrow H$ is **bounded by** an atom p if for all goals $G(x)$ and $\sigma, \tau \in T_\Sigma^X$, $p\sigma \geq G(t\tau)$ implies $p\sigma > (G(u\tau) \wedge H\tau)$. ❑

Theorem 10.6.10 (implicit induction II) [Pad 95] Suppose that SP is confluent and terminating and CS is a set of Horn clauses over Σ. For any Horn clause C, let $cc(C)$ be the set of $(SP \cup CS)$-critical clauses of C on AX and $Y(C) = var(cc(C)) \setminus var(C)$. If (CS, AX) is $(SP, SP \cup CS)$-convergent and for all $C = (t\{\equiv u\} \Leftarrow H) \in CS$,

$$\exists Y(C) \bigvee \{G \wedge \wedge_{x \in dom(\sigma)} x \equiv x\sigma \mid (v\{\equiv u\sigma\} \Leftarrow H\sigma \wedge G) \in cc(C)\}$$

is an inductive theorem of SP and C is bounded by all conclusions of clauses of CS, then CS consists of inductive theorems of SP. ❑

Let us finally present two criteria for the SP-subjoinability of a critical clause CC (Def. 10.6.5). The first one simply requires the inductive validity of CC. The second one demands that the conclusion of CC is reducible into the premise by applying inductive theorems of SP as well as bounded clauses of CS, which are thus used as implicit induction hypotheses!

Lemma 10.6.11 (criteria for SP-subjoinability) Let SP be confluent and terminating, CS be a set of Horn clauses over Σ and $CC = (u\{\equiv v\} \Leftarrow H)$ be an $(SP \cup CS)$-critical clause induced by some C on some C' at some t. CC is SP-subjoinable below $t\{\equiv v\}$ if C and C' are bounded by $t\{\equiv v\}$ and

(1) CC is an inductive theorem of SP or

(2) there is a subset DS of $ITh(SP) \cup CS$ consisting of clauses bounded by $t\{\equiv v\}$ such that for all $\sigma \in T_\Sigma^X$ there is a DS-reduction of $u\sigma\{\equiv v\sigma\}$ into a subset of $H\sigma$.

Proof. Let $\sigma \in T_\Sigma^X$ such that $H\sigma$ is $(SP \cup CS)$-convergent and all $(SP \cup CS)$-convergent ground goals $G < t\sigma\{\equiv v\sigma\}$ are SP-convergent and strongly $(SP \cup CS)$-convergent. Since C, C' are bounded by $t\{\equiv v\}$, we have $t\sigma\{\equiv v\sigma\} > (u\sigma\{\equiv v\sigma\} \wedge H\sigma)$. Hence $H\sigma$ is SP-convergent. In Case (1), the Church-Rosser Theorem implies that $u\sigma\{\equiv v\sigma\}$ is SP-convergent. In Case (2) there is a subset H' of σ such that $t\sigma\{\equiv v\sigma\} > u\sigma\{\equiv v\sigma\} \vdash_{DS} H' \vdash_{SP} \emptyset$. Suppose that

(a) for all ground goals G, $t\sigma\{\equiv v\sigma\} > G \vdash_{DS} \emptyset$ implies $G \vdash_{SP} \emptyset$.

Then $u\sigma\{\equiv v\sigma\}$ is SP-convergent as well.

We show (a) by induction on the length of a shortest DS-reduction R of G into $\emptyset$. There are $C = (l\{\equiv r\} \Leftarrow H'') \in DS$, a goal $G'(x)$ and $\tau \in T_\Sigma^X$ such that $G = G'(l\tau)$ and the shortest DS-reduction of $G'(r\tau) \wedge H''\tau$ into $\emptyset$ is shorter than R. Since C is bounded by $t\{\equiv v\}$, we have $t\sigma\{\equiv v\sigma\} > G'(r\tau) \wedge H''\tau$. Hence by induction hypothesis, $G'(r\tau) \wedge H''\tau$ is SP-convergent and thus SP-convergent and strongly $(SP \cup CS)$-convergent. If $C \in CS$, the convergence implies $G \vdash_{SP \cup CS} \emptyset$ and thus $G \vdash_{SP} \emptyset$ because $t\sigma\{\equiv v\sigma\} > G$. If C is an inductive theorem of SP, then $H''\tau \vdash_{SP} \emptyset$ implies $SP \vdash_{cut} l\tau\{\equiv r\tau\}$ and thus by the Church-Rosser Theorem, $l\tau\{\equiv r\tau\}$ is SP-convergent. Hence $G = G'(l\tau)$ is $(SP \cup CS)$-convergent because $G'(r\tau)$ is strongly $(SP \cup CS)$-convergent. Since $t\sigma\{\equiv v\sigma\} > G$, we conclude $G \vdash_{SP} \emptyset$. ❏

Theorems 10.6.8 and 10.6.10 provide—together with the above criteria for SP-subjoinability—two methods for proving inductive theorems implicitly. Similar procedures have been established by [Red 90], [KR 90] and [BKR 95] (see Section 9.6). In all these cases reduction orderings on formulas take over the rôle of induction orderings, which are typical for *explicit* induction. Any implicit-induction method is based on the equivalence of inductive validity and consistency (Theorem 10.5.16(1)), originally stated in [HH 82] and [KM 87]. Current research on explicit induction develops strategies for guiding proofs towards induction hypotheses by recognizing conjecture *skeletons* within subgoals and applying **wave rules** (see, e.g., [Hut 90], [BSHIS 93]), while implicit induction tries to make such strategies superfluous. However, the range of applications for implicit induction is quite limited because of a number of restrictions and assumptions such as the Horn clause form of conjectures and the involvement of reduction orderings in proofs. Comparative analyses of both approaches can be found in [Pad 92] and [BBH 94].

10.7 Unfolding and explicit induction

Let us now continue the presentation of Gentzen rules for building up Gentzen proofs, which we started in Section 10.4. First a couple of useful expansions of (conjunctions of) Gentzen clauses:

body splitting $$\frac{GS \Leftarrow HS \vee HS'}{(GS \Leftarrow HS) \wedge (GS \Leftarrow HS')} \Updownarrow$$

body reduction/extension $$\frac{GS(t) \Leftarrow H(t) \wedge t \equiv u}{GS(u) \Leftarrow H(u)} \Updownarrow$$

head success $$\frac{\emptyset \vee GS \ \Leftarrow \ HS}{TRUE} \Updownarrow$$

body failure $$\frac{GS \ \Leftarrow \ FALSE}{TRUE} \Updownarrow$$

subsumption $$\frac{\exists Y(G_1 \vee \cdots \vee G_n) \ \Leftarrow \ \exists Z(H_1 \vee \cdots \vee H_k)}{TRUE} \Updownarrow$$

if the premise $\exists Z(H_1 \vee \cdots \vee H_k)$ **subsumes** the conclusion $\exists Y(G_1 \vee \cdots \vee G_n)$, i.e. for all $1 \leq i \leq k$ there are $1 \leq j \leq n$ and a substitution σ such that $G_j\sigma_Y \subseteq H_i$ (see 10.2.2)

Intuitively, subsumption is a "syntactic implication" that generalizes the θ-subsumption of universally quantified disjunctions [Plo 70]. For instance, the goal $x \geq 0 \wedge \exists y(x < s(y))$ subsumes $\exists y(x < y)$.

Many non-inductive Gentzen proofs consist of head expansions, body contractions (Section 10.4) and a final subsumption of the head (= conclusion) by the body (= premise) of the Gentzen clause that was derived at latest. Head expansions and body contractions are applications of **goal set rules** each of which transforms either the premise or the conclusion of a Gentzen clause. Roughly said, a proof of a Gentzen clause $GS \Leftarrow HS$ that consists of head expansions and body contractions brings GS and HS "closer and closer together". Of course, it may happen that the expanded conclusion fails to meet the contracted premise, i.e. the latter will never subsume the former. Such failures can often be avoided if simplifications, which turn formulas into equivalent ones, are preferred to pure expansions or contractions (see Section 10.4).

The term "simplification" indicates that these rules often simplify the structure of their redices, i.e. the (sub)formulas they are applied to. Automated theorem provers usually apply simplification rules automatically. The proofs get shorter and more readable if simplifications are executed in the background and these steps are not recorded explicitly. As they are equivalence transformations, simplifications do not change the semantics (= set of solutions) of a formula, but they will modify the syntactic structure. This may lead to failures in cases where a rule was applicable before the modification, but not afterwards. For instance, an induction step can be performed only if an induction hypothesis has been generated. But an induction hypothesis can be recognized automatically only from its syntactic structure. Hence intermediate simplifications may block induction steps. Nevertheless, theorem provers can work reasonably fast only if they are equipped with **built-in simplifiers** that apply simplification rules automatically.

