Proof Obligations for Monomorphicity*

Arno Schönegge

Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe
D-76128 Karlsruhe, Germany
email: `schoenegge@ira.uka.de`

Abstract

In certain applications of formal methods to development of correct software one wants the requirement specification to be monomorphic, i.e. that every two term-generated models of it are isomorphic. Consequently, the question arises how to guarantee monomorphicity (which is not decidable in general). In this paper we show that the task of proving monomorphicity of a specification can be reduced to a task of proving certain properties of procedures (with indeterministic constructs). So this task can be directly dealt with in the KIV system [4] which was originally designed for software verification. We prove correctness and completeness of our method.

Contents

*This work was supported under grants no. Me 672/6-2,3 by the Deutsche Forschungsgemeinschaft as part of the focus program "Deduktion".

1 Introduction

Formal software development starts with making up a formal requirement specification that describes the features required of the software system to be developed. Requirement specifications may be(come) an essential part of a contract between a customer, who wants to get bug-free software for his (safety-critical) application, and the software developer: the customer assures to accept the software if (and only if) it meets this specification.

In general, the requirement specification must not be monomorphic, i.e. the specification must not determine one model uniquely (up to isomorphism). Quite the reverse holds: in order to provide the software developer with freedom that may facilitate more efficient implementations, it may be desired to specify only the relevant features. However, on the other hand, non-monomorphicity (i.e. ambiguity) can be very dangerous, especially if one is unaware of the (whole extent of the) gaps in the specification. In [6] we illustrate the risk of such ambiguity on an example and argue that (in certain applications) the customer should insist on a monomorphic requirement specification.

The question arises how to guarantee that a specification is monomorphic. Unfortunately, in general[1] monomorphicity is neither easy to see nor decidable at all. The set of all monomorphic specifications is not even effectively enumerable. However, it is possible to *prove* monomorphicity of a given specification, for example by meta-reasoning [3, 5].

In this paper another approach to this problem is presented. We show that the task of proving monomorphicity of a specification can be reduced to the task of proving certain properties of potential indeterministic procedures. This allows one to directly employ well-established techniques known from software verification. In fact, using our method, the task of proving monomorphicity can be directly dealt with in the KIV system (Karlsruhe Interactive Verifier) [4] which was originally designed for program verification.

We prove correctness and completeness of our method in a great detail, but leave out examples and motivations. Thus, this is a very technical paper, made for internal use rather than for external readers.

To keep the paper self-contained we give all the basic definitions and a few elementary facts in the next section. In section 3 a theorem (3.9) is proved, which states criteria necessary and sufficient for monomorphicity. These criteria are well suited to be formulated as proof obligations in dynamic logic, which is established in theorem 4.7 of section 4. Finally, in the last section we draw conclusions and indicate directions for future work.

[1]If one restricts oneself to freely generated data types enriched by algorithmic specifications, the things get much simpler, since in this case, determinism and totality of these algorithms are sufficient for monomorphicity.

2 Basic Definitions

In this section we provide the basic definitions and a few elementary facts. Some of them are adopted from [2] and [7].

2.1 Sets, Relations, Functions

Definition 2.1 (notions for sets of tuples)
Given a set A, $n \in \mathbb{N}$ the set of n-tuples of A is denoted by A^n. We use $a_1 \cdots a_n$ or $(a_1, \ldots, a_n)$ as notations for tuples; the empty tuple (i.e. $n = 0$) is written λ or $()$. $set(a_1 \cdots a_n)$ denotes the set $\{a_1, \ldots, a_n\}$. The concatenation $a_1 \cdots a_n b_1 \cdots b_m$ of two tuples $\underline{a} = a_1 \cdots a_n$ and $\underline{b} = b_1 \cdots b_m$ is written $\underline{ab}$. Furthermore, we set:

- $A^+ := \bigcup_{n \in \mathbb{N} \setminus \{0\}} A^n$
- $A^* := \bigcup_{n \in \mathbb{N}} A^n$
- $\hat{A}^n := \{\ (a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in A^n,\ a_i \neq a_j \text{ for all } 0 \leq i < j \leq n\ \}$
- $\hat{A}^+ := \bigcup_{n \in \mathbb{N} \setminus \{0\}} \hat{A}^n$
- $\hat{A}^* := \bigcup_{n \in \mathbb{N}} \hat{A}^n$.

For a set of (so-called) indices I and sets A_i, $i \in I$ the family (or system) of the sets A_i is denoted by $(A_i)_{i \in I}$. For $i_1 \cdots i_n \in I^n$, we use the following abbreviations:

- $A_{i_1 \cdots i_n} := A_{i_1} \times \cdots \times A_{i_n}$
- $\hat{A}_{i_1 \cdots i_n} := \left\{ (a_1, \ldots, a_n) \;\middle|\; \begin{array}{l} (a_1, \ldots, a_n) \in A_{i_1 \cdots i_n}, \\ a_i \neq a_j \text{ for all } 0 \leq i < j \leq n \end{array} \right\}$.

Definition 2.2 (relations)
A (binary) **relation** R between two sets A and B is a subset of the cartesian product $A \times B$. The set $R^{-1} := \{(b, a) \mid (a, b) \in R\}$ is the **inverse** of R. R is said to be

- **rightunique**, if for all $a \in A$, $b_1, b_2 \in B$ holds that $(a, b_1), (a, b_2) \in R$ implies $b_1 = b_2$.
- **leftunique**, if for all $a_1, a_2 \in A$, $b \in B$ holds that $(a_1, b), (a_2, b) \in R$ implies $a_1 = a_2$.
- **righttotal**, if for all $b \in B$ there is some $a \in A$ with $(a, b) \in R$.
- **lefttotal**, if for all $a \in A$ there is some $b \in B$ with $(a, b) \in R$.

Definition 2.3 (functions)
A rightunique relation $R \subseteq A \times B$ is also called **(partial) function** from A into B, which is indicated by the notion $R : A \to B$. Furthermore, in this case we use the term **injective** instead of leftunique, **surjective** instead of righttotal, and **total** instead of lefttotal. A total function is said to be **bijective**, if it is injective and surjective.

For a total function $f : A \to B$, and $A' \subseteq A$ we write $f(A')$ to denote the set $\{f(a) \,|\, a \in A'\}$. For $a_1 \cdots a_n \in A^n$ the tuple $f(a_1) \cdots f(a_n)$ is abreviated by $f(a_1 \cdots a_n)$.

Definition 2.4 (families of relations/functions)
For a set of (so-called) indices I and relations (functions) R_i, $i \in I$ we call $R = (R_i)_{i \in I}$ a **family of relations (functions)**. If all R_i are rightunique (leftunique, righttotal, lefttotal) R itself is called **rightunique** (**leftunique**, **righttotal**, **lefttotal**). In the case of functions the corresponding terms **injective**, **surjective**, **total** and **bijective** will be used. Furthermore, the **inverse** of R is defined by $R^{-1} := (R_i^{-1})_{i \in I}$.

For two families of relations $R1 = (R1_i)_{i \in I}$ and $R2 = (R2_i)_{i \in I}$ with the same index set I we write $R1 \subseteq R2$ if $R1_i \subseteq R2_i$ for all $i \in I$.

Fact 2.5 (totality and uniqueness of relations)
For any family R of (binary) relations holds:

(1) $(R^{-1})^{-1} = R$.

(2) *R leftunique iff R^{-1} rightunique.*

(3) *R lefttotal iff R^{-1} righttotal.*

2.2 Signatures, Terms, Algebras

Definition 2.6 (signatures)
A **signature** $SIG = (S, F, P)$ consists of a finite set S of **sorts**, a finite set F of **function symbols**, where F is the disjoint union of sets $F_{\underline{s},s}$ with $\underline{s} \in S^*$, $s \in S$, and a finite set P of **predicate symbols**, where P is the disjoint union of sets $P_{\underline{s}}$ with $\underline{s} \in S^*$. For $\underline{s} = s_1 \cdots s_n$ is $F_{\underline{s},s}$ the set of all n-ary function symbols from sorts $s_1 \cdots s_n$ to sort s, and $P_{\underline{s}}$ the set of all n-ary predicate symbols over sorts $s_1 \cdots s_n$.

For any signature $SIG = (S, F, P)$ we assume a system $X = (X_s)_{s \in S}$ of countable, infinite, and pairwise disjoint sets X_s of **variables** for every sort $s \in S$, and a system $Pid = (Pid_{\underline{s}_1,\underline{s}_2})_{\underline{s}_1,\underline{s}_2 \in S^*}$ of countable, infinite, and pairwise disjoint sets of **procedure identifiers** for every $\underline{s}_1, \underline{s}_2 \in S^*$.

Definition 2.7 (terms)
Given $SIG = (S, F, P)$ and a system X of variables for SIG, the family $T_F(X) = \big(T_{F,s}(X)\big)_{s \in S}$ of **terms** over SIG and X is defined as the least family of sets such that

- $X_s \subseteq T_{F,s}(X)$ for every $s \in S$, and
- $f\underline{t} \in T_{F,s}(X)$ for every $\underline{s} \in S^*$, $s \in S$, $f \in F_{\underline{s},s}$, $\underline{t} \in T_{F,\underline{s}}(X)$.

The family $T_F = \big(T_{F,s}\big)_{s \in S}$ of variable-free terms (so-called **ground terms**) over SIG is defined as the least family of sets such that

- $f\underline{t} \in T_{F,s}$ for every $\underline{s} \in S^*$, $s \in S$, $f \in F_{\underline{s},s}$, $\underline{t} \in T_{F,\underline{s}}$.

Definition 2.8 (sensible signatures)
A signature $SIG = (S, F, P)$ is called **sensible**[2] (cf. [7]) iff there is at least one ground term for any sort, i.e. $T_{F,s} \neq \emptyset$ for all $s \in S$.

Definition 2.9 (algebras, valuations)
For a signature $SIG = (S, F, P)$ a SIG-**algebra** $\mathcal{A}$ is a triple, written $\mathcal{A} = \big((A_s)_{s \in S}, (f_{\mathcal{A}})_{f \in F}, (p_{\mathcal{A}})_{p \in P}\big)$, where $(A_s)_{s \in S}$ is a family of non-empty carrier sets (domain), $(f_{\mathcal{A}})_{f \in F}$ is a family of interpretations for the function symbols in F, and $(p_{\mathcal{A}})_{p \in P}$ is a family of interpretations for the predicate symbols in P. For $f \in F_{\underline{s},s}$, with $\underline{s} \in S^*, s \in S$ is $f_{\mathcal{A}}$ a *total* function from $A_{\underline{s}}$ into A_s. For $p \in P_{\underline{s}}$, with $\underline{s} \in S^*$ is $p_{\mathcal{A}}$ a subset of $A_{\underline{s}}$. The set of all SIG-algebras is denoted by $Alg(SIG)$.

For a system X of variables for SIG and an $\mathcal{A} \in Alg(SIG)$ an $\mathcal{A}$-**valuation** $v = (v_s)_{s \in S}$ is a family of total functions $v_s : X_s \to A_s$. In the setting of dynamic logic, valuations are also called **states**. $Val(X, \mathcal{A})$ is the set of all such $\mathcal{A}$-valuations. For $\underline{s} \in S^*$, $\underline{x} = x_1 \cdots x_n \in \hat{X}_{\underline{s}}$, and $\underline{a} = a_1 \cdots a_n \in A_{\underline{s}}$, we write $v\frac{\underline{a}}{\underline{x}}$ for the modification of v which assigns a_i to x_i and is otherwise the same as v.

Definition 2.10 (disjoint signatures, sum of algebras)
Two signatures $SIG = (S, F, P)$, $SIG' = (S', F', P')$ are said to be **disjoint**, written $SIG \cap SIG' = \emptyset$, if $S \cap S' = \emptyset$, $F \cap F' = \emptyset$, and $P \cap P' = \emptyset$. In this case $SIG \cup SIG' := (S \cup S', F \cup F', P \cup P')$ is again a signature, and from $\mathcal{A} \in Alg(SIG)$, $\mathcal{B} \in Alg(SIG')$ a $(SIG \cup SIG')$-algebra

$$\mathcal{A} + \mathcal{B} \;:=\; \Big(\big((A+B)_s\big)_{s \in S \cup S'}, \big(f_{\mathcal{A}+\mathcal{B}}\big)_{f \in F \cup F'}, \big(p_{\mathcal{A}+\mathcal{B}}\big)_{p \in P \cup P'}\Big)$$

[2]In fact, the restriction to sensible signatures is sensible in practice.

can be constructed by (cf. in [1]: amalgamated sum of algebras)

$$(A+B)_s \;:=\; \begin{cases} A_s & \text{, if } s \in S \\ B_s & \text{, if } s \in S' \end{cases}$$

$$f_{(\mathcal{A}+\mathcal{B})} := \begin{cases} f_{\mathcal{A}} & \text{, if } f \in F \\ f_{\mathcal{B}} & \text{, if } f \in F' \end{cases} \quad \text{and} \quad p_{(\mathcal{A}+\mathcal{B})} := \begin{cases} p_{\mathcal{A}} & \text{, if } p \in P \\ p_{\mathcal{B}} & \text{, if } p \in P'. \end{cases}$$

For two sets of algebras $\mathcal{M} \subseteq Alg(SIG)$ and $\mathcal{M}' \subseteq Alg(SIG')$ we write $\mathcal{M} + \mathcal{M}'$ to denote the set $\{\mathcal{A}+\mathcal{B} \mid \mathcal{A} \in \mathcal{M}, \mathcal{B} \in \mathcal{M}'\}$.