Simplifiers "optimize" theorem provers. They "massage" formulas for the purpose of speeding up the proof process. Since they perform in the background, the verifier may concentrate on those parts of a proof that need his guidance. In Section 10.4, we have discussed the analogy between program synthesis and proof construction. Both activities: designing a program and building a proof, are successful only if a reasonable balance between automatic support and user interaction is maintained. On the one hand, the set of inductive theorems of a specification SP is not recursively enumerable. On the other hand, restrictions of the formal setting that aim at enumerable or even decidable theories cuts down the range of applications for deductive methods considerably.

Rewriting-oriented theorem provers and goal solvers simplify formulas usually by **normalization**, i.e. by reducing terms and goals into reduced ones. Normalization also increases the efficiency of functional-logic language interpreters significantly [Han 94]. General-purpose provers like Isabelle [Pau 94] employ rewriting for simplifying formulas, for instance for removing propositional operators. The simplifier of our program verifier *Expander* [Pad 94] applies some of the simplification rules given below automatically and normalizes subgoals constructed from standard functions and predicates on natural numbers, lists, bags, sets and maps. Here normalization is actually (partial) evaluation with respect to fixed parameter models (see Section 10.1). Inference rules with built-in standard type evaluating/solving mechanisms are presented in, e.g., [Sti 85], [JKi 86] and [AB 92]. CLP (= constraint logic programming) integrates goal solvers tailored to standard types into logic-language interpreters or compilers. The efficiency of a theorem prover that uses partial evaluators is increased considerably if the formulas to be evaluated are compiled into, say, ML expressions, normalized by ML functions and then retranslated into the prover language. In principle, any simplification rule could be implemented in this way as long as the evaluating programs preserve the (initial/final) semantics of their input. The separation of simplification from higher proof issues such as induction steps makes theorem provers designable in a modular way. The simplifier can be developed and extended more or less independently of other parts of the prover.

In the sequel we suppose that SP is functional and behaviorally congruent. Fundamental simplification rules for Gentzen proofs over SP can be derived directly from "meta-theorems" stated and discussed in previous sections. They are concerned with different parts of the signature of SP: Lemma 10.3.9 with μ-predicates and defined functions, 10.3.10 with ν-predicates, Theorem 10.3.12 with complements, including hidden inequalities, Lemma 10.4.1 with visible equalities and inequalities, and Lemma 10.5.7 with ν-predicates turned into μ-predicates. All these results contribute to the correctness of the following Gentzen rules.

total atom unfolding $$\frac{\exists Y(r(t)\wedge G)}{\exists Z_1(t\equiv t_1\wedge H_1\wedge G)\vee\cdots\vee\exists Z_n(t\equiv t_n\wedge H_n\wedge G)}\Updownarrow$$

partial atom unfolding $$\frac{\exists Y(r(t)\wedge G)}{\exists Z_{i_1}(t\equiv t_{i_1}\wedge H_{i_1}\wedge G)\vee\cdots\vee\exists Z_{i_k}(t\equiv t_{i_k}\wedge H_{i_k}\wedge G)}\Uparrow$$

if r is a logical μ-predicate of SP, $\{r(t_i)\Leftarrow H_i\mid 1\le i\le n\}$ is the set of axioms for r, $Z_i=Y\uplus var(r(t_i)\Leftarrow H_i)$ and $\{i_1,\ldots,i_k\}\subseteq\{1,\ldots,n\}$

total term unfolding $$\frac{\exists Y G(f(t))}{\exists Z_1(t\equiv t_1\wedge H_1\wedge G(u_1))\vee\cdots\vee\exists Z_n(t\equiv t_n\wedge H_n\wedge G(u_n))}\Updownarrow$$

partial term unfolding $$\frac{\exists Y G(f(t))}{\exists Z_{i_1}(t\equiv t_{i_1}\wedge H_1\wedge G(u_{i_1}))\vee\cdots\vee\exists Z_{i_k}(t\equiv t_{i_k}\wedge H_n\wedge G(u_{i_k}))}\Uparrow$$

if f is a defined function of SP, $\{f(t_i)\equiv u_i\Leftarrow H_i\mid 1\le i\le n\}$ is the set of axioms for f, $Z_i=Y\uplus var(f(t_i)\equiv u_i\Leftarrow H_i)$ and $\{i_1,\ldots,i_k\}\subseteq\{1,\ldots,n\}$

normal form splitting $$\frac{c(t)\equiv c(u)\wedge G}{(t_1\equiv u_1\wedge\cdots\wedge t_n\equiv u_n\wedge G)\vee(c(t)\equiv c(u)\equiv\bot\wedge G)}\Updownarrow$$

$$\frac{c(t)\equiv\bot\wedge G}{(t_1\equiv\bot\wedge G)\vee\cdots\vee(t_n\equiv\bot\wedge G)}\Updownarrow$$

if c is a visible constructor, $t=(t_1,\ldots,t_n)$ and $u=(u_1,\ldots,u_n)$

normal form clash $$\frac{c(t)\equiv d(u)\wedge G}{c(t)\equiv d(u)\equiv\bot\wedge G}\Updownarrow$$ if c and d are different visible constructors

bottom removal $$\frac{c\equiv\bot\wedge G}{FALSE}\Updownarrow$$ if c is a visible constructor $$\frac{t\equiv\bot\wedge Def(t)}{FALSE}\Updownarrow$$

If there are many axioms for r or f, then total unfolding produces many goals. A clever implementation of these rules combines them with normal form splitting and clash and does not generate goals that would be removed by the latter rules almost immediately. In effect, these rule combinations are *simplication* instances of partial unfolding.

variable removal $$\frac{\exists Y(x\equiv t\wedge G(x))}{\exists Y\setminus\{x\}G(t)}\Updownarrow$$ if $x\in Y\setminus var(t)$

If x occurs more than once in $G(x)$ and t is not a normal form, then variable removal leads to multiple occurrences of t in $G(t)$ and thus to multiple redices for defined function unfolding. Hence it is reasonable to restrict the applications of this rule to normal forms t. Note that without this case of variable removal we could never eliminate valid, but non-reflective equations. The following rule generalizes variable removal from x to a term u, but, since u may be an arbitrary term, it is only an expansion rule.

matching rule $$\frac{\exists Y \uplus dom(\sigma)(u \equiv u\sigma \wedge G)}{\exists Y G\sigma} \Uparrow \quad \text{if } var(u) \cap var(u\sigma) = \emptyset$$

A special case is

reflection $$\frac{t \equiv t \wedge G}{G} \Updownarrow$$

The following rule can be regarded both as a kind of atom unfolding and as the *hidden*-term counterpart of normal form splitting.

hidden equation unfolding $$\frac{\exists Y(t \equiv u \wedge G)}{\exists Y \forall\{z_1,\dots,z_n\}(f_1(t,z_1) \equiv f_1(u,z_1) \wedge \cdots \wedge f_n(t,z_n) \equiv f_n(u,z_n) \wedge G)} \Updownarrow$$

if $s = sort(t)$ is hidden, $f_1,\dots,f_n$ are the s-destructors of SP and there are no s-transitions in SP (Def. 10.2.1)

The main steps of a proof by **context induction** [Hen 91], i.e. by induction on the size of contexts (see Section 10.3), are applications of hidden equation unfolding. If there are also relational s-destructors or s-transitions, then equations between s-sorted terms must be subjected to coinduction (see below).

Given a logical predicate r of SP, any cut calculus proof of a goal $G(r(t))$ must include at least one axiom for r. Taking this into account, we strengthen atom unfolding as follows.

strong atom unfolding $$\frac{\exists Y(r(t) \wedge G \wedge \{H \rhd r(t)\})}{\begin{array}{l}\exists Z_1(t \equiv t_1 \wedge H_1 \wedge \{H \rhd\} r(t) \rhd H_1 \wedge G) \vee \cdots \vee \\ \exists Z_n(t \equiv t_n \wedge H_n \wedge \{H \rhd\} r(t) \rhd H_n \wedge G)\end{array}} \Updownarrow$$

if r is a logical μ-predicate of SP, $\{r(t_i) \Leftarrow H_i \mid 1 \leq i \leq n\}$ is the set of axioms for r and $Z_i = Y \uplus var(r(t_i) \Leftarrow H_i)$

$\{$ and $\}$ enclose optional parts of a formula. $\rhd$ denotes the following ordering on goals:

$$G \rhd G' \quad \Longleftrightarrow_{def} \quad \text{Some cut calculus proof of } G' \text{ is shorter than the shortest proof of } G.$$

Unfolding rules apply only axioms. The following rules allow us to use an arbitrary Gentzen clause

$$\boxed{C \quad = \quad \exists Y(\{u_0 \equiv\}u_1 \vee \cdots \vee \{u_0 \equiv\}u_n) \;\Leftarrow\; \exists Z H}$$

with $Y \cap var(u_0) = \emptyset$ as a lemma within proofs of other theorems.

contracting lemma application $$\frac{\exists Z(G\{t_0\} \wedge H\sigma)}{\exists Y((G(t_1) \wedge \{t_0 \equiv\}t_1) \vee \cdots \vee (G(t_n) \wedge \{t_0 \equiv\}t_n))} \Downarrow$$

if for all $0 \leq i \leq n$, $t_i = u_i\sigma$ and $(Y \cup Z) \cap dom(\sigma) = \emptyset$

expanding lemma application $$\frac{\exists Y(G_1(t_1)\vee\cdots\vee G_n(t_n))}{\exists Y\uplus Z(H\sigma\wedge G_1\{u_0\}\sigma\wedge\cdots\wedge G_n\{u_0\}\sigma\wedge\bigwedge_{x\in V}x\equiv x\sigma)}\Uparrow$$

if V is the set of free variables of the rule premise and for all $1\leq i\leq n$, $t_i\sigma=u_i\sigma$ and $(Y\cup Z)\cap dom(\sigma)=Y\cap var(t_i)\cap var(G_i)=\emptyset$

If C is a Horn clause, say, $r(t)\Leftarrow H$, and r is a logical predicate, then expanding lemma application agrees with (input) **resolution** as known from logic programming and classical theorem proving. If C is a conditional equation, say, $t\equiv u\Leftarrow H$, then expanding lemma application coincides with **paramodulation**, which, in turn, generalizes rewriting (Def. 10.5.1) and narrowing (Def. 10.8.1). In contrast to unfolding, expanding lemma application can be applied only if certain parts of C are *unifiable* with corresponding parts of the rule premise (see Def. 10.2.2). Contracting lemma application is applicable only if parts of C *subsume* corresponding parts of the rule premise.

The lemma application rules can be turned easily into induction rules. The idea is to use a conjecture such as C above as a lemma in the proof of C. However, the actual instances of C applied in the proof must represent induction hypotheses. This is ensured as follows.

Let $\vec{x}$ be a list of the free variables of C and $\gg$: $sort(\vec{x})\times sort(\vec{x})$ be a μ-predicate of SP such that $var(u_0)$ is a subset of $\vec{x}$ and the final model of SP interprets $\gg$ as a well-founded relation. $\gg^{Fin(SP)}$ is supposed to be the induction ordering along which C shall be proved. It need not be known at the beginning of a proof, but should be supplied at latest when induction steps are to be executed. The variants of lemma application that apply C as an induction hypothesis and perform explicit induction steps read as follows. Let $\rho:\vec{x}\rightarrow X$ be a renaming of $\vec{x}$ away from $\vec{x}$ (Def. 10.2.2).

contracting term induction $$\frac{\exists Z(\vec{x}\gg\vec{x}\rho\sigma\wedge H\rho\sigma\wedge G\{t_0\})}{\exists Y((G(t_1)\wedge\{t_0\equiv\}t_1)\vee\cdots\vee(G(t_n)\wedge\{t_0\equiv\}t_n))}\Downarrow$$

if for all $0\leq i\leq n$, $t_i=u_i\rho\sigma$ and $(Y\cup Z)\cap dom(\sigma)=\emptyset$

expanding term induction $$\frac{\exists Y(G_1(t_1)\vee\cdots\vee G_n(t_n))}{\exists Y\uplus Z(\vec{x}\gg\vec{x}\rho\sigma\wedge H\rho\sigma\wedge G_1\{u_0\rho\}\sigma\wedge\cdots\wedge G_n\{u_0\rho\}\sigma\wedge\bigwedge_{x\in V}x\equiv x\sigma)}\Uparrow$$

if V is the set of free variables of the rule premise and for all $1\leq i\leq n$, $t_i\sigma=u_i\rho\sigma$ and $(Y\cup Z)\cap dom(\sigma)=Y\cap var(t_i)\cap var(G_i)=\emptyset$

Contracting induction assumes, expanding induction generates a **descent condition** $\vec{x}\gg\vec{x}\rho\sigma$, which indicates that the instance $C\sigma$ of the conjecture C is regarded as an induction hypothesis. Since descent conditions do not occur in C itself, they can only be generated in the course of proving C. Symmetrically, they will be removed eventually before the proof of C can be finished successfully. This means they are valid and hence the instances of C applied as lemmas have really been induction hypotheses. In other words, the above rules are correct *in the context of successful proofs of C*.[26] Of course, these rules also admit **simultaneous induction** where the conjecture comprises several Gentzen clauses, but each induction step uses only one of them.

Term induction covers all induction schemas that are based upon well-founded *data* orderings. Theorems provers employing term induction are, for instance, the Boyer-Moore prover [BM 79], INKA [Hut 90], CLAM [BSHIS 93], the Larch Prover [GH 93] and Isabelle [Pau 94]. Since data are presented as normal forms, data induction often reduces to **structural induction**[27], though it need not be the pure size of a normal form one induces upon.[28] The only conceptual alternative to structural induction is **computational induction** where

[26] For proofs of this conclusion, see [Pad 91], Theorem 5.2; [Pad 92], Theorem 5.5; or [Pad 96], Theorems 3.6 and 3.11.
[27] This was originally formulated in [Bur 69].
[28] The generalization of structural induction to arbitrary well-founded data orderings is also known as **Noetherian induction**.

one induces on the length of computations of a program or derivations of a formula, but not on data parts of the program or formula. A classical example of computational induction is Park's **fixpoint induction** [Par 69] for reasoning about partial-recursive functions. To prove a property of f one induces on the indices of functions f_i, $i \in \mathbb{N}$, that approximate f, similarly to the predicate $r' : nat \times w$ that approximates a ν-predicate r within a Horn clause completion (see Section 10.3).

Fixpoint induction is based on a variant of Kleene's fixpoint theorem (Theorem 10.2.14) where the powerset CPO (complete partial order) $\wp(A)$ is replaced by a CPO of partial functions. Park just employed the fixpoint property, while Morris [Mor 71] used the approximating functions f_i explicitly. The essence of his method—for which the notion "computational induction" was introduced—is, roughly said, to prove a goal $G(f(t))$ by showing $G(f_0(t))$ and $G(f_i(t)) \Rightarrow G(f_{i+1}(t))$. Suppose that f is specified by the axiom $f(x) \equiv y \Leftarrow H(f, x, y)$. Then $f_{i+1}(x) \equiv y \Leftarrow H(f_i, x, y)$ is the axiom for f_{i+1}. Moreover, a proof of $G(f_{i+1}(t))$ consists of a proof of $\exists y : (H(f_i, t, y) \wedge G(y))$ and a final application of the axiom for f_{i+1}. Since i correlates with the length of a proof of $G(f_i(t))$, classical computational induction is indeed an induction on proof lengths. With respect to cut calculus proofs, the corresponding induction ordering was defined above and denoted by $\rhd$. The following variants of term induction use $\rhd$ instead of a term ordering $\gg$.

contracting proof induction
$$\frac{\exists Z(H \rhd H\rho\sigma \wedge H\rho\sigma \wedge G\{t_0\})}{\exists Y((G(t_1) \wedge \{t_0 \equiv\}t_1) \vee \cdots \vee (G(t_n) \wedge \{t_0 \equiv\}t_n))} \Downarrow$$
if for all $0 \leq i \leq n$, $t_i = u_i\rho\sigma$ and $(Y \cup Z) \cap dom(\sigma) = \emptyset$

expanding proof induction
$$\frac{\exists Y(G_1(t_1) \vee \cdots \vee G_n(t_n))}{\exists Y \uplus Z} \Uparrow$$
$$(H \rhd H\rho\sigma \wedge H\rho\sigma \wedge G_1\{u_0\rho\}\sigma \wedge \cdots \wedge G_n\{u_0\rho\}\sigma \wedge \textstyle\bigwedge_{x \in V} x \equiv x\sigma)$$
if V is the set of free variables of the rule premise and for all $1 \leq i \leq n$, $t_i\sigma = u_i\rho\sigma$ and $(Y \cup Z) \cap dom(\sigma) = Y \cap var(t_i) \cap var(G_i) = \emptyset$

Probably the most important consequence of Theorem 10.3.5(1)-(3) is the soundness of the following inference rules. Let $r_1, \ldots, r_n$ be logical predicates of SP or $rel(SP)$ (Def. 10.2.15) and AX be the set of axioms for $r_1, \ldots, r_n$.