Fact 2.11 (disjoint signatures do not mix in terms)
For disjoint signatures $SIG = (S, F, P)$, $SIG' = (S', F', P')$, and all $s \in S \cup S'$ holds

$$T_{F \cup F', s} \;=\; \begin{cases} T_{F,s} & \text{, if } s \in S \\ T_{F',s} & \text{, if } s \in S'. \end{cases}$$

Proof. Induction on the structure of ground terms. ∎

Definition 2.12 (semantics of terms)
Let $SIG = (S, F, P)$ be a signature with a system X of variables, $\mathcal{A} \in Alg(SIG)$, $v \in Val(X, \mathcal{A})$. The **value** $t_{v,\mathcal{A}}$ of a term $t \in \bigcup_{s \in S} T_{F,s}(X)$ in $\mathcal{A}$ under v is given by:

- $x_{v,\mathcal{A}} := v_s(x)$ for $x \in X_s$, $s \in S$;
- $(f\underline{t})_{v,\mathcal{A}} := f_{\mathcal{A}}(\underline{t}_{v,\mathcal{A}})$ for $\underline{s} \in S^*$, $s \in S$, $f \in F_{\underline{s},s}$, $\underline{t} = t_1 \cdots t_n \in T_{F,\underline{s}}(X)$,

where $(t_1 \cdots t_n)_{v,\mathcal{A}} := t_{1v,\mathcal{A}} \cdots t_{nv,\mathcal{A}}$. If t is a ground term, its value does not depend on v; therefore, we sometimes write $t_{\mathcal{A}}$ instead of $t_{v,\mathcal{A}}$ in this case.

Definition 2.13 (generatedness)
An algebra $\mathcal{A} \in Alg(SIG)$, $SIG = (S, F, P)$ is called **term-generated** or **generated**, if for any $s \in S$ and any $a \in A_s$ exists a ground term $t \in T_{F,s}$ with $t_{\mathcal{A}} = a$. The set of all generated SIG-algebras (which are also called SIG-**computation structures** [7]) is denoted by $Gen(SIG)$.[3]

Fact 2.14 (disjoint sum of generated algebras)
For disjoint signatures SIG, SIG' is

$$Gen(SIG \cup SIG') \;=\; Gen(SIG) + Gen(SIG').$$

Proof. Let $SIG = (S, F, P)$, $SIG' = (S', F', P')$ be disjoint signatures. It is quite obvious that the sum $\mathcal{A} + \mathcal{A}'$ of two algebras $\mathcal{A} \in Gen(SIG)$, $\mathcal{A}' \in Gen(SIG')$ is in $Gen(SIG \cup SIG')$. For the reverse inclusion of the above set equation, let $\mathcal{B} \in Gen(SIG \cup SIG')$. Define $\mathcal{A}$ to be the so-called SIG-reduct of $\mathcal{B}$, i.e. $\mathcal{A} := \big((B_s)_{s \in S}, (f_{\mathcal{B}})_{f \in F}, (p_{\mathcal{B}})_{p \in P}\big)$. Accordingly, define $\mathcal{A}'$ to be the SIG'-reduct of $\mathcal{B}$. Then holds $\mathcal{A} \in Alg(SIG)$ and $\mathcal{A}' \in Alg(SIG')$ with $\mathcal{A} + \mathcal{A}' = \mathcal{B}$. To see, that $\mathcal{A}$ and $\mathcal{A}'$ are even generated use fact 2.11. ∎

[3]Notice that $Gen(SIG)$ is empty if SIG is not sensible.

2.3 Homomorphisms, Isomorphisms

Definition 2.15 (homomorphisms, isomorphisms)
Let $SIG = (S, F, P)$, $\mathcal{A}, \mathcal{B} \in Alg(SIG)$, and $R = (R_s)_{s \in S}$ a family of relations $R_s \subseteq A_s \times B_s$. R is said to be

- **closed against** F **wrt**[4] $\mathcal{A}$ **and** $\mathcal{B}$, if for all $\underline{s} = s_1 \cdots s_n \in S^*$, $s \in S$, $f \in F_{\underline{s},s}$ holds that $(a_1, b_1) \in R_{s_1}, \ldots, (a_n, b_n) \in R_{s_n}$ implies $\big(f_{\mathcal{A}}(a_1, \ldots, a_n), f_{\mathcal{B}}(b_1, \ldots, b_n)\big) \in R_s$.
- **monotonic in** P **wrt** $\mathcal{A}$ **and** $\mathcal{B}$, if for all $\underline{s} = s_1 \cdots s_n \in S^*$, $p \in P_{\underline{s}}$, $(a_1, b_1) \in R_{s_1}, \ldots, (a_n, b_n) \in R_{s_n}$ holds that $(a_1, \ldots, a_n) \in p_{\mathcal{A}}$ implies $(b_1, \ldots, b_n) \in p_{\mathcal{B}}$.

A **weak** (SIG-)**homomorphism** h from $\mathcal{A}$ to $\mathcal{B}$ is a family $h = (h_s)_{s \in S}$ of total functions $h_s : A_s \to B_s$ which is closed against F wrt $\mathcal{A}$ and $\mathcal{B}$ and monotonic in P wrt $\mathcal{A}$ and $\mathcal{B}$. If even h^{-1} is monotonic in P wrt $\mathcal{B}$ and $\mathcal{A}$, then h is called a (SIG-)**homomorphism**. An **isomorphism** is a bijective homomorphism. Two algebras $\mathcal{A}, \mathcal{B} \in Gen(SIG)$ are called **isomorphic** if there exists an isomorphism $h : \mathcal{A} \to \mathcal{B}$ from $\mathcal{A}$ to $\mathcal{B}$.

The following fact says that these definitions of weak homomorphisms, homomorphisms and isomorphicity are equivalent to the usual ones, e.g. given in [2].

Fact 2.16 (homomorphisms are defined as usual)
Given $SIG = (S, F, P)$, $\mathcal{A}, \mathcal{B} \in Alg(SIG)$ and a family $h = (h_s)_{s \in S}$ of total functions $h_s : A_s \to B_s$.

(1) *h is closed against F wrt $\mathcal{A}$ and $\mathcal{B}$ iff for all $\underline{s} = s_1 \cdots s_n \in S^*$, $s \in S$, $f \in F_{\underline{s},s}$, $a_1 \cdots a_n \in A_{\underline{s}}$ holds that*

$$h_s(f_{\mathcal{A}}(a_1, \ldots, a_n)) \quad = \quad f_{\mathcal{B}}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)).$$

(2) *h is monotonic in P wrt $\mathcal{A}$ and $\mathcal{B}$ iff for all $\underline{s} = s_1 \cdots s_n \in S^*$, $p \in P_{\underline{s}}$, $a_1 \cdots a_n \in A_{\underline{s}}$ holds that*

$$(a_1, \ldots, a_n) \in p_{\mathcal{A}} \quad \textit{implies} \quad (h_{s_1}(a_1), \ldots, h_{s_n}(a_n)) \in p_{\mathcal{B}}.$$

Proof. The fact is quite obvious. To see this, remember that in the case that h is a total function the notion $(a, b) \in h$ can be replaced by the (equivalent) notion $h(a) = b$. ■

[4]wrt = with respect to

Fact 2.17 (inverse of isomorphisms)
Let $SIG = (S, F, P)$ and $\mathcal{A}, \mathcal{B} \in Alg(SIG)$.

(1) *A family $R = (R_s)_{s \in S}$ of relations $R_s \subseteq A_s \times B_s$ is closed against F wrt $\mathcal{A}$ and $\mathcal{B}$ iff R^{-1} is closed against F wrt $\mathcal{B}$ and $\mathcal{A}$.*

(2) *The inverse h^{-1} of an isomorphism h from $\mathcal{A}$ to $\mathcal{B}$ is an isomorphism from $\mathcal{B}$ to $\mathcal{A}$.*

Proof. To prove (1) assume R to be closed against F wrt $\mathcal{A}$ and $\mathcal{B}$, and $f \in F_{\underline{s},s}$ for some $\underline{s} = s_1 \cdots s_n \in S^*$, $s \in S$. Then for any $(b_1, a_1) \in R^{-1}_{s_1}, \ldots, (b_n, a_n) \in R^{-1}_{s_n}$ holds that $(a_1, b_1) \in R_{s_1}, \ldots, (a_n, b_n) \in R_{s_n}$, and therefore, by the assumption that $\big(f_{\mathcal{A}}(a_1, \ldots, a_n), f_{\mathcal{B}}(b_1, \ldots, b_n)\big) \in R_s$, i.e. $\big(f_{\mathcal{B}}(b_1, \ldots, b_n), f_{\mathcal{A}}(a_1, \ldots, a_n)\big) \in R^{-1}_s$. Thus R^{-1} is closed against F wrt $\mathcal{B}$ and $\mathcal{A}$. The other direction follows from this one together with fact 2.5(1).

Now we turn to the proof of (2). Due to the bijectivity of h is h^{-1} a family of total, bijective functions, and from (1) follows that h^{-1} is closed against F wrt $\mathcal{B}$ and $\mathcal{A}$. So, it remains to show that h^{-1} is monotonic in P wrt $\mathcal{B}$ and $\mathcal{A}$ and $(h^{-1})^{-1} = h$ is monotonic in P wrt $\mathcal{A}$ and $\mathcal{B}$, which is guaranteed by assumption. ■

2.4 First-order Logic, Dynamic Logic

Definition 2.18 (atomic formulas)
Let $SIG = (S, F, P)$ and X be a system of variables for SIG. The set $AT(SIG, X)$ of **atomic formulas** over SIG and X is the least set satisfying:

- $\mathbf{true}, \mathbf{false} \in AT(SIG, X)$,
- for $s \in S$, $t_1, t_2 \in T_{F,s}(X)$ is $(t_1 = t_2) \in AT(SIG, X)$,
- for $\underline{s} \in S^*$, $p \in P_{\underline{s}}$, $\underline{t} \in T_{F,\underline{s}}(X)$ is $p\underline{t} \in AT(SIG, X)$.

Definition 2.19 (boolean expressions)
Let $SIG = (S, F, P)$ and X be a system of variables for SIG. The set $BXP(SIG, X)$ of **boolean expressions** over SIG and X is the least set satisfying:

- $AT(SIG, X) \subseteq BXP(SIG, X)$,
- for $\varphi, \psi \in BXP(SIG, X)$ is
  $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$, $(\varphi \leftrightarrow \psi) \in BXP(SIG, X)$.

Definition 2.20 (first-order formulas)
Let $SIG = (S, F, P)$ and X be a system of variables for SIG. The set $FO(SIG, X)$ of **first-order formulas** over SIG and X is the least set satisfying:

- $AT(SIG, X) \subseteq FO(SIG, X)$,
- for $\varphi, \psi \in FO(SIG, X)$ is
  $\neg\varphi\,,\ (\varphi \wedge \psi)\,,\ (\varphi \vee \psi)\,,\ (\varphi \rightarrow \psi)\,,\ (\varphi \leftrightarrow \psi) \in FO(SIG, X)$,
- for $\varphi \in FO(SIG, X)$, $\underline{s} \in S^+$ and $\underline{x} \in \hat{X}_{\underline{s}}$ is
  $\forall \underline{x}.\varphi\,,\ \exists \underline{x}.\varphi \in FO(SIG, X)$.

Definition 2.21 (extension by counters)
For the **counter signature** $CSIG = (\{ctr\}, F_{(),ctr} \cup F_{ctr,ctr}, P_{ctr,ctr})$ with $F_{(),ctr} := \{czero\}$, $F_{ctr,ctr} := \{csucc\}$, and $P_{ctr,ctr} := \{<_{ctr}\}$ we fix a standard algebra $\mathcal{A}_{ctr}$ with the carrier $A_{ctr} = \mathbb{N}$, and which gives $czero$, $csucc$ and $<_{ctr}$ their usual meanings, i.e. zero, successor-function and less-predicate on natural numbers.