fixpoint induction
$$\frac{\bigwedge_{i=1}^n GS_i(x) \Leftarrow r_i(x)}{\exists q_1 \ldots \exists q_n : (\bigwedge_{C \in AX} C[q_1/r_1, \ldots, q_n/r_n] \ \wedge \ \bigwedge_{i=1}^n GS_i(x) \Leftarrow q_i(x))} \Updownarrow$$
if $r_1, \ldots, r_n$ are μ-predicates

coinduction
$$\frac{\bigwedge_{i=1}^n GS_i(x) \Rightarrow r_i(x)}{\exists q_1 \ldots \exists q_n : (\bigwedge_{C \in AX} C[q_1/r_1, \ldots, q_n/r_n] \ \wedge \ \bigwedge_{i=1}^n GS_i(x) \Rightarrow q_i(x))} \Updownarrow$$
if $r_1, \ldots, r_n$ are ν-predicates

These rules reflect the general induction resp. coinduction principles of set theory (see, e.g., [Gor 95]). When applying them one soon finds out that fixpoint induction is tailored to other kinds of conjectures than coinduction. The former handles conjectures $GS_i(x) \Leftarrow r_i(x)$ stating that GS_i holds true for all data satisfying r_i. In many cases, the μ-predicate r_i is the input-output predicate of a defined function f and GS_i describes the *desired* input-output relation of f. Coinduction, however, deals with inverse conjectures $GS_i(x) \Rightarrow r_i(x)$, which state that r_i holds true for all data specified by GS_i. Here GS_i is often solved (see Section 10.4), i.e. consists of equations $y \equiv t$ where t is a normal form (and y is a component of x). Then the conjecture says that all data that can be represented as instances of these normal forms satisfy the ν-predicate r_i. For bisimulation predicates r_i, coinduction becomes a rule for proving (even conditional) hidden equations. In order to match the rule premise, a hidden equation $H \Rightarrow t \equiv t'$ must first be turned into the clause $(H \wedge x \equiv t \wedge y \equiv t') \Rightarrow x \equiv y$ before the rule can be applied.

Both rules are simplifications. The downward implication $\Downarrow$ holds true because $Fin(SP)$ satisfies AX and thus we may define q_i as r_i. The upward implication $\Uparrow$ is valid because all solutions of AX in $r_1, \dots, r_n$ are super- resp. subsets of the interpretations of $r_1, \dots, r_n$ in $Fin(SP)$ because these are the least resp. greatest solutions of AX (Theorem 10.3.5). In other words, if the final model satisfies $C[q_1/r_1, \dots, q_n/r_n]$ for all $C \in AX$, then it also satisfies $q_i(x) \Leftarrow r_i(x)$ resp. $q_i(x) \Rightarrow r_i(x)$ for all $1 \leq i \leq n$. Hence the respective rule premise follows from the rule conclusion $GS_i(x) \Leftarrow q_i(x)$ resp. $GS_i(x) \Rightarrow q_i(x)$.

q_i is an existentially quantified predicate variable whose possible value ranges between GS_i and r_i (with respect to set inclusion). Choosing q_i strictly smaller resp. greater than GS_i means to **generalize** resp. **co-generalize** GS_i. This complies with the intuition that a smaller resp. greater relation expresses a stronger resp. weaker condition.

Example 10.7.1 (see Ex. 10.4.3)

```
PART2 = PARTITION +
    preds      part:list*list(list)
    vars       x,y:entry  L,L':list  P:list(list)
    Horn axioms
               part([x],[[x]])                                    (1)
               part(x::y::L,[x]::P)  <==  part(y::L,P)            (2)
               part(x::y::L,(x::L')::P)  <==  part(y::L,L'::P)    (3)
    conjects   L = flatten(P)  <==  part(L,P)                     (1)
```

$part(L,P)$ holds true iff P is a partition of L into successive sublists of L. If L is a singleton consisting of x, then $[[x]]$ is the only partition of L (Axiom 1). If L starts with x and y, then there are two ways of obtaining a partition of L from a partition P of $y :: L$: either one adds the singleton $[x]$ to the elements of P (Axiom 2) or one appends x to the first element L' of P and adds $x\&L'$ to P (Axiom 3). We show the correctness of $part$ by expanding Conjecture 1 (the colon connects Gentzen clauses conjunctively):

$$L \equiv flatten(P) \;\Leftarrow\; part(L,P)$$

fixpoint induction without generalization $(q = GS = (L \equiv flatten(P)))$

$$\vdash \quad [x] \equiv flatten([[x]]),$$
$$x :: y :: L \equiv flatten([x] :: P) \;\Leftarrow\; y :: L \equiv flatten(P),$$
$$x :: y :: L \equiv flatten((x :: L') :: P) \;\Leftarrow\; y :: L \equiv flatten(L' :: P)$$

body reduction

$$\vdash \quad [x] \equiv flatten([[x]]),$$
$$x :: flatten(P) \equiv flatten([x] :: P),$$
$$x :: flatten(L' :: P) \equiv flatten((x :: L') :: P)$$

partial $flatten$-term unfolding

$$\vdash \quad \exists\{L,P\}([[x]] \equiv L :: P \wedge [x] \equiv L@flatten(P)),$$
$$\exists\{L',P'\}([x] :: P \equiv L' :: P' \wedge x :: flatten(P) \equiv L'@flatten(P')),$$
$$\exists\{L'',P'\}((x :: L') :: P \equiv L'' :: P' \wedge x :: flatten(L' :: P) \equiv L''@flatten(P'))$$

total [ ]-term unfolding

$$\vdash \quad \exists\{L,P\}((x :: nil) :: nil \equiv L :: P \wedge x :: nil \equiv L@flatten(P)),$$
$$\exists\{L',P'\}((x :: nil) :: P \equiv L' :: P' \wedge x :: flatten(P) \equiv L'@flatten(P')),$$
$$\exists\{L'',P'\}((x :: L') :: P \equiv L'' :: P' \wedge x :: flatten(L' :: P) \equiv L''@flatten(P'))$$

normal form splitting, clash and variable removal

$\vdash \quad x :: nil \equiv (x :: nil)@flatten(nil),$
$x :: flatten(P) \equiv (x :: nil)@flatten(P),$
$x :: flatten(L' :: P) \equiv (x :: L')@flatten(P)$

partial @-term unfolding

$\vdash \quad x :: nil \equiv x :: flatten(nil),$
$x :: flatten(P) \equiv x :: flatten(P),$
$x :: flatten(L' :: P) \equiv x :: (L'@flatten(P))$

partial $flatten$-term unfolding

$\vdash \quad x :: nil \equiv x :: nil,$
$x :: flatten(P) \equiv x :: flatten(P),$
$x :: (L'@flatten(P)) \equiv x :: (L'@flatten(P))$

reflection and head success

$\vdash \quad TRUE$

An alternative—less straightforward—proof employing strong atom unfolding and contracting proof induction can be found in [Pad 97]. ❑

On the one hand, many proofs involving term induction can be simplified considerably if fixpoint induction is used instead. Since fixpoint induction generates them automatically, using this rule relieves us from interactively guiding the proof towards induction hypotheses, which is required if term induction shall be applied eventually. However, the fixpoint rules are not applicable to every Gentzen clause conjecture. For instance, if it has the form $GS(x) \Leftarrow (r(x) \wedge H(x))$, i.e. GS need not hold for all data satisfying r, but only for those that satisfy H, then we must either turn the conjecture into the "Curried form" $(GS(x) \Leftarrow H(x)) \Leftarrow r(x)$ or find axioms for a subpredicate r' of r such that $r'(x)$ is equivalent to $r(x) \wedge H(x)$. The first solution is simple, but may leave us with a quite complex conclusion of fixpoint induction because an implication has to be substituted for each occurrence of r in AX. Hence the second solution is more convenient, especially since program transformation rules may help us constructing axioms for r' from those for r. Moreover, the combination of such transformation rules with fixpoint rules would lead to proof rules for other conjecture patterns than those handled by the above fixpoint rules.

The rule for subgoal induction mentioned in Section 10.4 is a special case of fixpoint induction for two predicates r_1, r_2, namely the input-output predicates of f and $iter$, and where no generalization is involved ($q_i = GS_i$; see above). **Hoare induction** can be derived from subgoal induction by transforming subgoal invariants into Hoare invariants (see Section 10.4).

Example 10.7.2 (see Ex. 10.2.5). We show that the infinite stream 01 of alternating zeros and ones is fair by expanding the clause $fair(01)$:

$fair(01)$

body extension

$\vdash \quad s \equiv 01 \;\Rightarrow\; fair(s)$

coinduction with co-generalization $(q \neq GS = (s \equiv 01))$

$\vdash \quad q(s) \;\Rightarrow\; \exists\{L, s'\}(0 \notin L \wedge s \equiv L\#(0\&s') \wedge q(s')), \qquad s \equiv 01 \;\Rightarrow\; q(s)$

body reduction

$\vdash \quad q(s) \;\Rightarrow\; \exists\{L, s'\}(0 \notin L \wedge s \equiv L\#(0\&s') \wedge q(s')), \qquad q(01)$

total q-atom unfolding by applying the axiom $q(s) \Leftarrow (s \equiv 01 \vee s \equiv 10)$

$\vdash \quad (s \equiv 01 \vee s \equiv 10) \;\Rightarrow\; \exists\{L, s'\}(0 \notin L \wedge s \equiv L\#(0\&s') \wedge (s' \equiv 01 \vee s' \equiv 10)),$

$01 \equiv 01 \vee 01 \equiv 10$

body splitting, reflection and head success

$\vdash \quad s \equiv 01 \;\Rightarrow\; \exists\{L,s'\}(0 \notin L \wedge s \equiv L\#(0\&s') \wedge (s' \equiv 01 \vee s' \equiv 10)),$
$\quad s \equiv 10 \;\Rightarrow\; \exists\{L,s'\}(0 \notin L \wedge s \equiv L\#(0\&s') \wedge (s' \equiv 01 \vee s' \equiv 10))$

body reduction

$\vdash \quad \exists\{L,s'\}(0 \notin L \wedge 01 \equiv L\#(0\&s') \wedge (s' \equiv 01 \vee s' \equiv 10)),$
$\quad \exists\{L,s'\}(0 \notin L \wedge 10 \equiv L\#(0\&s') \wedge (s' \equiv 01 \vee s' \equiv 10))$

a Boolean transformation

$\vdash \quad \exists\{L,s'\}((0 \notin L \wedge 01 \equiv L\#(0\&s') \wedge s' \equiv 01) \vee (0 \notin L \wedge 01 \equiv L\#(0\&s') \wedge s' \equiv 10)),$
$\quad \exists\{L,s'\}((0 \notin L \wedge 10 \equiv L\#(0\&s') \wedge s' \equiv 01) \vee (0 \notin L \wedge 10 \equiv L\#(0\&s') \wedge s' \equiv 10))$

variable removal

$\vdash \quad \exists\{L\}((0 \notin L \wedge 01 \equiv L\#(0\&01)) \vee (0 \notin L \wedge 01 \equiv L\#(0\&10))),$
$\quad \exists\{L\}((0 \notin L \wedge 10 \equiv L\#(0\&01)) \vee (0 \notin L \wedge 10 \equiv L\#(0\&10)))$

partial #-term unfolding and variable removal

$\vdash \quad (0 \notin nil \wedge 01 \equiv 0\&01) \vee \exists\{L\}(0 \notin L \wedge 01 \equiv L\#(0\&10)),$
$\quad \exists\{L\}(0 \notin L \wedge 10 \equiv L\#(0\&01)) \vee \exists\{x,L\}(0 \notin x\&L \wedge 10 \equiv x\&(L\#(0\&10)))$

goal elimination

$\vdash \quad 0 \notin nil \wedge 01 \equiv 0\&01,$
$\quad \exists\{x,L\}(0 \notin x\&L \wedge 10 \equiv x\&(L\#(0\&10)))$

partial $\notin$-atom and total 10-term unfolding

$\vdash \quad 01 \equiv 0\&01,$
$\quad \exists\{x,L\}(0 \not\equiv x \wedge 0 \notin L \wedge 1\&01 \equiv x\&(L\#(0\&10)))$

hidden equation unfolding

$\vdash \quad head(01) \equiv head(0\&10) \wedge tail(01) \equiv tail(0\&10),$
$\quad \exists\{x,L\}(0 \not\equiv x \wedge 0 \notin L \wedge head(1\&01) \equiv head(x\&(L\#(0\&10))) \wedge tail(1\&01) \equiv tail(x\&(L\#(0\&10))))$

partial *head*-term and *tail*-term unfolding

$\vdash \quad 0 \equiv 0 \wedge 10 \equiv 10,$
$\quad \exists\{x,L\}(0 \not\equiv x \wedge 0 \notin L \wedge 1 \equiv x \wedge 01 \equiv L\#(0\&10))$

reflection, variable removal and head success

$\vdash \quad \exists\{L\}(0 \not\equiv 1 \wedge 0 \notin L \wedge 01 \equiv L\#(0\&10))$

partial $\not\equiv$-atom and #-term unfolding and variable removal

$\vdash \quad 0 \notin nil \wedge 01 \equiv 0\&10$

partial $\notin$-atom unfolding

$\vdash \quad 01 \equiv 0\&10$

hidden equation unfolding

$\vdash \quad head(01) \equiv head(0\&10) \wedge tail(01) \equiv tail(0\&10)$

partial *head*-term and *tail*-term unfolding

$\vdash \quad 0 \equiv 0 \wedge 10 \equiv 10$

reflection and head success

$\vdash \quad TRUE$ ❑

We were lucky here because INFSEQ-bisimilarity is defined only in terms of the *functional* destructors *head* and *tail* and thus we could reduce all hidden equations by hidden equation unfolding. In the general case we may use the following simplification rule that combines body extension with coinduction. Let s be a hidden sort, AX be the set of axioms of SP for $\sim$: ss and $t,t' \in T_\Sigma(X)_s$.

$$\textbf{hidden equation expansion} \qquad \frac{H \Rightarrow t \equiv t'}{\bigwedge_{C \in AX} C[q/\sim]} \Updownarrow \quad \text{if } Fin(SP) \text{ satisfies } \begin{cases} q(x,x) \\ q(t,t') \Leftarrow H \\ q(t',t) \Leftarrow H \end{cases}$$

For instance, based on a specification of finite and infinite streams, [Pad 97] presents a Gentzen proof of the hidden-type benchmark equation

$$filter(f, map(g,s)) \;\equiv\; map(g, filter(f \circ g, s)) \tag{1}$$

(see Ex. 10.3.8 and [Gor 95]). The proof starts with hidden equation expansion and proceeds with strong atom unfolding and contracting proof induction upon subconjectures. In contrast to Ex. 10.7.2, (1) is a statement about infinitely many streams. Hence both coinduction and proof induction seem to be indispensable for proving this conjecture.

10.8 Goal solving

Goal solving as a particular verification problem werde already mentioned in Section 10.4. Rules and methods for solving equations and other goals with respect to a functional specification are usually based on *narrowing* procedures, originally invented by [Sla 74] and [Hul 80]. Narrowing generalizes rewriting and resolution (Def. 10.5.1) insofar as narrowing—besides transforming a term or atom—generates a substitution that may solve a given goal. Usually only the generalization of rewriting is called narrowing, while the corresponding generalization of resolution as defined in Def. 10.5.1 yields the input resolution in the sense of automated theorem proving and logic programming. Resolution is the basic rule for solving goals that consist of logical predicates and normal forms. Narrowing adapts the resolution principle to goals that involve defined functions, in particular to sets of equations. Hence interpreters of functional-logic programming languages actually implement narrowing procedures [Han 94].

Unfolding and expanding lemma application also combine rewriting with substitution. But these rules do not separate the rewritten goal from the generated substitution. Instead, the latter is made part of the former: the substitution is presented as an (equational) subgoal. For the sake of clarity (and efficiency), solution procedures should better keep substitutions different from goals. Pairs of a goal and a substitution whose domain does not include variables of the goal are called constraints:

Definition 10.8.1 Let $SP = (\Sigma, AX)$ be a standard specification. A **Σ-constraint** $\langle G, \tau\rangle$ consists of a goal G and a substitution $\tau : X \to NF_\Sigma(X)$ such that $var(G) \cap dom(\sigma) = \emptyset$. The **narrowing calculus for** SP consists of the following rules for reducing Σ-constraints to Σ-constraints. Let G be a goal and σ be a be a minimal unifier of two terms t and t' and $(p \Leftarrow H) \in AX$ such that $H\sigma$ is SP-narrowable and $fresh(p \Leftarrow H)\sigma$ consists of normal forms.

$$\textbf{narrowing} \qquad \frac{\langle G(t), \tau\rangle}{\langle G(u)\sigma \wedge H\sigma, \tau\sigma_{var(G(t))}\rangle} \Uparrow \qquad \text{if } t \notin X \text{ and } p = (t' \equiv u)$$

$$\textbf{resolution} \qquad \frac{\langle r(t) \wedge G, \tau\rangle}{\langle H\sigma \wedge G\sigma, \tau\sigma_{var(r(t)\wedge G)}\rangle} \Uparrow \qquad \text{if } r \neq \equiv \text{ and } p = r(t')$$

$$\textbf{unification} \qquad \frac{\langle t \equiv t' \wedge G, \tau\rangle}{\langle G\sigma, \tau\sigma\rangle} \Uparrow$$

A sequence $cs_1, \ldots, cs_n$ of Σ-constraints such that for all $1 \le i < n$, cs_{i+1} is obtained from cs_i by applying one of the above rules, is called an SP-**narrowing reduction of** cs_1 **into** cs_n and we write $cs_1 \vdash_{\sqrt{SP}} cs_n$.