We assume all signatures SIG considered in the following to be disjoint from $CSIG$. So the **standard extension** $SIG_+ := SIG \cup CSIG$ of SIG is well-defined. Correspondingly, we fix a countable, infinite set X_{ctr} of variables for sort ctr, and assume all other sets of variables considered in the following, to be disjoint from X_{ctr}. So we can define the standard extension $X_+ := (X_s)_{s \in S \cup \{ctr\}}$ of a system $X = (X_s)_{s \in S}$ of variables for SIG. The standard extension of an $\mathcal{A} \in Alg(SIG)$ is the SIG_+-algebra $\mathcal{A}_+ := \mathcal{A} + \mathcal{A}_{ctr}$.

Definition 2.22 (commands)
Given $SIG = (S, F, P)$, X a system of variables for SIG, and Pid a system of procedure identifiers for SIG. The set $CMD(SIG, X, Pid)$ of **commands** over SIG, X and Pid is the least set satisfying:

- (skip, abort)
  skip, **abort** $\in CMD(SIG, X, Pid)$,
- (assignment, random assignment)
  for $s \in S$, $x \in X_s$, and $t \in T_{F,s}(X)$ is
  $(x := t)\,,\ (x :=?) \in CMD(SIG, X, Pid)$,
- (nondeterministic choice)
  for $\underline{\alpha} \in CMD(SIG, X, Pid)^*$ is
  $\bigcup \underline{\alpha} \in CMD(SIG, X, Pid)$,
- (composition, conditional)
  for $\alpha, \beta \in CMD(SIG, X, Pid)$ and $\epsilon \in BXP(SIG, X)$ is
  $(\alpha; \beta)$, **if** ϵ **then** α **else** β $\in CMD(SIG, X, Pid)$,
- (local (random) variable declaration)
  for $\alpha \in CMD(SIG, X, Pid)$, $\underline{s} \in S^+$, $\underline{x} \in \hat{X}_{\underline{s}}$ and $\underline{t} \in T_{F,\underline{s}}(X)$ is
  var $\underline{x} = \underline{t}$ **in** α , **var** $\underline{x} =?$ **in** α $\in CMD(SIG, X, Pid)$,

- (procedure call)
  for $\underline{s}_1, \underline{s}_2 \in S^*$, $q \in Pid_{\underline{s}_1,\underline{s}_2}$, $\underline{t} \in T_{F,\underline{s}_1}(X)$, and $\underline{z} \in \hat{X}_{\underline{s}_2}$ is
  $q(\underline{t};\underline{z}) \in CMD(SIG, X, Pid)$.

Definition 2.23 (free variables)
Let $SIG = (S, F, P)$, X a system of variables for SIG, and Pid a system of procedure identifiers for SIG. The function

$$freevars : CMD(SIG, X, Pid) \to \bigcup_{s \in S} X_s$$

is defined by:

- $freevars(\mathbf{skip}) := \emptyset$,
  $freevars(\mathbf{abort}) := \emptyset$,
- $freevars(x := t) := \{x\} \cup vars(t)$,
  $freevars(x :=?) := \{x\}$,
- $freevars(\bigcup \underline{\alpha}) := \bigcup_{\alpha \in \underline{\alpha}} freevars(\alpha)$,
- $freevars(\alpha;\beta) := freevars(\alpha) \cup freevars(\beta)$,
  $freevars(\mathbf{if}\ \epsilon\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta) := vars(\epsilon) \cup freevars(\alpha) \cup freevars(\beta)$,
- $freevars(\mathbf{var}\ \underline{x} = \underline{t}\ \mathbf{in}\ \alpha) := \big(freevars(\alpha) \setminus set(\underline{x})\big) \cup \bigcup_{t \in set(\underline{t})} vars(t)$,
  $freevars(\mathbf{var}\ \underline{x} =?\ \mathbf{in}\ \alpha) := freevars(\alpha) \setminus set(\underline{x})$,
- $freevars(q(\underline{t};\underline{z})) := \bigcup_{t \in set(\underline{t})} vars(t) \cup set(\underline{z})$.

Here, $vars$ yields the set of variables occurring in a term or in a boolean expression.

Definition 2.24 (procedure declarations)
Given $SIG = (S, F, P)$, X a system of variables for SIG and Pid a system of procedure identifiers for SIG. The set $PD(SIG, X, Pid)$ of **procedure declarations** over SIG, X and Pid is defined by:

$PD(SIG, X, Pid) :=$

$$\left\{ q(\underline{x};\mathbf{var}\ \underline{y}).\alpha \;\middle|\; \begin{array}{l} \underline{s}_1, \underline{s}_2 \in S^*, q \in Pid_{\underline{s}_1,\underline{s}_2}, \underline{x} \in X_{\underline{s}_1}, \underline{y} \in X_{\underline{s}_2}, \underline{xy} \in \hat{X}_{\underline{s}_1\underline{s}_2} \\ \alpha \in CMD(SIG, X, Pid), freevars(\alpha) \subseteq set(\underline{xy}) \end{array} \right\}.$$

Definition 2.25 (look-up of declarations)
Given SIG a signature, X a system of variables for SIG, and Pid a system of procedure identifiers for SIG. For

$$\delta \;=\; \big(q_1(\underline{x}_1;\mathbf{var}\ \underline{y}_1).\alpha_1, \ldots, q_n(\underline{x}_n;\mathbf{var}\ \underline{y}_n).\alpha_n\big) \;\in\; PD(SIG, X, Pid)^*$$

we set $dom(\delta) := \{q_1, \ldots, q_n\}$ and for a $q \in dom(\delta)$ we declare the **declaration of q in δ** by $\delta_q := q_i(\underline{x}_i;\mathbf{var}\ \underline{y}_i).\alpha_i$, where i is the least number with $q_i = q$.

Definition 2.26 (programs)
Let $SIG = (S, F, P)$, X a system of variables for SIG, and Pid a system of procedure identifiers for SIG. Let further $SIG_+ = (S_+, F_+, P_+)$ the standard extension of SIG by counters. The set $PROG(SIG, X, Pid)$ of **programs** over SIG, X and Pid is the least set satisfying:

- $CMD(SIG, X, Pid) \subseteq PROG(SIG, X, Pid)$,
- for $\delta \in PD(SIG, X, Pid)^*$ and $\alpha \in CMD(SIG, X, Pid)$ is
  proc δ **in** $\alpha \ \in PROG(SIG, X, Pid)$,
- for $\delta \in PD(SIG, X, Pid)^*$, $\kappa \in T_{F_+,ctr}(X_{ctr})$,
  and $\alpha \in CMD(SIG, X, Pid)$ is
  proc δ **times** κ **in** $\alpha \ \in PROG(SIG, X, Pid)$.

Definition 2.27 (dynamic logic formulas)
Let $SIG = (S, F, P)$, X a system of variables for SIG, Pid a system of procedure identifiers for SIG, $SIG_+ = (S_+, F_+, P_+)$ the standard extension of SIG, and X_+ the standard extension of X. The set $DL(SIG, X, Pid)$ of **dynamic logic formulas** over SIG, X and Pid is the least set satisfying:

- $AT(SIG_+, X_+) \subseteq DL(SIG, X, Pid)$,
- for $\varphi, \psi \in DL(SIG, X, Pid)$ is
  $\neg\varphi\, , (\varphi \wedge \psi)\, , (\varphi \vee \psi)\, , (\varphi \rightarrow \psi)\, , (\varphi \leftrightarrow \psi) \in DL(SIG, X, Pid)$,
- for $\varphi \in DL(SIG, X, Pid)$, $\underline{s} \in S_+^+$, and $\underline{x} \in \hat{X}_{\underline{s}}$ is
  $\forall \underline{x}.\varphi\, , \exists \underline{x}.\varphi \in DL(SIG, X, Pid)$,
- for $\alpha \in PROG(SIG, X, Pid)$ and $\varphi \in DL(SIG, X, Pid)$ is
  $\langle\alpha\rangle\varphi\, , [\alpha]\varphi \ \in DL(SIG, X, Pid)$.

Remark. Notice that:

- $BXP(SIG, X) \subset FO(SIG, X) \subset DL(SIG, X, Pid)$.
- There are no counters involved in commands.
- We have restricted ourselves to programs without **while**-loops, local procedure declarations, and global variables. This allows to simplify the definition of semantics (in comparison with the one given in [2]). Especially, we get by without replacement of variables in programs and without the (so-called) environment construct.

Definition 2.28 (semantics of programs and formulas)
Let $SIG = (S, F, P)$ a signature with a system X of variables and a system Pid of procedure identifiers, and $\mathcal{A} \in Alg(SIG)$. Let further $SIG_+ = (S_+, F_+, P_+)$, X_+ and $\mathcal{A}_+$ the standard extensions of SIG, X and $\mathcal{A}$ and $v, v' \in Val(X_+, \mathcal{A}_+)$. For $\varphi \in DL(SIG, X, Pid)$ we write $\mathcal{A}, v \models \varphi$ if φ **is true in $\mathcal{A}$ under** v, and $\mathcal{A}, v \not\models \varphi$ otherwise. For $\alpha \in PROG(SIG, X, Pid)$ we write $v \, [\![\alpha]\!]_{\mathcal{A}} \, v'$ if v' is a state that can be reached from state v by executing α interpreted under $\mathcal{A}$. The relation $[\![\alpha]\!]_{\mathcal{A}} \subseteq Val(X_+, \mathcal{A}_+) \times Val(X_+, \mathcal{A}_+)$ describes the input-output behavior of α under $\mathcal{A}$. These notions are defined simultaneously[5] as follows:

- $\mathcal{A}, v \models \textbf{true}$, $\mathcal{A}, v \not\models \textbf{false}$
- $\mathcal{A}, v \models t_1 = t_2$ iff $t_{1v,\mathcal{A}_+} = t_{2v,\mathcal{A}_+}$
- $\mathcal{A}, v \models p\underline{t}$ iff $\underline{t}_{v,\mathcal{A}_+} \in p_{\mathcal{A}_+}$
- $\mathcal{A}, v \models \neg\,\varphi$ iff $\mathcal{A}, v \not\models \varphi$
  $\mathcal{A}, v \models \varphi \wedge \psi$ iff ($\mathcal{A}, v \models \varphi$ and $\mathcal{A}, v \models \psi$)
  $\mathcal{A}, v \models \varphi \vee \psi$ iff ($\mathcal{A}, v \models \varphi$ or $\mathcal{A}, v \models \psi$)
  $\mathcal{A}, v \models \varphi \rightarrow \psi$ iff ($\mathcal{A}, v \models \varphi$ implies $\mathcal{A}, v \models \psi$)
  $\mathcal{A}, v \models \varphi \leftrightarrow \psi$ iff ($\mathcal{A}, v \models \varphi$ iff $\mathcal{A}, v \models \psi$)
- $\mathcal{A}, v \models \forall \underline{x}.\varphi$ (where $\underline{x} \in \hat{X}_{\underline{s}}$ for some $\underline{s} \in S^*$) iff for all $\underline{a} \in A_{\underline{s}}$ holds $\mathcal{A}, v^{\underline{a}}_{\underline{x}} \models \varphi$
  $\mathcal{A}, v \models \exists \underline{x}.\varphi$ (where $\underline{x} \in \hat{X}_{\underline{s}}$ for some $\underline{s} \in S^*$) iff there is an $\underline{a} \in A_{\underline{s}}$ such that $\mathcal{A}, v^{\underline{a}}_{\underline{x}} \models \varphi$
- $\mathcal{A}, v \models \langle\alpha\rangle\varphi$ iff there is a $v' \in Val(X_+, \mathcal{A}_+)$ with $v[\![\alpha]\!]_{\mathcal{A}}v'$ and $\mathcal{A}, v' \models \varphi$
  $\mathcal{A}, v \models [\alpha]\varphi$ iff for all $v' \in Val(X_+, \mathcal{A}_+)$ with $v \, [\![\alpha]\!]_{\mathcal{A}} \, v'$ holds $\mathcal{A}, v' \models \varphi$
- $v \, [\![\alpha]\!]_{\mathcal{A}} \, v'$ iff $v \, [\![\textbf{proc}\ ()^{6}\ \textbf{in}\ \alpha]\!]_{\mathcal{A}} \, v'$ for $\alpha \in CMD(SIG, X, Pid)$
  $v \, [\![\textbf{proc}\ \delta\ \textbf{in}\ \alpha]\!]_{\mathcal{A}} \, v'$ iff there is some ground term $\kappa \in T_{F_+,ctr}$ with $v \, [\![\textbf{proc}\ \delta\ \textbf{times}\ \kappa\ \textbf{in}\ \alpha]\!]_{\mathcal{A}} \, v'$
- $v \, [\![\textbf{proc}\ \delta\ \textbf{times}\ \kappa\ \textbf{in}\ \textbf{skip}]\!]_{\mathcal{A}} \, v'$ iff $v = v'$
  $v \, [\![\textbf{proc}\ \delta\ \textbf{times}\ \kappa\ \textbf{in}\ \textbf{abort}]\!]_{\mathcal{A}} \, v'$ for no $v, v' \in Val(X_+, \mathcal{A}_+)$
- $v \, [\![\textbf{proc}\ \delta\ \textbf{times}\ \kappa\ \textbf{in}\ x := t]\!]_{\mathcal{A}} \, v'$ iff $v' = v_x^{t_{v,\mathcal{A}_+}}$
  $v \, [\![\textbf{proc}\ \delta\ \textbf{times}\ \kappa\ \textbf{in}\ x :=?]\!]_{\mathcal{A}} \, v'$ (where $x \in X_s$ for some $s \in S$) iff there is some $a \in A_s$ such that $v' = v_x^a$

[5]Since the semantics of programs and formulas depend on each other, we have to define it simultaneously.

[6]Here () is the empty procedure declaration.

- $v\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ \bigcup \underline{\alpha}]\!]_{\mathcal{A}}\, v'$ iff
  $v\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ \alpha]\!]_{\mathcal{A}}\, v'$ for some $\alpha \in set(\underline{\alpha})$

- $v\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ (\alpha;\beta)]\!]_{\mathcal{A}}\, v'$ iff
  there is a $v'' \in Val(X_+, \mathcal{A}_+)$ such that $v\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ \alpha]\!]_{\mathcal{A}}\, v''$ and $v''\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ \beta]\!]_{\mathcal{A}}\, v'$

  $v\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ \mathbf{if}\ \epsilon\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta]\!]_{\mathcal{A}}\, v'$ iff
  either $\mathcal{A}, v \models \epsilon$ and $v\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ \alpha]\!]_{\mathcal{A}}\, v'$
  or else $\mathcal{A}, v \not\models \epsilon$ and $v\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ \beta]\!]_{\mathcal{A}}\, v'$

- $v\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ \mathbf{var}\ \underline{x} = \underline{t}\ \mathbf{in}\ \alpha]\!]_{\mathcal{A}}\, v'$ iff
  there is a $v'' \in Val(X_+, \mathcal{A}_+)$ such that $v_{\underline{x}}^{\underline{t}_{v,\mathcal{A}_+}}\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ \alpha]\!]_{\mathcal{A}}\, v''$ and[7] $v' = v''^{v(\underline{x})}_{\underline{x}}$

  $v\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ \mathbf{var}\ \underline{x}\ =?\ \mathbf{in}\ \alpha]\!]_{\mathcal{A}}\, v'$ (where $\underline{x} \in X_{\underline{s}}$ for some $\underline{s} \in S^*$) iff there is an $\underline{a} \in A_{\underline{s}}$ and a $v'' \in Val(X_+, \mathcal{A}_+)$ such that $v_{\underline{x}}^{\underline{a}}\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ \alpha]\!]_{\mathcal{A}}\, v''$ and $v' = v''^{v(\underline{x})}_{\underline{x}}$

- $v\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa\ \mathbf{in}\ q(\underline{t};\underline{z})]\!]_{\mathcal{A}}\, v'$ iff[8] $q \in dom(\delta)$, $\delta_q = q(\underline{x}; \mathbf{var}\ \underline{y}).\alpha$, $\kappa_{v,\mathcal{A}_+} \neq 0$, and there is a $v'' \in Val(X_+, \mathcal{A}_+)$ such that $v' = v_{\underline{z}}^{v''(\underline{y})}$ and $\left(v_{\underline{x}}^{\underline{t}_{v,\mathcal{A}_+}}\right)_{\underline{y}}^{v(\underline{z})}\,[\![\mathbf{proc}\ \delta\ \mathbf{times}\ \kappa'\ \mathbf{in}\ \alpha]\!]_{\mathcal{A}}\, v''$ where $\kappa' \in T_{F_+,ctr}$ is the ground term with $\kappa'_{\mathcal{A}_+} = \kappa_{v,\mathcal{A}_+} - 1$.

A formula φ is said to be **true in** $\mathcal{A}$, written $\mathcal{A} \models \varphi$, if for all $v \in Val(X_+, \mathcal{A}_+)$ holds $\mathcal{A}, v \models \varphi$. Furthermore, for a set of formulas $\Phi \subseteq DL(SIG, X, Pid)$ an algebra $\mathcal{A} \in Alg(SIG)$ is called a **model** of Φ, written $\mathcal{A} \models \Phi$, if $\mathcal{A} \models \varphi$ for all $\varphi \in \Phi$.

Fact 2.29 (formulas of disjoint signatures)
Let SIG, SIG' be disjoint signatures (with $X \cap X' = Pid \cap Pid' = \emptyset$), $\mathcal{A} \in Alg(SIG)$ and $\mathcal{B} \in Alg(SIG')$. Then holds:

$$\mathcal{A} + \mathcal{B} \models \varphi \;\Leftrightarrow\; \mathcal{A} \models \varphi \text{ , for all } \varphi \in DL(SIG, X, Pid)$$
$$\mathcal{A} + \mathcal{B} \models \psi \;\Leftrightarrow\; \mathcal{B} \models \psi \text{ , for all } \psi \in DL(SIG', X', Pid').$$

Proof. The proof works similar to the one of fact 2.33, i.e. by generalization (which introduces valuations) and induction on the syntactical structure of φ (and ψ, respectively). ∎

[7]Notice that $\underline{x}$ is bound to the value it had before execution of the **var** construct.

[8]Notice, that there may be calls in a program that are not declared. Execution of such a call is defined to behave just as non-termination.

2.5 Signature Morphisms

Definition 2.30 (signature morphisms, σ-reducts)
For two signatures $SIG = (S, F, P)$ and $SIG' = (S', F', P')$ a **signature morphism** $\sigma : SIG \to SIG'$ from SIG into SIG' is a triple of total functions $\sigma = (\sigma_S : S \to S', \sigma_F : F \to F', \sigma_P : P \to P')$, such that for any $\underline{s} \in S^*$, $s \in S$, $f \in F_{\underline{s},s}$, $p \in P_{\underline{s}}$ holds $\sigma_F(f) \in F_{\sigma_S(\underline{s}),\sigma_S(s)}$ and $\sigma_P(p) \in P_{\sigma_S(\underline{s})}$, i.e. the types are preserved. We write $\sigma(s)$ for $\sigma_S(s)$, $\sigma(f)$ for $\sigma_F(f)$, and $\sigma(p)$ for $\sigma_P(p)$. A signature morphism σ is said to be **injective** (**surjective**, **bijective**) if σ_S, σ_F and σ_P are injective (surjective, bijective). In the case that σ is bijective, $\sigma^{-1} := (\sigma_S^{-1} : S' \to S, \sigma_F^{-1} : F' \to F, \sigma_P^{-1} : P' \to P)$ is a signature morphism from SIG' into SIG.

The **σ-reduct** $\mathcal{A}'|_\sigma$ of a given SIG'-algebra $\mathcal{A}' \in Alg(SIG')$ is the SIG-algebra

$$\mathcal{A}'|_\sigma \quad := \quad \Big((A'_{\sigma(s)})_{s\in S}, (\sigma(f)_{\mathcal{A}'})_{f\in F}, (\sigma(p)_{\mathcal{A}'})_{p\in P}\Big).$$

The function $\cdot|_\sigma : Alg(SIG') \to Alg(SIG)$ is called the **forgetful operation** defined by σ (cf. [2]). The obvious extension of σ on terms and first-order formulas will also be denoted by σ. Especially, σ maps variables in an injective manner. For an $\mathcal{A}'$-valuation v' we write $v'|_\sigma$ for the $\mathcal{A}'|_\sigma$-valuation defined by[9] $v'|_\sigma := v' \circ \sigma$.

Fact 2.31 (reduct preserves valuation of terms)
Let $SIG = (S, F, P)$ a signature, X a system of variables for SIG, $\sigma : SIG \to SIG'$ a signature morphism, $\mathcal{A}' \in Alg(SIG')$ a SIG'-algebra, v' an $\mathcal{A}'$-valuation, and t a term in $T_F(X)$. Then holds:

$$\sigma(t)_{v',\mathcal{A}'} \quad = \quad t_{v'|_\sigma,\mathcal{A}'|_\sigma}.$$

Proof. Simple induction on the structure of the term t. ■

Fact 2.32 (surjective reduct preserves generatedness)
For a surjective signature morphism $\sigma : SIG \to SIG'$ is the reduct $\mathcal{A}'|_\sigma$ of a generated SIG'-algebra $\mathcal{A}' \in Gen(SIG')$ again generated, i.e. $\mathcal{A}'|_\sigma \in Gen(SIG)$. In short:

$$Gen(\sigma(SIG))|_\sigma \quad \subseteq \quad Gen(SIG).$$

Proof. Let $SIG = (S, F, P)$, $\mathcal{A}' = \Big((A'_{s'})_{s'\in\sigma(S)}, (f'_{\mathcal{A}'})_{f'\in\sigma(F)}, (p'_{\mathcal{A}'})_{p'\in\sigma(P)}\Big)$ a generated $\sigma(SIG)$-algebra, $s \in S$, and $a \in A_s$ where A_s denotes the domain of $\mathcal{A}'|_\sigma$ for s. Due to definition 2.30 holds $A_s = A'_{\sigma(s)}$, i.e. $a \in A'_{\sigma(s)}$. From

[9]For total functions $f : A \to B$, $g : C \to D$ with $f(A) \subseteq C$ is $g \circ f : A \to D$ the total function defined by $(g \circ f)(x) := g(f(x))$.

generatedness of $\mathcal{A}'$ follows that there is a ground term $t' \in T_{\sigma(F),\sigma(s)}$ such that $t'_{\mathcal{A}'} = a$. Because σ is surjective there is a ground term $t \in T_{F,s}$ with $\sigma(t) = t'$. Using fact 2.31 we get $a = t'_{\mathcal{A}'} = \sigma(t)_{\mathcal{A}'} = t_{\mathcal{A}'|_\sigma}$. Thus, for every $s \in S$ and $a \in A_s$ exists a ground term $t \in T_{F,s}$ with $t_{\mathcal{A}'|_\sigma} = a$, i.e. $\mathcal{A}'|_\sigma$ is generated. ■

Fact 2.33 (reduct preserves validity)
For a signature morphism $\sigma : SIG \to SIG'$, a system of variables X for SIG, a first-order formula $\varphi \in FO(SIG, X)$, and a SIG'-algebra $\mathcal{A}' \in Alg(SIG')$ holds:

$$\mathcal{A}' \models \sigma(\varphi) \quad \textit{iff} \quad \mathcal{A}'|_\sigma \models \varphi.$$

Proof. We prove the following slightly generalized lemma: For all SIG'-algebras $\mathcal{A}' \in Alg(SIG')$ and all $\mathcal{A}'$-valuations v' holds

$$\mathcal{A}', v' \models \sigma(\varphi) \quad \text{iff} \quad \mathcal{A}'|_\sigma, v'|_\sigma \models \varphi.$$

This lemma is sufficient to prove the above fact, since for any $\mathcal{A}'|_\sigma$-valuation v exists an $\mathcal{A}'$-valuations v' with $v'|_\sigma = v$, which follows from the assumption that σ maps variables in an injective manner.

The proof of the lemma works by structural induction on the formula φ (let $SIG = (S, F, P)$):

- $\varphi \in \{\mathbf{true}, \mathbf{false}\}$:
  obvious.

- $\varphi \equiv t_1 = t_2$ (where $t_1, t_2 \in T_F(X)$):
  follows immediately from fact 2.31.

- $\varphi \equiv p\underline{t}$ (where $p \in P_{\underline{s},s}$, $\underline{t} \in T_{F,\underline{s}}(X)$):
  from fact 2.31 and $\sigma(p)_{\mathcal{A}'} = p_{\mathcal{A}'|_\sigma}$ follows that:

$$\sigma(\underline{t})_{v',\mathcal{A}'} \in \sigma(p)_{\mathcal{A}'} \quad \text{iff} \quad \underline{t}_{v'|_\sigma,\mathcal{A}'|_\sigma} \in p_{\mathcal{A}'|_\sigma}.$$

- $\varphi \equiv \neg\varphi_1$, $\varphi \equiv \varphi_1 \wedge \varphi_2$, $\varphi \equiv \varphi_1 \vee \varphi_2$, $\varphi \equiv \varphi_1 \to \varphi_2$, $\varphi \equiv \varphi_1 \leftrightarrow \varphi_2$:
  simple application of the induction hypothesis.