Given a goal G, $\tau : X \to NF_\Sigma(X)$ is a **narrowing solution** of G if $\langle G, id\rangle \vdash\!\!\bigvee_{SP}\langle\emptyset,\tau\rangle$. G is **SP-narrowable** if $\langle G, id\rangle \vdash\!\!\bigvee_{SP} cs$ for some cs. Otherwise G is **SP-narrowed.** ❑

The arrow $\Uparrow$ indicates that the rules of the narrowing calculus are—like those of the reduction calculus—expansions. Since rule premises and conclusions are constraints, the expansion property here means:

$$\langle G,\tau\rangle \vdash\!\!\bigvee_{SP}\langle G',\tau'\rangle \quad\Rightarrow\quad Ini(SP) \models G\tau' \Leftarrow G'. \tag{1}$$

Hence, in particular,

$$\langle G, id\rangle \vdash\!\!\bigvee_{SP}\langle G',\tau\rangle \quad\Rightarrow\quad SP \vdash_{cut} G\tau \tag{2}$$

(see (1) in Section 10.5).

The relationship between $\vdash_{SP}$ (Def. 10.5.1) and $\vdash\!\!\bigvee_{SP}$ is as follows: If $\langle G,\tau\rangle \vdash\!\!\bigvee_{SP}\langle G',\tau'\rangle$, then $G\sigma \vdash_{SP} G'$ for σ with $\tau\sigma = \tau'$. Conversely, given $\tau : X \to NF_\Sigma(X)$, if $G\tau$ is SP-convergent, then τ is a narrowing solution of G.

Let us again suppose that $SP = (\Sigma, AX)$ is functional and behaviorally congruent. The Church-Rosser Theorem 10.5.3 tells us that functionality implies not only that the reduction calculus is complete, but also that the narrowing calculus is **solution complete**, i.e. for all goals G and ground normal substitutions τ,

$$SP \vdash_{cut} G\tau \quad\Rightarrow\quad \langle G, id\rangle \vdash\!\!\bigvee_{SP}\langle\emptyset,\tau\rangle. \tag{3}$$

Hence goals can be disproved with narrowing and resolution: if each narrowing reduction of $\langle G, id\rangle$ is either infinite or ends with a constraint $\langle H,\tau\rangle$ such that H is SP-narrowed, then G is unsolvable in $Ini(SP)$:

$$\forall\tau : \langle G, id\rangle \not\vdash\!\!\bigvee_{SP}\langle\emptyset,\tau\rangle \quad\Rightarrow\quad \forall\tau : SP \not\vdash_{cut} G\tau.$$

Since narrowing reductions are expansions, a Gentzen clause $GS \Leftarrow HS$ may sometimes be proved by establishing narrowing reductions of all constraints $\langle G, id\rangle$ with $G \in GS$ into, say, $\langle G_1,\tau_1\rangle,\dots,\langle G_n,\tau_n\rangle$ and then expanding the goal set

$$(G_1 \wedge \bigwedge_{x\in dom(\tau_1)} x \equiv x\tau_1) \vee \cdots \vee (G_n \wedge \bigwedge_{x\in dom(\tau_n)} x \equiv x\tau_n) \tag{4}$$

into HS or contracting HS into (4) by applying Gentzen rules. This also shows the similarity between narrowing and unfolding (Section 10.7). On the one hand, narrowing improves over unfolding because the latter might generate a number of invalid equations that are rejected by narrowing if their respective sides are not unifiable. On the other hand, unfolding applies all axioms for a function or predicate in parallel, while narrowing considers only one axiom in each step. Applying all axioms simultaneously is crucial for the contraction property of unfolding. A sequence of unfolding steps yields the levels of a proof tree whose nodes are constraints and whose paths are narrowing reductions. A narrowing procedure also builds up such **narrowing trees** because, given a goal G, the procedure shall derive a **complete set** Sol **of narrowing solutions** of G such that each ground solution of G is subsumed by some $\sigma \in Sol$. In general, the outdegree of a narrowing tree increases both with the number of narrowable subterms resp. atoms of a goal and with the number of axioms that are applicable to the same redex. However, the latter agrees with the number of possible overlays at a single redex (Def. 10.5.8), and this number is usually small, if not one.

Hence the efficiency of a narrowing procedure depends on the number of redices (subterms or atoms) of the goal to be solved. A **redex selector** R assigns to each narrowable goal a single redex. The intention is to figure out *complete* redex selectors, which distinguish proper subtrees of a narrowing tree T each of which returns (at its leaves) a complete set of narrowing solutions of the goal at the root of T.

Definition 10.8.2 Given an atom or a term $t \notin X$ and a goal G with a unique occurrence of the variable x, the expression $G \bullet t$ is a **position** of the goal $G[t/x]$. For a substitution σ, $(G \bullet t)\sigma$ stands for the position $G\sigma_{X\setminus\{x\}} \bullet t\sigma$ (Def. 10.2.2).

Let σ be a minimal unifier of two terms t and t' and $(p \Leftarrow H) \in AX$ such that $H\sigma$ is SP-narrowable and $fresh(p \Leftarrow H)\sigma$ consists of normal forms.

- If $t \notin X$ and $p = (t' \equiv u)$, then $\langle G(u)\sigma \wedge H\sigma, \sigma_{var(G(t))}\rangle$ is a **narrowing reduct of** $G \bullet t$.
- If $p = r(t')$ and $r \not\equiv \equiv$, then $\langle H\sigma \wedge G\sigma, \sigma_{var(r(t)\wedge G)}\rangle$ is a **narrowing reduct of** $(x \wedge G) \bullet r(t)$.
- $\langle G\sigma, \sigma\rangle$ is a **narrowing reduct of** $(x \wedge G) \bullet (t \equiv t')$.

A position $G \bullet t$ with a narrowing reduct is a **narrowing redex of** $G[t/x]$. A function Sel that assigns to each SP-narrowable goal G a redex of G is a **redex selector**. Sel is **uniform** if $Sel(G)\sigma = Sel(G\sigma)$ for all goals G and ground normal substitutions σ. ❑

A redex selector prunes narrowing trees with respect to their breadth, but does not reduce their depth. The latter is accomplished if narrowing reducts are *simplified* before they are committed to further narrowing steps (see Section 10.7). Here simplification mostly means **normalization**, i.e. reduction into reduced goals, normal form splitting, clash and variable removal (see, e.g., [Han 94]). Since any simplification rule turns a formula into an equivalent one, simplifications can always be used in order to shorten narrowing reductions. However, simplifiers may also remove successful narrowing reductions whose resulting solutions are not achieved on other paths of the narrowing tree. This effect can be avoided if simplifications are restricted to *reductive* ones, i.e. those that comply with a reduction ordering for SP.

Definition 10.8.3 A function that assigns to each goal G a goal $Sim(G)$ such that $G \Leftrightarrow Sim(G)$ is an inductive theorem of SP is a **simplifier**. Given a reduction ordering $>$ for SP (Def. 10.5.6), Sim is $>$-**reductive** if $G\sigma \geq Sim(G)\sigma$ for all goals G and ground normal substitutions σ.

Given a redex selector Sel and a simplifier Sim, let $succs$ be the function from constraints to sets of constraints, defined by

$$succs^{Sel}_{Sim}\langle G, \tau\rangle \;=\; \{\langle Sim(G'), \tau\sigma\rangle \mid \langle G', \sigma\rangle \text{ is a narrowing reduct of } Sel(G)\}.$$

A narrowing reduction $cs_1, \ldots, cs_n$ such that for all $1 \leq i < n$, $cs_{i+1} \in succs^{Sel}_{Sim}(cs_i)$, is called an (Sel, Sim)-**narrowing reduction of** cs_1 **into** cs_n and we write $cs_1 \vdash^{Sel}_{Sim} cs_n$. ❑

Theorem 10.8.4 ([Pad 96], Lemmas 10.4 and 10.9) Let Sel be a redex selector and Sim be a simplifier. For all constraints $\langle G, id\rangle$ and $\sigma : X \to NF_\Sigma(X)$, $G\sigma$ is SP-convergent if $\langle G, id\rangle \vdash^{Sel}_{Sim} \langle\emptyset, \tau\rangle$. Moreover, if Sel is uniform, $>$ is a reduction ordering for SP and Sim is $>$-reductive, then the converse holds true for all ground normal substitutions τ, more precisely:

$$G\tau \vdash_{SP} \emptyset \quad \Rightarrow \quad \exists\sigma \leq \tau : \langle G, id\rangle \vdash^{Sel}_{Sim} \langle\emptyset, \sigma\rangle$$

where $\leq$ denotes the subsumption relation on substitutions (Def. 10.2.2). ❑

From this and the Church-Rosser Theorem (10.5.3) we conclude that the set of (Sel, Sim)-narrowing reductions is solution complete, i.e. for all goals G and ground normal substitutions τ,

$$SP \vdash_{cut} G\tau \quad \Rightarrow \quad \exists\sigma \leq \tau : \langle G, id\rangle \vdash^{Sel}_{Sim} \langle\emptyset, \sigma\rangle \tag{5}$$