- $\varphi \equiv \forall \underline{x}.\varphi_1$:
  Let $\underline{x} \in X_{\underline{s}}$, A' denote the domain of $\mathcal{A}'$, and A denote the domain of $\mathcal{A}'|_\sigma$. By unfolding the semantic definition, applying the induction hypothesis, and using that for any $\underline{a} \in A_{\underline{s}}$ holds $\left(v'\frac{\underline{a}}{\sigma(\underline{x})}\right)\Big|_\sigma = \left(v'|_\sigma\right)\frac{\underline{a}}{\underline{x}}$,

we get:

$$\begin{aligned}
&\mathcal{A}', v' \models \sigma(\forall \underline{x}.\varphi_1)\\
&\quad\Leftrightarrow\quad \mathcal{A}', v' \models \forall \sigma(\underline{x}).\sigma(\varphi_1)\\
&\quad\Leftrightarrow\quad \text{for all } \underline{a} \in A'_{\sigma(\underline{s})} \text{ holds } \mathcal{A}', v'^{\underline{a}}_{\sigma(\underline{x})} \models \sigma(\varphi_1)\\
&\quad\Leftrightarrow\quad \text{for all } \underline{a} \in A'_{\sigma(\underline{s})} \text{ holds } \mathcal{A}'|_\sigma, \left(v'^{\underline{a}}_{\sigma(\underline{x})}\right)\Big|_\sigma \models \varphi_1\\
&\quad\Leftrightarrow\quad \text{for all } \underline{a} \in A_{\underline{s}} \text{ holds } \mathcal{A}'|_\sigma, \left(v'|_\sigma\right)^{\underline{a}}_{\underline{x}} \models \varphi_1\\
&\quad\Leftrightarrow\quad \mathcal{A}'|_\sigma, v'|_\sigma \models \forall \underline{x}.\varphi_1
\end{aligned}$$

- $\varphi \equiv \exists \underline{x}.\varphi_1$:
  follows from the above.

■

Fact 2.34 (reduct by a bijective signature morphism)
For a bijective signature morphism $\sigma : SIG \to SIG'$ holds $(\cdot|_\sigma)^{-1} = \cdot|_{\sigma^{-1}}$.

Proof. Let $SIG = (S, F, P)$, $SIG' = (S', F', P')$, $\sigma : SIG \to SIG'$ a bijective signature morphism, and $\mathcal{A} \in Alg(SIG)$. By unfolding the definitions we get:

$$\begin{aligned}
(\mathcal{A}|_{\sigma^{-1}})|_\sigma &= \left((A_{\sigma^{-1}(s')})_{s'\in S'}, (\sigma^{-1}(f')_\mathcal{A})_{f'\in F'}, (\sigma^{-1}(p')_\mathcal{A})_{p'\in P'}\right)\Big|_\sigma\\
&= \left((A_{\sigma^{-1}(\sigma(s))})_{s\in S}, (\sigma^{-1}(\sigma(f))_\mathcal{A})_{f\in F}, (\sigma^{-1}(\sigma(p))_\mathcal{A})_{p\in P}\right)\\
&= \left((A_s)_{s\in S}, (f_\mathcal{A})_{f\in F}, (p_\mathcal{A})_{p\in P}\right)\\
&= \mathcal{A}.
\end{aligned}$$

■

2.6 Algebraic Specifications

Definition 2.35 (algebraic specifications)
A **specification** $SPEC = (SIG, X, Ax)$ consists of a signature $SIG = (S, F, P)$, a system $X = (X_s)_{s\in S}$ of countable, infinite sets X_s of variables for any sort $s \in S$, and a finite set of first-order formulas (so-called **axioms**) $Ax \subset FO(SIG, X)$. By $sig(SPEC) := SIG$ we denote the signature of $SPEC$.

The **semantics**[10] $SEM(SPEC)$ of $SPEC$ is the set of all generated models of the axioms, i.e. $SEM(SPEC) := \{\mathcal{A} \mid \mathcal{A} \in Gen(SIG), \mathcal{A} \models Ax\}$. $SPEC$ is said to be **monomorphic** if any two algebras in $SEM(SPEC)$ are isomorphic.[11] For a set of formulas $\Phi \subseteq DL(SIG, X, Pid)$ we write $SPEC \models \Phi$ to denote that $\mathcal{A} \models \Phi$ for all $\mathcal{A} \in SEM(SPEC)$.

[10] so-called **loose semantics**.

[11] Thus any inconsistent specification is a monomorphic one. In this point our definition differs from the one given in [7].

Definition 2.36 (renaming of specifications)
Given a specification $SPEC = (SIG, X, Ax)$ and a signature morphism $\sigma : SIG \to SIG'$, the **renaming of** $SPEC$ **via** σ, written $\sigma(SPEC)$, is the specification $\sigma(SPEC) = (SIG', \sigma(X), \sigma(Ax))$.

Fact 2.37 (semantics of renamed specifications)
For a specification $SPEC = (SIG, X, Ax)$ and a signature morphism $\sigma : SIG \to SIG'$ holds:

(1) *if σ is surjective, then $SEM(\sigma(SPEC))|_\sigma \subseteq SEM(SPEC)$.*

(2) *if σ is bijective, even $SEM(\sigma(SPEC))|_\sigma = SEM(SPEC)$.*

Proof. To prove (1) we consider any $\mathcal{A}' \in SEM(\sigma(SPEC))$. Then, by definition it holds $\mathcal{A}' \in Gen(\sigma(SIG))$ and $\mathcal{A}' \models \sigma(Ax)$. Using fact 2.32 and fact 2.33 we get $\mathcal{A}'|_\sigma \in Gen(SIG)$ and $\mathcal{A}'|_\sigma \models Ax$, i.e. $\mathcal{A}'|_\sigma \in SEM(SPEC)$.

For the proof of (2) we assume σ to be bijective. Then, because σ^{-1} is a surjective signature morphism too, we get

$$SEM(\sigma^{-1}(\sigma(SPEC)))|_{\sigma^{-1}} \subseteq SEM(\sigma(SPEC))$$

from (1), and so by fact 2.34

$$\begin{aligned} SEM(SPEC) &= SEM(\sigma^{-1}(\sigma(SPEC)))|_{\sigma^{-1}}|_\sigma \\ &\subseteq SEM(\sigma(SPEC))|_\sigma. \end{aligned}$$ ■

Definition 2.38 (disjoint union of specifications)
The **union** $SPEC + SPEC'$ of two specifications $SPEC = (SIG, X, Ax)$ and $SPEC' = (SIG', X', Ax')$ with disjoint signatures, i.e. $SIG \cap SIG' = \emptyset$, is the specification $SPEC + SPEC' := (SIG \cup SIG', X \cup X', Ax \cup Ax')$.

Fact 2.39 (semantics of union specifications)
For two specifications $SPEC$, $SPEC'$ with disjoint signatures holds

$$SEM(SPEC + SPEC') = SEM(SPEC) + SEM(SPEC').$$

Proof. Let $SPEC = (SIG, X, Ax)$ and $SPEC' = (SIG', X', Ax')$ two specifications with $SIG \cap SIG' = \emptyset$. Then holds:

$$\begin{aligned} & SEM(SPEC + SPEC') \\ &= \left\{ \mathcal{A} \,\middle|\, \mathcal{A} \in Gen(SIG \cup SIG'), \mathcal{A} \models Ax \cup Ax' \right\} \\ &= \left\{ \mathcal{B} + \mathcal{B}' \,\middle|\, \mathcal{B} \in Gen(SIG), \mathcal{B}' \in Gen(SIG'), \mathcal{B} + \mathcal{B}' \models Ax \cup Ax' \right\} \\ &= \left\{ \mathcal{B} + \mathcal{B}' \,\middle|\, \mathcal{B} \in Gen(SIG), \mathcal{B}' \in Gen(SIG'), \mathcal{B} \models Ax, \mathcal{B}' \models Ax' \right\} \\ &= \left\{ \mathcal{B} + \mathcal{B}' \,\middle|\, \mathcal{B} \in SEM(SPEC), \mathcal{B}' \in SEM(SPEC') \right\} \\ &= SEM(SPEC) + SEM(SPEC'). \end{aligned}$$

The second equation is due to fact 2.14, the third equation to fact 2.29. ■

3 Monomorphicity Criteria

In this section we present some criteria for a specification to be monomorphic (theorem 3.9). We prove that these criteria are both necessary and sufficient for monomorphicity.

Definition 3.1 (ground term relation)
Let $SIG = (S, F, P)$ and $\mathcal{A}, \mathcal{B} \in Alg(SIG)$. The **ground term relation** $GT_{\mathcal{A},\mathcal{B}}$ between $\mathcal{A}$ and $\mathcal{B}$ is the family $GT_{\mathcal{A},\mathcal{B}} = (GT_s)_{s\in S}$ of relations $GT_s \subseteq A_s \times B_s$ with $GT_s := \{(t_\mathcal{A}, t_\mathcal{B}) \,|\, t \in T_{F,s}\}$.

Fact 3.2 (properties of the ground term relation)
Let $SIG = (S, F, P)$ a signature and $\mathcal{A}, \mathcal{B} \in Alg(SIG)$. Then holds:

(1) $GT^{-1}_{\mathcal{A},\mathcal{B}} = GT_{\mathcal{B},\mathcal{A}}$.

(2) *$GT_{\mathcal{A},\mathcal{B}}$ is lefttotal iff $\mathcal{A} \in Gen(SIG)$.*

(3) *$GT_{\mathcal{A},\mathcal{B}}$ is righttotal iff $\mathcal{B} \in Gen(SIG)$.*

(4) *$GT_{\mathcal{A},\mathcal{B}}$ is closed against F wrt $\mathcal{A}$ and $\mathcal{B}$.*

Proof. (1), (2), and (3) are obvious. (4) is proved as follows. Let $f \in F_{\underline{s},s}$ for some $\underline{s} = s_1 \cdots s_n \in S^*$, $s \in S$, and $(a_1, b_1) \in GT_{s_1}, \ldots, (a_n, b_n) \in GT_{s_n}$. Then by definition 3.1 there are ground terms $t_1 \in T_{F,s_1}, \ldots, t_n \in T_{F,s_n}$ with $t_{1\mathcal{A}} = a_1, \ldots, t_{n\mathcal{A}} = a_n$ and $t_{1\mathcal{B}} = b_1, \ldots, t_{n\mathcal{B}} = b_n$. Consequently, we get

$$\begin{aligned}
&\big(f_\mathcal{A}(a_1, \ldots, a_n), f_\mathcal{B}(b_1, \ldots, b_n)\big)\\
&= \big(f_\mathcal{A}(t_{1\mathcal{A}}, \ldots, t_{n\mathcal{A}}), f_\mathcal{B}(t_{1\mathcal{B}}, \ldots, t_{n\mathcal{B}})\big)\\
&= \big(f(t_1, \ldots, t_n)_\mathcal{A}, f(t_1, \ldots, t_n)_\mathcal{B}\big)\\
&\in GT_s.
\end{aligned}$$

■

Lemma 3.3 (conclusion from closure against F)
Let $SIG = (S, F, P)$ and $\mathcal{A}, \mathcal{B} \in Alg(SIG)$. If a family $R = (R_s)_{s\in S}$ of relations $R_s \subseteq A_s \times B_s$ is closed against F wrt $\mathcal{A}$ and $\mathcal{B}$, then holds $R \supseteq GT_{\mathcal{A},\mathcal{B}}$.

Proof. We have to prove that for all $s \in S$ and all ground terms $t \in T_{F,s}$ holds $(t_\mathcal{A}, t_\mathcal{B}) \in R_s$. This is done by induction on the structure of ground terms, and simultaneous on tuples of ground terms. The base case says that $\{(t_\mathcal{A}, t_\mathcal{B}) \,|\, t \in set(())\}$ is a subset of R, which is trivial since $set(()) = \emptyset$. For the step case we consider any ground term $f(t_1, \ldots, t_n)$ with $f \in F_{\underline{s},s}$, $\underline{s} = s_1 \cdots s_n \in S^*$, $s \in S$, and $t_1 \in T_{F,s_1}, \ldots, t_n \in T_{F,s_n}$. By induction hypothesis is $(t_{1\mathcal{A}}, t_{1\mathcal{B}}) \in R_{s_1}, \ldots, (t_{n\mathcal{A}}, t_{n\mathcal{B}}) \in R_{s_n}$, and thus, by assumption $\big(f(t_1, \ldots, t_n)_\mathcal{A}, f(t_1, \ldots, t_n)_\mathcal{B}\big) \in R_s$. ■

Lemma 3.4 (characterization of closure against F)[12]
Let $SIG = (S, F, P)$, $\mathcal{A} \in Gen(SIG)$, $\mathcal{B} \in Alg(SIG)$, and $h = (h_s)_{s \in S}$ a family of total functions $h_s : A_s \to B_s$. Then are equivalent:

(a) *h is closed against F wrt $\mathcal{A}$ and $\mathcal{B}$.*

(b) *$h \supseteq GT_{\mathcal{A},\mathcal{B}}$.*

(c) *$h = GT_{\mathcal{A},\mathcal{B}}$.*

Proof. Since (a) $\Rightarrow$ (b) is a special case of lemma 3.3 and (c) $\Rightarrow$ (a) is already shown in fact 3.2(4), it remains to prove (b) $\Rightarrow$ (c). We assume $h \supseteq GT_{\mathcal{A},\mathcal{B}}$ and show that $h_s \subseteq GT_s$ for any $s \in S$. Let $(a, b) \in h_s$. Then, by generatedness of $\mathcal{A}$ there is a ground term $t \in T_{F,s}$ with $t_{\mathcal{A}} = a$. Thus, $(a, t_{\mathcal{B}}) = (t_{\mathcal{A}}, t_{\mathcal{B}}) \in GT_s$, and because of $h \supseteq GT_{\mathcal{A},\mathcal{B}}$ we get $(a, t_{\mathcal{B}}) \in h_s$. Since h_s is rightunique, it follows $b = t_{\mathcal{B}}$, and therefore, $(a, b) \in GT_s$. ■

Corollary 3.5 (uniqueness of homomorphisms)
Let $SIG = (S, F, P)$ a signature, $\mathcal{A} \in Gen(SIG)$, and $\mathcal{B} \in Alg(SIG)$. Then holds:

(1) *There exists at most one family $h = (h_s)_{s \in S}$ of total functions $h_s : A_s \to B_s$ which is closed against F wrt $\mathcal{A}$ and $\mathcal{B}$.*

(2) *There exists at most one weak homomorphism from $\mathcal{A}$ to $\mathcal{B}$.*

(3) *There exists at most one homomorphism from $\mathcal{A}$ to $\mathcal{B}$.*

Proof. (1) is a consequence of lemma 3.4. (2) and (3) follow from (1). ■

Corollary 3.6 (ground term relation captures homomorphisms)
Let SIG a signature and $\mathcal{A}, \mathcal{B} \in Gen(SIG)$. Then holds:

(1) *For any weak homomorphism h from $\mathcal{A}$ to $\mathcal{B}$ holds $h = GT_{\mathcal{A},\mathcal{B}}$.*

(2) *For any homomorphism h from $\mathcal{A}$ to $\mathcal{B}$ holds $h = GT_{\mathcal{A},\mathcal{B}}$.*

(3) *For any weak homomorphism h from $\mathcal{B}$ to $\mathcal{A}$ holds $h = GT^{-1}_{\mathcal{A},\mathcal{B}}$.*

(4) *For any homomorphism h from $\mathcal{B}$ to $\mathcal{A}$ holds $h = GT^{-1}_{\mathcal{A},\mathcal{B}}$.*

Proof. (1) and (2) are consequences of lemma 3.4. (3) and (4) follow from (1) and (2), together with fact 3.2(1). ■

[12]This and the following results are in some sense similar to ([7], Fact 2.2.6).

Remark. Corollary 3.6 says that the homomorphism from a generated algebra $\mathcal{A}$ to an algebra $\mathcal{B}$ — if existing at all — can be constructed by building the ground term relation $GT_{\mathcal{A},\mathcal{B}} = (GT_s)_{s \in S}$ with $GT_s := \{(t_\mathcal{A}, t_\mathcal{B}) \mid t \in T_{F,s}\}$.

Lemma 3.7 (isomorphicity and homomorphisms)
Let SIG a signature and $\mathcal{A}, \mathcal{B} \in Gen(SIG)$. Then are equivalent:

(a) *$\mathcal{A}$ and $\mathcal{B}$ are isomorphic.*

(b) *there is a homomorphism g from $\mathcal{A}$ to $\mathcal{B}$ and a homomorphism h from $\mathcal{B}$ to $\mathcal{A}$.*

(c) *there is a weak homomorphism g from $\mathcal{A}$ to $\mathcal{B}$ and a weak homomorphism h from $\mathcal{B}$ to $\mathcal{A}$.*

Proof. Since (a) $\Rightarrow$ (b) follows from fact 2.17(2) and (b) $\Rightarrow$ (c) is obvious, it remains to prove (c) $\Rightarrow$ (a). Let g a weak homomorphism from $\mathcal{A}$ to $\mathcal{B}$ and h a weak homomorphism from $\mathcal{B}$ to $\mathcal{A}$. We show that g (as well as h) is even a isomorphism, which is divided up in two assertions:

(i) g is bijective:
Because of corollary 3.6 it holds for any $s \in S$, $t \in T_{F,s}$ that:

$$h_s(g_s(t_\mathcal{A})) = h_s(t_\mathcal{B}) = t_\mathcal{A}.$$

Due to the generatedness of $\mathcal{A}$ this is sufficient for[13] $h_s \circ g_s = id_{A_s}$. Similarly it can be shown that $g_s \circ h_s = id_{B_s}$. So[14], g_s is bijective with $g_s^{-1} = h_s$.

(ii) g^{-1} is monotonic in P wrt $\mathcal{B}$ and $\mathcal{A}$:
This follows from (i), i.e. from $g_s^{-1} = h_s$ together with the assumption that h is a weak homomorphism from $\mathcal{B}$ to $\mathcal{A}$. ■

Lemma 3.8 (monomorphicity and weak homomorphisms)
A specification $SPEC$ is monomorphic if and only if for any two algebras $\mathcal{A}, \mathcal{B} \in SEM(SPEC)$ exists a weak homomorphism from $\mathcal{A}$ to $\mathcal{B}$.

Proof. If $SPEC$ is monomorphic, then, by definition, there exists a homomorphism h from $\mathcal{A}$ to $\mathcal{B}$ for any two algebras $\mathcal{A}, \mathcal{B} \in SEM(SPEC)$. To prove the other implication, let $\mathcal{A}, \mathcal{B} \in SEM(SPEC)$ be any two models of $SPEC$. From assumption we get the existence of a weak homomorphism g from $\mathcal{A}$ to $\mathcal{B}$, but — using a symmetry argument — also the existence of a weak homomorphism h from $\mathcal{B}$ to $\mathcal{A}$. Due to lemma 3.7 this is sufficient for $\mathcal{A}$ and $\mathcal{B}$ to be isomorphic. ■

[13]For a set A the identity function $id_A : A \to A$ is defined by $id_A(x) := x$.
[14]See any textbook on analysis.

Theorem 3.9 (monomorphicity criteria)
Let $SIG = (S, F, P)$, $SPEC$ a specification with $sig(SPEC) = SIG$. Then the following statements are equivalent:

(a) *$SPEC$ is monomorphic.*

(b) *for any two algebras $\mathcal{A}, \mathcal{B} \in SEM(SPEC)$ exists a family $R = (R_s)_{s \in S}$ of relations $R_s \subseteq A_s \times B_s$ such that:*

- *R is rightunique,*
- *R is closed against F wrt $\mathcal{A}$ and $\mathcal{B}$,*
- *R is monotonic in P wrt $\mathcal{A}$ and $\mathcal{B}$.*

(c) *for any two algebras $\mathcal{A}, \mathcal{B} \in SEM(SPEC)$ exists a family $R = (R_s)_{s \in S}$ of relations $R_s \subseteq A_s \times B_s$ such that:*

- *R is rightunique,*
- *$R \supseteq GT_{\mathcal{A},\mathcal{B}}$,*
- *R is monotonic in P wrt $\mathcal{A}$ and $\mathcal{B}$.*

(d) *for any two algebras $\mathcal{A}, \mathcal{B} \in SEM(SPEC)$ holds*

- *$GT_{\mathcal{A},\mathcal{B}}$ is rightunique,*
- *$GT_{\mathcal{A},\mathcal{B}}$ is monotonic in P wrt $\mathcal{A}$ and $\mathcal{B}$.*

(e) *for any two algebras $\mathcal{A}, \mathcal{B} \in SEM(SPEC)$ holds*

- *$GT_{\mathcal{A},\mathcal{B}}$ is leftunique,*
- *$GT_{\mathcal{B},\mathcal{A}}$ is monotonic in P wrt $\mathcal{B}$ and $\mathcal{A}$.*

Proof. The equivalence of (a) – (e) follows from the following implications, which we prove separately.

(a) $\Rightarrow$ (b): Let $SPEC$ be a monomorphic specification. Then for any $\mathcal{A}, \mathcal{B} \in SEM(SPEC)$ exists a weak homomorphism h from $\mathcal{A}$ to $\mathcal{B}$. By choosing $R := h$ we get a family of relations with all the properties required in (b).

(b) $\Rightarrow$ (c): Follows directly from lemma 3.3.

(c) $\Rightarrow$ (d): We assume the existence of a family R of relations, which fulfills the conditions in (c) and prove $R = GT_{\mathcal{A},\mathcal{B}}$ as follows: Due to the assumption $R \supseteq GT_{\mathcal{A},\mathcal{B}}$ and fact 3.2(3) is R righttotal, and therefore a family of total functions. Using lemma 3.4 yields $R = GT_{\mathcal{A},\mathcal{B}}$.

(d) $\Rightarrow$ (e): By assumption, for any two algebras $\mathcal{A}, \mathcal{B} \in SEM(SPEC)$ is $GT_{\mathcal{B},\mathcal{A}}$ rightunique and monotonic in P wrt $\mathcal{B}$ and $\mathcal{A}$. Using fact 3.2(1) and fact 2.5(2) we get that $GT_{\mathcal{A},\mathcal{B}}$ is leftunique.

(e) $\Rightarrow$ (a): By assumption, for any two algebras $\mathcal{A}, \mathcal{B} \in SEM(SPEC)$ is $GT_{\mathcal{B},\mathcal{A}}$ leftunique and $GT_{\mathcal{A},\mathcal{B}}$ monotonic in P wrt $\mathcal{A}$ and $\mathcal{B}$. Using fact 3.2(1) and fact 2.5(2) we get that $GT_{\mathcal{A},\mathcal{B}}$ is rightunique. Due to fact 3.2(3), is $GT_{\mathcal{A},\mathcal{B}}$ a family $(h_s)_{s\in S}$ of total functions $h_s : A_s \to B_s$ which is monotonic in P wrt $\mathcal{A}$ and $\mathcal{B}$. Together with fact 3.2(4) follows that $GT_{\mathcal{A},\mathcal{B}}$ is a weak homomorphism from $\mathcal{A}$ to $\mathcal{B}$. Due to lemma 3.8 this is sufficient for $SPEC$ to be monomorphic.

■

4 Proof Obligations in Dynamic Logic

This section aims to reformulate the monomorphicity criteria presented in theorem 3.9 as proof obligations in dynamic logic (theorem 4.7). The reformulation is based on the following techniques:

- copying the specification:[15]
  The phrase "for any two algebras $\mathcal{A}, \mathcal{B} \in SEM(SPEC)$" in theorem 3.9 needs special treatment since formulas (over $sig(SPEC)$) express properties of a *single* model and not relations between *two* algebras. We tackle this problem by using a renamed copy $SPEC' := \sigma(SPEC)$ of $SPEC$. If this signature morphism σ is bijective, then the above phrase is equivalent to "for any $\mathcal{A} \in SEM(SPEC)$ and any $\mathcal{A}' \in SEM(SPEC')$" (where $\mathcal{B}$ has to be replaced by $\mathcal{A}'|_\sigma$). Provided the signatures $sig(SPEC)$ and $sig(SPEC')$ to be disjoint, this phrase is equivalent to (cf. fact 2.39) "for any $\mathcal{A}'' \in SEM(SPEC + SPEC')$" (where $\mathcal{A}$ is replaced by the $sig(SPEC)$-reduct of $\mathcal{A}''$ and $\mathcal{A}'$ is replaced by $sig(SPEC')$-reduct of $\mathcal{A}''$). To summarize, in order to prove monomorphicity of $SPEC$, we reason on the union of $SPEC$ and a bijective renaming of it.