(compare (3)), provided that the assumptions of Theorem 10.8.4 hold true. By combining (2) with (5) we obtain the following refutation rule :

$$(Sel, Sim)\textbf{-refutation} \qquad \frac{G}{FALSE} \Updownarrow \quad \text{if } Sel \text{ is a uniform redex selector, } Sim \text{ is a } >\text{-reductive simplifier and } \langle G, id\rangle \not\vdash^{Sel}_{Sim} \langle \emptyset, \sigma\rangle \text{ for all } \sigma : X \to NF_\Sigma(X)$$

Narrowing procedures select either innermost or outermost redex positions (see, e.g., [Fri 85], [AEH 94]). The completeness of SP implies that innermost redex selectors are uniform (see [Pad 96], Lemma 9.3). Choosing innermost redices becomes inefficient if many of them occur in unsolvable subgoals, which are not reducible into the empty goal and thus do not contribute to the derivation of solutions. With outermost redex selection many of these subgoals will have been removed before the redex selector meets them. Unfortunately, outermost redex selectors are not uniform unless additional constraints are imposed on the axioms, e.g. **non-sub-unifiability** (see [Ech 88] and [Pad 88], Section 8.4). This is a rather restrictive condition that forbids axiom premises. **Needed narrowing** (see [AEH 94]) overcomes the condition by replacing narrowing with pure substitution whenever an outermost narrowing step fails because a normal form t' in the axiom to be applied is not unifiable with a non-normal form t in the goal to be narrowed. The substitution step ensures that the want-to-be redex is at least instantiated. Then a new redex is selected, t will eventually be reduced to a normal form and again compared with t'. If both terms t and t' are normal forms, then either they are unifiable and the narrowing step is carried out or we have a normal form clash and thus may reject the application of the given axiom to the given goal position without even instantiating the goal (see [Pad 94], Section 4.4; [Pad 96], Section 9).

Whether the redex selector works inner- or outermost, a smart simplifier, which detects and removes unsolvable goals as early as possible, decreases the number of redices occurring in the course of a reduction significantly. Narrowing and simplification are also combined with **residuation** and **lazy evaluation** where unification steps are delayed until certain variables of the terms to be unified have been instantiated "sufficiently" (see [LFA 93], [Han 95]). Term and atom unfolding perform lazy unification in the sense that these rules generate an equation E as a subgoal, while narrowing would try to unify the two sides of E. With respect to (Sel, Sim)-narrowing, a complementary unfold rule performing eager unification reads as follows. It correctness follows from [Pad 96], Lemmas 10.5 and 10.10.

$$(Sel, Sim)\textbf{-unfolding} \qquad \frac{G}{\exists Y((G_1 \wedge \bigwedge_{x\in dom(\tau_1)} x \equiv x\tau_1) \vee \cdots \vee (G_n \wedge \bigwedge_{x\in dom(\tau_n)} x \equiv x\tau_n))} \Updownarrow$$

if Sel is a uniform redex selector, Sim is a $>$-reductive simplifier, $succ^{Sel}_{Sim}\langle G, id\rangle = \{\langle G_1, \tau_1\rangle, \ldots, \langle G_n, \tau_n\rangle\}$, $Y = \uplus_{i=1}^n var(dom(\tau_i)\tau_i)$ and $Y \cap (var(G) \cup dom(\tau_i)) = \emptyset$

Atom and term unfolding as defined Section 10.7 applies all axioms for a function or predicate f to an arbitrary redex of the form $G \bullet f(t)$. (Sel, Sim)-unfolding applies from these axioms only those whose left-hand sides are unifiable with the redex selected by Sel and discards the others. Some of the discarded axioms can only be rejected by atom and term unfolding if these rules are combined with normal form splitting and clash.

References

[AM 82] M.A. Arbib, E.G. Manes, *Parametrized Data Types Do Not Need Highly Constrained Parameters*, Information and Control 52 (1982) 139-158

[AB 92] J. Avenhaus, K. Becker, *Conditional Rewriting modulo a Built-in Algebra*, SEKI Report SR-92-11, University of Kaiserslautern (1992)

[AL 94] J. Avenhaus, C. Loría-Sáenz, *On Conditional Rewrite Systems with Extra Variables and Deterministic Logic Programs*, Proc. LPAR '94, Springer LNAI 822 (1994) 215-229

[AEH 94] S. Antoy, R. Echahed, M. Hanus, *A Needed Narrowing Strategy*, Proc. 21st ACM POPL (1994) 268-279

[Apt 90] K.R. Apt, *Logic Programming*, in: J. van Leeuwen, ed., Handbook of Theoretical Computer Science, Elsevier (1990) 493-574

[ABW 88] K.R. Apt, H.A. Blair, A. Walker, *Towards a Theory of Declarative Knowledge*, in: J. Minker, ed., Deductive Databases and Logic Programming, Morgan Kaufmann (1988) 89-148

[AW 89] E. Astesiano, M. Wirsing, *Bisimulation in Algebraic Specifications*, in: H. Ait-Kaci, M. Nivat, eds., Resolution of Equations in Algebraic Structures 1, Academic Press (1989) 1-31

[ARS 96] http://www-formal.stanford.edu/clt/ARS/systems.html (contains links to almost all currently available automated reasoning systems)

[BBH 94] R. Barnett, D. Basin, J. Hesketh, *A Recursion Planning Analysis of Inductive Completion*, Report MPI-I-94-230, Max-Planck-Institut für Informatik Saarbrücken 1994

[BK 86] J.A. Bergstra, J.W. Klop, *Conditional Rewrite Rules: Confluence and Termination*, J. Computer and System Sciences 32 (1986) 323-362

[BG 89] H. Bertling, H. Ganzinger, *Completion-Time Optimization of Rewrite-Time Goal Solving*, Proc. RTA '89, Springer LNCS 355 (1989) 45-58

[BHW 95] M. Bidoit, R. Hennicker, M. Wirsing, *Behavioural and Abstractor Specifications*, Science of Computer Programming 25 (1995) 149-186

[BKR 95] A. Bouhoula, E. Kounalis, M. Rusinowitch, *Automated Mathematical Induction*, J. Logic and Computation 5 (1995) 631-668

[BM 79] R.S. Boyer, J.S. Moore, *A Computational Logic*, Academic Press 1979

[BSHIS 93] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, A. Smaill, *Rippling: A Heuristic for Guiding Inductive Proofs*, J. Artificial Intelligence 62 (1993) 185-253

[BW 82] M. Broy, M. Wirsing, *Partial Abstract Types*, Acta Informatica 18 (1982) 47-64

[Bur 69] R.M. Burstall, *Proving Properties of Programs by Structural Induction*, Comput. J. 12 (1969) 41-48

[DOS 87] N. Dershowitz, M. Okada, G. Sivakumar, *Confluence of Conditional Rewrite Systems*, Proc. CTRS '87, Springer LNCS 308 (1988) 31-44

[DOS 88] N. Dershowitz, M. Okada, G. Sivakumar, *Canonical Conditional Rewrite Systems*, Proc. CADE '88, Springer LNCS 310 (1988) 538-549

[Ech 88] R. Echahed, *On Completeness of Narrowing Strategies*, Proc. CAAP '88, Springer LNCS 299 (1988) 89-101

[EM 85] H. Ehrig, B. Mahr, *Fundamentals of Algebraic Specification 1*, Springer 1985

[Fri 85] L. Fribourg, *Handling Function Definitions through Innermost Superposition and Rewriting*, Proc. RTA '85, Springer LNCS 202 (1985) 325-344

[GW 93] H. Ganzinger, U. Waldmann, *Termination Proofs of Well-Moded Logic Programs via Conditional Rewrite Systems*, Proc. CTRS '92, Springer LNCS 656 (1993) 430-437

[GGM 76] V. Giarratana, F. Gimona, U. Montanari, *Observability Concepts in Abstract Data Type Specifications*, Proc. MFCS '76, Springer LNCS 45 (1976) 576-587

[GD 94] J.A. Goguen, R. Diaconescu, *Towards an Algebraic Semantics for the Object Paradigm*, Proc. 9th ADT Workshop, Springer LNCS 785 (1994) 1-29

[GM 97] J.A. Goguen, G. Malcolm, *A Hidden Agenda*, UCSD Technical Report CS97-538, 1997, http://www-cse.ucsd.edu/users/goguen/ps/ha.ps.gz

[Gor 95] A.D. Gordon, *A Tutorial on Co-induction and Functional Programming*, Proc. Functional Programming Glasgow 1994, Springer (1995) 78-95