- proving existence constructively:
  For proving the criteria (b) or (c) of theorem 3.9 the existence of "a family $R = (R_s)_{s\in S}$ of relations ..." has to be shown. This can be done constructively by explicitly giving a family of (possibly indeterministic) procedures (one procedure for each sort $s \in S$) and proving that the input-output relations they compute satisfy the required properties. Because of the copying of the specification (as mentioned just above)

[15] The idea of using this technique is due to a hint of Wolfgang Reif.

this procedures map from $sig(SPEC)$ to $sig(\sigma(SPEC))$, i.e. the procedure for $s \in S$ maps from s to $\sigma(s)$. We will call such a family of procedures a *mapping program*.

- representing GT by a uniform mapping program:
  The ground term relation $GT_{\mathcal{A},\mathcal{B}}$ is a specific family $R = (R_s)_{s \in S}$ of relations $R_s \subseteq A_s \times B_s$. Therefore, it can be represented as the input-output relations of a mapping program. In fact, such a mapping program can be constructed automatically; we will call it the *uniform mapping program*.

Definition 4.1 (mapping program)
Let $SIG = (S, F, P)$ a signature, $\sigma : SIG \to SIG'$ a signature morphism with[16] $SIG \cap SIG' = \emptyset$. X a system of variables for $SIG \cup SIG'$, and Pid a system of procedure identifiers for $SIG \cup SIG'$. A **mapping program** MP for σ is a pair (mp, δ) where $mp : S \to Pid$ assigns a procedure identifier $mp(s) \in Pid_{s,\sigma(s)}$ to any sort $s \in S$, and $\delta \in PD(SIG \cup SIG', X, Pid)^*$ is a tuple of procedure declarations.[17] Given $s \in S$, $t \in T_{F,s}(X)$, $y \in X_{\sigma(s)}$, and $\kappa \in T_{F_+,ctr}$, we abbreviate **proc** δ **in begin** $mp(s)(t; y)$ **end** by $MP(s)(t; y)$ and **proc** δ **times** κ **in begin** $mp(s)(t; y)$ **end** by $\kappa \# MP(s)(t; y)$.

For $\mathcal{A} \in Alg(SIG)$, $\mathcal{A}' \in Alg(SIG')$, and $s \in S$, we set

$$MP(s)_{\mathcal{A}+\mathcal{A}'} := \left\{ (v(x), v'(y)) \;\middle|\; \begin{array}{l} v, v' \in Val(X, \mathcal{A} + \mathcal{A}'),\ x \in X_s,\ y \in X_{\sigma(s)} \\ v\ [\![MP(s)(x; y)]\!]_{\mathcal{A}+\mathcal{A}'}\ v' \end{array} \right\}$$

and $MP_{\mathcal{A}+\mathcal{A}'} := \Big(MP(s)_{\mathcal{A}+\mathcal{A}'}\Big)_{s \in S}$.

Definition 4.2 (uniform mapping program MP^u)
Let $SIG = (S, F, P)$ a signature, $\sigma : SIG \to SIG'$ a signature morphism with $SIG \cap SIG' = \emptyset$, X a system of variables for $SIG \cup SIG'$, and Pid a system of procedure identifiers for $SIG \cup SIG'$. The **uniform mapping program** $MP^u := (mp^u, \delta^u)$ for σ is defined as follows. $mp^u : S \to Pid$ is any total function with $mp^u(s) \in Pid_{s,\sigma(s)}$ for all $s \in S$. For each $s \in S$ we fix a $x_s \in X_s$, and a $y_s \in X_{\sigma(s)}$. Then we set[18]

$$\delta^u \;:=\; \left\{ (mp^u(s))(x_s; \mathbf{var}\ y_s). \bigcup FRAGS(s, x_s, y_s) \;\middle|\; s \in S \right\}$$

[16]This is required to guarantee the sum $\mathcal{A} + \mathcal{A}'$ of $\mathcal{A} \in Alg(SIG)$ and $\mathcal{A}' \in Alg(SIG')$ to be well-defined.

[17]Because we allow δ to contain (auxiliary) procedures not in the range of mp a mapping program cannot be described as a family of procedures indexed with S.

[18]The indeterminism of the uniform mapping program introduced by the $\bigcup$-command is inherent in the problem of computing the ground term relation, i.e. in general there is no effective way to find an ordering $\{f_1, \ldots, f_n\}$ of the function symbols in $F_{\underline{s},s}$ such that $frag(f_{i+1}, x_s, y_s)$ has only to be executed if execution of $frag(f_i, x_s, y_s)$ gets in the **else** branch. An example illustrating this point is given in appendix A.

where[19]

$$FRAGS(s, x_s, y_s) \ := \ \left\{ \, frag(f, x_s, y_s) \, \middle| \, f \in F_{\underline{s},s} \, , \, \underline{s} \in S^* \, \right\}$$

with the fragments $frag(f, x_s, y_s)$ for $f \in F_{\underline{s},s}$, $\underline{s} = s_1 \cdots s_n \in S^*$ defined as follows. Choose new (and different) variables $x_1 \cdots x_n \in X_{\underline{s}}$, $y_1 \cdots y_n \in X_{\sigma(\underline{s})}$. Then define

```
$frag(f, x_s, y_s) :\equiv$ var $x_1 =?, \ldots, x_n =?$ in
                  if $x_s = f(x_1, \ldots, x_n)$
                  then var $y_1 =?, \ldots, y_n =?$ in
                       $\big(mp^u(s_1)(x_1; y_1); \ldots; mp^u(s_n)(x_n; y_n);$
                        $y_s := \sigma(f)(y_1, \ldots, y_n)\big)$
                  else abort.
```

Remark. The values assigned to $y_1, \ldots, y_n$ in the random variable declaration **var** $y_1 =?, \ldots, y_n =?$ **in** are not used. Therefore, instead of it we can use any deterministic variable declaration **var** $y_1 = t_1, \ldots, y_n = t_n$ **in** $\ldots$, where the t_i are arbitrary ground terms of sort $\sigma(s_i)$, i.e. $t_i \in T_{\sigma(F),\sigma(s_i)}$. This is deductively more tractable, since there are less existential quantors (due to the indeterministic assignments) involved. Notice, that these t_i actually exist, if one restricts oneself to sensible signatures. Furthermore, such ground terms t_i can be effectively computed, for instance like in the algorithm "kanonische Auswahl" presented in ([2], page 173).

Lemma 4.3 (properties of MP^u)
Let $SIG = (S, F, P)$ a signature, $\sigma : SIG \to SIG'$ a signature morphism with $SIG \cap SIG' = \emptyset$, X a system of variables for $SIG \cup SIG'$, MP^u the uniform mapping program for σ, $\mathcal{A} \in Alg(SIG)$, $\mathcal{A}' \in Alg(SIG')$, and $s \in S$.

(1) *For all $f \in F_{s_1 \cdots s_n, s}$ ($s_1 \cdots s_n \in S^*$) and $t_i \in T_{F,s_i}$ ($i = 1, \ldots, n$) with $({t_i}_{\mathcal{A}}, \sigma(t_i)_{\mathcal{A}'}) \in MP^u(s_i)_{\mathcal{A}+\mathcal{A}'}$ holds $\big(f(t_1, \ldots, t_n)_{\mathcal{A}}, \sigma(f(t_1, \ldots, t_n))_{\mathcal{A}'}\big) \in MP^u(s)_{\mathcal{A}+\mathcal{A}'}$.*

(2) *For all $x \in X_s$, $y \in X_{\sigma(s)}$, $v, v' \in Val(X, \mathcal{A} + \mathcal{A}')$, and $\kappa \in T_{F_+,ctr}$ with $v \, [\![\kappa \# MP^u(s)(x; y)]\!]_{\mathcal{A}+\mathcal{A}'} \, v'$ holds $(v(x), v'(y)) \in \left\{ (t_{\mathcal{A}}, \sigma(t)_{\mathcal{A}'}) \middle| t \in T_{F,s} \right\}$.*

(3) $MP^u(s)_{\mathcal{A}+\mathcal{A}'} \ \supseteq \ \left\{ (t_{\mathcal{A}}, \sigma(t)_{\mathcal{A}'}) \middle| t \in T_{F,s} \right\}$.

[19]More accurately δ^u and $FRAGS(s, x_s, y_s)$ (cf. definition 2.22) should be tuples rather than finite sets. However, since semantically the ordering in these tuples turns out to be irrelevant, we simplify the presentation by abstracting from a potential ordering.

Proof. The proofs of (1) and (2) are quite technical. They mainly work by unfolding the definitions, especially definition 2.28. We tacitly make use of the fact that for all $\delta \in PD(SIG \cup SIG', X, Pid)^*$ and all commands $\alpha \in CMD(SIG \cup SIG', X, Pid)$ holds

$$[\![\textbf{proc } \delta \textbf{ in } \alpha]\!] \;=\; \bigcup_{\kappa \in T_{F_+,ctr}} [\![\textbf{proc } \delta \textbf{ times } \kappa \textbf{ in } \alpha]\!].$$

Since δ^u is the only procedure declaration occurring in this proof, we simply write α as abbreviation for **proc** δ^u **in** α, and $\kappa\#\alpha$ as abbreviation for **proc** δ^u **times** κ **in** α.

For the proof of (1) let $\underline{s} = s_1 \cdots s_n \in S^*$, $s \in S$, $f \in F_{s_1\cdots s_n,s}$, and $t_i \in T_{F,s_i}$ $(i = 1,\ldots,n)$ with $(t_{i\mathcal{A}}, \sigma(t_i)_{\mathcal{A}'}) \in MP^u(s_i)_{\mathcal{A}+\mathcal{A}'}$. Furthermore, let $x \in X_s$, $y \in X_{\sigma(s)}$, $x_1 \cdots x_n \in X_{\underline{s}}$, $y_1 \cdots y_n \in X_{\sigma(\underline{s})}$ be different variables. From the assumption $(t_{i\mathcal{A}}, \sigma(t_i)_{\mathcal{A}'}) \in MP^u(s_i)_{\mathcal{A}+\mathcal{A}'}$ follows that (for $i = 1,\ldots,n$) there are $v_i, v_i' \in Val(X, \mathcal{A}+\mathcal{A}')$ with $v_i(x_i) = t_{i\mathcal{A}}$, $v_i'(y_i) = \sigma(t_i)_{\mathcal{A}'}$, and $v_i\, [\![MP^u(s_i)(x_i; y_i)]\!]_{\mathcal{A}+\mathcal{A}'}\, v_i'$. As MP^u contains side-effect free procedures only, it is $v_i\, [\![MP^u(s_i)(x_i; y_i)]\!]_{\mathcal{A}+\mathcal{A}'}\, v_{i\,y_i}^{\sigma(t_i)_{\mathcal{A}'}}$ even *for all* $v_i \in Val(X, \mathcal{A}+\mathcal{A}')$ with $v_i(x_i) = t_{i\mathcal{A}}$. Since all variables $x_1,\ldots,x_n,y_1,\ldots,y_n$ are different from each other, we get

$$v\, [\![mp^u(s_1)(x_1; y_1); \ldots; mp^u(s_n)(x_n; y_n)]\!]\, v_{y_1\cdots y_n}^{\sigma(t_1)_{\mathcal{A}'}\cdots\sigma(t_n)_{\mathcal{A}'}}$$

for all $v \in Val(X, \mathcal{A}+\mathcal{A}')$ with $v(x_i) = t_{i\mathcal{A}}$ (for $i = 1,\ldots,n$). Thus, for all $v \in Val(X, \mathcal{A}+\mathcal{A}')$ with $v(x_i) = t_{i\mathcal{A}}$ exists a $v' \in Val(X, \mathcal{A}+\mathcal{A}')$ with $v\, [\![mp^u(s_1)(x_1; y_1); \ldots; mp^u(s_n)(x_n; y_n); y := \sigma(f)(y_1,\ldots,y_n)]\!]\, v'$ and

$$v'(y) = \sigma(f)_{\mathcal{A}'}(\sigma(t_1)_{\mathcal{A}'},\ldots,\sigma(t_n)_{\mathcal{A}'}) = \sigma(f(t_1,\ldots,t_n))_{\mathcal{A}'}.$$

So, for all $v \in Val(X, \mathcal{A}+\mathcal{A}')$ with $v(x_i) = t_{i\mathcal{A}}$ (for $i = 1,\ldots,n$) and $v(x) = f(x_1,\ldots,x_n)_{v,\mathcal{A}+\mathcal{A}'} = f(t_1,\ldots,t_n)_{\mathcal{A}}$ there is some $v' \in Val(X, \mathcal{A}+\mathcal{A}')$ with

```
v [[ if $x = f(x_1,\ldots,x_n)$
      then var $y_1 =?,\ldots,y_n =?$ in
            $\big(mp^u(s_1)(x_1; y_1); \ldots; mp^u(s_n)(x_n; y_n);$
             $y := \sigma(f)(y_1,\ldots,y_n)\big)$
      else abort ]] $v'$
```

and $v'(y) = \sigma(f(t_1,\ldots,t_n))_{\mathcal{A}'}$. Consequently, for all $v \in Val(X, \mathcal{A}+\mathcal{A}')$ with $v(x) = f(t_1,\ldots,t_n)_{\mathcal{A}}$ there is some $v' \in Val(X, \mathcal{A}+\mathcal{A}')$ with $v'(y) = \sigma(f(t_1,\ldots,t_n))_{\mathcal{A}'}$ and $v\, [\![frag(f,x,y)]\!]\, v'$, i.e. $v\, [\![MP(s)(x; y)]\!]\, v'$. Since there is of course a $v \in Val(X, \mathcal{A}+\mathcal{A}')$ with $v(x) = f(t_1,\ldots,t_n)_{\mathcal{A}}$, it follows $\big(f(t_1,\ldots,t_n)_{\mathcal{A}}, \sigma(f(t_1,\ldots,t_n))_{\mathcal{A}'}\big) \in MP^u(s)_{\mathcal{A}+\mathcal{A}'}$.