[GV 92] J.F. Groote, F. Vaandrager, *Structured Operational Semantics and Bisimulation as a Congruence*, Information and Computation 100 (1992) 202-260

[GH 93] J.V. Guttag, J.J. Horning, *Larch: Languages and Tools for Formal Specification*, Springer 1993

[Han 94] M. Hanus, *The Integration of Functions into Logic Programming: From Theory to Practice*, J. Logic Programming 19/20 (1994) 583-628

[Han 95] M. Hanus, *Analysis of Residuating Logic Programs*, J. Logic Programming 24 (1995) 219-245

[Hen 91] R. Hennicker, *Context Induction: A Proof Principle for Behavioural Abstractions*, Formal Aspects of Computing (1991) 326-345

[HH 82] G. Huet, J.M. Hullot, *Proofs by Induction in Equational Theories with Constructors*, J. Computer and System Sciences 25 (1982) 239-266

[Hul 80] J.M. Hullot, *Canonical Forms and Unification*, Proc. CADE '80, Springer LNCS 87 (1980) 318-334

[Hut 90] D. Hutter, *Guiding Induction Proofs*, Proc. CADE '90, Springer LNCS 449 (1990) 147-161

[JR 97] B. Jacobs, J. Rutten, *A Tutorial on (Co)Algebras and (Co)Induction*, EATCS Bulletin 62 (June 1997) 222-259

[JKi 86] J.-P. Jouannaud, H. Kirchner, *Completion of a Set of Rules Modulo a Set of Equations*, SIAM J. Computing 15 (1986) 1155-1194

[JKo 86] J.-P. Jouannaud, H. Kounalis, *Automatic Proofs by Induction in Equational Theories without Constructors*, Proc. LICS '86 (1986) 358-366

[JW 86] J-P. Jouannaud, B. Waldmann, *Reductive Conditional Term Rewriting Systems*, Proc. Conf. Formal Description of Programming Concepts III, North-Holland (1986) 223-244

[Kam 83] S. Kamin, *Final Data Type Specifications: A New Data Type Specification Method*, ACM TOPLAS 5 (1983) 97-123

[Kap 87] S. Kaplan, *Simplifying Conditional Term Rewriting Systems: Unification, Termination and Confluence*, J. Symbolic Computation 4 (1987) 295-334

[KM 87] D. Kapur, D.R. Musser, *Proof by Consistency*, Artificial Intelligence 31 (1987) 125-157

[KR 90] E. Kounalis, M. Rusinowitch, *Mechanizing Inductive Reasoning*, EATCS Bulletin 41 (1990) 216-226

[Küc 89] W. Küchlin, *Inductive Completion by Ground Proof Transformation*, in: H. Ait-Kaci, M. Nivat, eds., Resolution of Equations in Algebraic Structures, Vol. 2, Academic Press (1989) 211-244

[Lar 88] K.G. Larsen, *Proof Systems for Hennessy-Milner Logic with Recursion*, Proc. CAAP '88, Springer LNCS 299 (1988) 215-230

[LNS 82] J.-L. Lassez, V.L. Nguyen, E.A. Sonenberg, *Fixed Point Theorems and Semantics: A Folk Tale*, Information Processing Letters 14 (1982) 112-116

[LS 87] J. Loeckx, K. Sieber, *The Foundations of Program Verification*, 2nd ed., Teubner 1987

[LFA 93] R. Loogen, F.L.Fraguas, M.R. Artalejo, *A Demand Driven Computation Strategy for Lazy Narrowing*, Proc. PLILP '93, Springer LNCS 714 (1993) 184-200

[MG 85] J. Meseguer, J.A. Goguen, *Initiality, Induction and Computability*, in: M. Nivat, J. Reynolds, eds., Algebraic Methods in Semantics, Cambridge University Press (1985) 459-541

[MTW 88] B. Möller, A. Tarlecki, M. Wirsing, *Algebraic Specifications of Reachable Higher-Order Algebras*, Proc. 5th ADT Workshop, Springer LNCS 332 (1988) 154-169

[MPS 93] B. Möller, H. Partsch, S. Schuman, eds., *Formal Program Development*, IFIP WG 2.1 State-of-the-Art Report, Springer LNCS 755 (1993)

[Mor 71] J.H. Morris, *Another Recursion Induction Principle*, Communications ACM 14 (1971) 351-354

[MW 77] J.H. Morris, B. Wegbreit, *Subgoal Induction*, Communications ACM 20 (1977) 209-222

[MR 94] S. Muggleton, L. de Raedt, *Inductive Logic Programming: Theory and Methods*, J. Logic Programming 19/20 (1994) 629-679

[Pad 88] P. Padawitz, *Computing in Horn Clause Theories*, Springer 1988

[Pad 91] P. Padawitz, *Inductive Expansion: A Calculus for Verifying and Synthesizing Functional and Logic Programs*, J. Automated Reasoning 7 (1991) 27-103

[Pad 92] P. Padawitz, *Deduction and Declarative Programming*, Cambridge University Press 1992

[Pad 94] P. Padawitz, *Expander: A System for Testing and Verifying Functional-Logic Programs*, Report, FB Informatik, University of Dortmund 1994-1996, http://ls5.cs.uni-dortmund.de/~peter/ExpaTex.ps.gz

[Pad 95] P. Padawitz, *Swinging Data Types: Syntax, Semantics, and Theory*, Proc. WADT '95, Springer LNCS 1130 (1996) 409-435; extended version: http://ls5.cs.uni-dortmund.de/~peter/Swing.ps.gz

[Pad 96] P. Padawitz, *Inductive Theorem Proving for Design Specifications*, J. Symbolic Computation 21 (1996) 41-99

[Pad 97] P. Padawitz, *Theory of Programming*, German course notes, FB Informatik, University of Dortmund 1997, http://ls5.cs.uni-dortmund.de/~peter/TdP96.ps.gz

[Pad 97a] P. Padawitz, *Towards the One-Tiered Design of Data Types and Transition Systems*, Proc. WADT '97, Springer LNCS; extended version: *How to Integrate Transition Systems into Hidden Types*, http://ls5.cs.uni-dortmund.de/~peter/Rome.ps.gz

[Par 69] D. Park, *Fixpoint Induction and Proofs of Program Properties*, in: B. Meltzer, D. Michie, eds., Machine Intelligence 5, Elsevier (1969) 59-78

[Pau 94] L.C. Paulson, *Isabelle: A Generic Theorem Prover*, Springer LNCS 828 (1994)

[Pau 96] L.C. Paulson, *ML for the Working Programmer*, 2nd ed., Cambridge University Press 1996

[PS 87] A. Pettorossi, A. Skowron, *Higher-Order Generalization in Program Derivation*, Proc. TAPSOFT '87, Springer LNCS 250 (1987) 182-196

[Plo 70] G.D. Plotkin, *A Note on Inductive Generalization*, in: B. Meltzer, D. Michie, eds., Machine Intelligence 5, Elsevier (1970) 153-163

[Plo 72] G.D. Plotkin, *Building-in Equational Theories*, in: B. Meltzer, D. Michie, eds., Machine Intelligence 7, Elsevier (1972) 73-90

[Plo 83] G.D. Plotkin, *An Operational Semantics for CSP*, in: D. Bjørner, ed., Proc. IFIP TC-2 Working Conf. Formal Description of Programming Concepts II, North-Holland (1983) 199-225

[Red 90] U. Reddy, *Term Rewriting Induction*, Proc. CADE 10, Springer LNCS 449 (1990) 162-177

[Rei 95] H. Reichel, *An Approach to Object Semantics based on Terminal Coalgebras*, Math. Structures in Comp. Sci. 5 (1995) 129-152

[Rut 96] J.J.M.M. Rutten, *Universal Coalgebra: A Theory of Systems*, Report CS-R9652, CWI, SMC Amsterdam 1996

[Sla 74] J.R. Slagle, Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity, J. ACM 21 (1974) 622-642

[Sti 85] M. Stickel, *Automated Deduction by Theory Resolution*, J. Automated Reasoning 1 (1985) 333-356

[Sti 92] C. Stirling, *Modal and Temporal Logics*, in: S. Abramsky et al., eds., Handbook of Logic in Computer Science, Clarendon Press (1992) 477-563

[SMI 95] T. Suzuki, A. Middeldorp, T. Ida, *Level-Confluence of Conditional Rewrite Systems with Extra Variables in Right-Hand Sides*, Proc. RTA '95, Springer LNCS 914 (1995) 179-193

[Wan 79] M. Wand, *Final Algebra Semantics and Data Type Extensions*, J. Computer and System Sciences 19 (1979) 27-44

[Wir 90] M. Wirsing, *Algebraic Specification*, in: J. van Leeuwen, ed., Handbook of Theoretical Computer Science, Elsevier (1990) 675-788