For the proof of (2) let $v, v' \in Val(X, \mathcal{A}+\mathcal{A}')$, $s \in S$, $x \in X_s$, and $y \in X_{\sigma(s)}$. We have to show that for all ground terms $\kappa \in T_{F_+,ctr}$ holds that $v\,[\![\kappa\#MP^u(s)(x;y)]\!]_{\mathcal{A}+\mathcal{A}'}\,v'$ implies $(v(x), v'(y)) \in \{(t_\mathcal{A}, \sigma(t)_{\mathcal{A}'}) | t \in T_{F,s}\}$. This is done by structural induction on counter terms κ. In the base case, where $\kappa \equiv czero$, $v\,[\![\kappa\#MP^u(s)(x;y)]\!]_{\mathcal{A}+\mathcal{A}'}\,v'$ does not hold, and so the implication is true by triviality. In the step case is $\kappa \equiv csucc(\kappa')$ for some $\kappa' \in T_{F_+,ctr}$. Assuming $v\,[\![\kappa\#MP^u(s)(x;y)]\!]_{\mathcal{A}+\mathcal{A}'}\,v'$ we get from definition 2.28 that $v\,[\![\kappa'\#frag(f,x,y)]\!]_{\mathcal{A}+\mathcal{A}'}\,v'$ for some $f \in F_{\underline{s},s}$ ($\underline{s} = s_1 \cdots s_n \in S^*$). Consequently, there are some $a_1 \cdots a_n \in A_{\underline{s}}$ and a $v'' \in Val(X, \mathcal{A}+\mathcal{A}')$ such that ($x_1 \cdots x_n \in X_{\underline{s}}$, $y_1 \cdots y_n \in X_{\sigma(\underline{s})}$ are new and different variables)

$$\begin{aligned} v^{a_1\cdots a_n}_{x_1\cdots x_n}\,[\![\kappa'\#\ & \mathbf{if}\ x = f(x_1,\ldots,x_n) \\ & \mathbf{then\ var}\ y_1 =?,\ldots,y_n =?\ \mathbf{in} \\ & \qquad \big(mp^u(s_1)(x_1;y_1);\ldots;mp^u(s_n)(x_n;y_n); \\ & \qquad\ y := \sigma(f)(y_1,\ldots,y_n)\big) \\ & \mathbf{else\ abort}\ ]\!]\,v'' \end{aligned}$$

and $v''(y) = v'(y)$. Since the **else** branch aborts the condition of the **if** construct must be evaluated to true in $\mathcal{A}+\mathcal{A}'$ under $v^{a_1\cdots a_n}_{x_1\cdots x_n}$. That is, it holds $v(x) = v^{a_1\cdots a_n}_{x_1\cdots x_n}(x) = f_\mathcal{A}(a_1,\ldots,a_n)$. By unfolding the semantics of **var** $y_1 =?,\ldots,y_n =?$ **in** $\ldots$ we get that there are $b_1 \cdots b_n \in A_{\sigma(\underline{s})}$ and a $v''' \in Val(X, \mathcal{A}+\mathcal{A}')$ with (we abbreviate $v_0 := v^{a_1\cdots a_n, b_1\cdots b_n}_{x_1\cdots x_n, y_1\cdots y_n}$)

$$\begin{aligned} v_0\,[\![\kappa'\#\ & mp^u(s_1)(x_1;y_1);\ldots;mp^u(s_n)(x_n;y_n); \\ & y := \sigma(f)(y_1,\ldots,y_n)\ ]\!]\,v''' \end{aligned}$$

and $v'''(y) = v''(y)$. Let $v_1,\ldots,v_n \in Val(X, \mathcal{A}+\mathcal{A}')$ be valuations with $v_{i-1}\,[\![\kappa'\#mp^u(s_i)(x_i;y_i)]\!]\,v_i$ for $i = 1,\ldots,n$, and $v_n\,[\![y := \sigma(f)(y_1,\ldots,y_n)]\!]\,v'''$, i.e. $v'''(y) = \sigma(f)_{\mathcal{A}'}(v_n(y_1),\ldots,v(y_n))$. Then, by induction hypothesis, it holds $(v_{i-1}(x_i), v_i(y_i)) \in \{(t_\mathcal{A}, \sigma(t)_{\mathcal{A}'}) | t \in T_{F,s_i}\}$ for $i = 1,\ldots,n$. As MP^u contains side-effect free procedures only and all the variables are different from each other, it follows that $(v_0(x_i), v_n(y_i)) \in \{(t_\mathcal{A}, \sigma(t)_{\mathcal{A}'}) | t \in T_{F,s_i}\}$ for $i = 1,\ldots,n$. So we get

$$\big(f_\mathcal{A}(v_0(x_1),\ldots,v_0(x_n)), \sigma(f)_{\mathcal{A}'}(v_n(y_1),\ldots,v(y_n))\big) \in \{(t_\mathcal{A}, \sigma(t)_{\mathcal{A}'}) | t \in T_{F,s_i}\}.$$

It remains to summarize the above. We have already shown that

$$\begin{aligned} v(x) &= f_\mathcal{A}(a_1,\ldots,a_n) = f_\mathcal{A}(v_0(x_1),\ldots,v_0(x_n)) \\ v'(y) &= v''(y) = v'''(y) = \sigma(f)_{\mathcal{A}'}(v_n(y_1),\ldots,v(y_n)), \end{aligned}$$

thus $(v(x), v'(y)) \in \{(t_\mathcal{A}, \sigma(t)_{\mathcal{A}'}) | t \in T_{F,s_i}\}$.

The proof of (3) works by structural induction on ground terms and is quite similar to the one of lemma 3.3. The induction step is already proven in (1) of the current lemma. ■

Theorem 4.4 (*MP^u* **computes the ground term relation)**
Let $\sigma : SIG \to SIG'$ a signature morphism with $SIG \cap SIG' = \emptyset$, $\mathcal{A} \in Alg(SIG)$, $\mathcal{A}' \in Alg(SIG')$, and MP^u the uniform mapping program for σ. Then holds:

$$MP^u_{\mathcal{A}+\mathcal{A}'} \quad = \quad GT_{\mathcal{A},\mathcal{A}'|_\sigma}.$$

Proof. Let $SIG := (S, F, P)$,$\mathcal{A} \in Alg(SIG)$, and $\mathcal{A}' \in Alg(SIG')$. We have to prove that $MP^u(s)_{\mathcal{A}+\mathcal{A}'} = \{(t_\mathcal{A}, t_{\mathcal{A}'|_\sigma})|t \in T_{F,s}\}$ for all $s \in S$. Since $t_{\mathcal{A}'|_\sigma} = \sigma(t)_{\mathcal{A}'}$ for all $t \in T_F$ (see fact 2.31), it remains to show that for all $s \in S$ holds:

$$MP^u(s)_{\mathcal{A}+\mathcal{A}'} \quad = \quad \left\{ (t_\mathcal{A}, \sigma(t)_{\mathcal{A}'}) \,\middle|\, t \in T_{F,s} \right\}.$$

The inclusion $\supseteq$ is already proved in lemma 4.3(3). For the other inclusion we argue as follows (X is a system of variables for $SIG \cup SIG'$):

$$\begin{aligned}
& MP^u(s)_{\mathcal{A}+\mathcal{A}'} \\
= & \left\{ (v(x), v'(y)) \,\middle|\, \begin{array}{l} v, v' \in Val(X, \mathcal{A}+\mathcal{A}'), x \in X_s, y \in X_{\sigma(s)} \\ v \, [\![MP^u(s)(x;y)]\!]_{\mathcal{A}+\mathcal{A}'} \, v' \end{array} \right\} \\
= & \left\{ (v(x), v'(y)) \,\middle|\, \begin{array}{l} v, v' \in Val(X, \mathcal{A}+\mathcal{A}'),\ x \in X_s,\ y \in X_{\sigma(s)} \\ \kappa \in T_{F_+,ctr},\ v \, [\![\kappa \# MP^u(s)(x;y)]\!]_{\mathcal{A}+\mathcal{A}'} \, v' \end{array} \right\} \\
\subseteq & \left\{ (v(x), v'(y)) \,\middle|\, \begin{array}{l} v, v' \in Val(X, \mathcal{A}+\mathcal{A}'),\ x \in X_s,\ y \in X_{\sigma(s)} \\ (v(x), v'(y)) \in \{(t_\mathcal{A}, \sigma(t)_{\mathcal{A}'})|t \in T_{F,s}\} \end{array} \right\} \\
= & \left\{ (t_\mathcal{A}, \sigma(t)_{\mathcal{A}'}) \,\middle|\, t \in T_{F,s} \right\}.
\end{aligned}$$

In the inclusion $\subseteq$ we have made use of lemma 4.3(2). ■

We now work out, how the primitives of the criteria given in theorem 3.9 can be expressed in terms of formulas from dynamic logic.

Definition 4.5 (proof obligations)
Let $SIG = (S, F, P)$ a signature, $\sigma : SIG \to SIG'$ a signature morphism with $SIG \cap SIG' = \emptyset$, X a system of variables for $SIG \cup SIG'$, and MP a mapping program for σ. For any $s \in S$, $x_s \in X_s$, $y_s \in X_{\sigma(s)}$ we abbreviate:

$$MP_s(x_s, y_s) \quad :\equiv \quad \langle MP(s)(x_s; z_s) \rangle \; z_s = y_s$$

where z_s is any variable from $X_{\sigma(s)} \setminus \{y_s\}$. For each $s \in S$ we denote the variables in X_s by $x_s, x1_s, x2_s$ and the variables in $X_{\sigma(s)}$ by $y_s, y1_s, y2_s$. In all the formulas below we assume all variables with different names not to be equal.

- $VC_{\text{leftunique}}(MP, \sigma) :=$
  $\bigcup_{s \in S} \left\{ MP_s(x1_s, y_s) \;\wedge\; MP_s(x2_s, y_s) \;\rightarrow\; x1_s = x2_s \right\}$

- $VC_{\text{rightunique}}(MP, \sigma) :=$
  $\bigcup_{s \in S} \left\{ MP_s(x_s, y1_s) \;\wedge\; MP_s(x_s, y2_s) \;\rightarrow\; y1_s = y2_s \right\}$

- $VC_{\text{closed against F}}(MP, \sigma) :=$
  $$\bigcup_{s_1,\ldots,s_n,s \in S} \; \bigcup_{f \in F_{s_1 \cdots s_n,s}} \Big\{ MP_{s_1}(x_{s_1}, y_{s_1}) \;\wedge\; \cdots \;\wedge\; MP_{s_n}(x_{s_n}, y_{s_n}) \;\rightarrow\; MP_s(f(x_{s_1}, \ldots, x_{s_n}), (\sigma(f))(y_{s_1}, \ldots, y_{s_n})) \Big\}$$

- $VC_{\text{supset of GT}}(MP, \sigma) :=$
  $\bigcup_{s \in S} \left\{ MP^u_s(x_s, y_s) \;\rightarrow\; MP_s(x_s, y_s) \right\}$

- $VC_{\text{monotonic in P}}(MP, \sigma) :=$
  $$\bigcup_{s_1,\ldots,s_n \in S} \; \bigcup_{p \in P_{s_1 \cdots s_n}} \Big\{ MP_{s_1}(x_{s_1}, y_{s_1}) \;\wedge\; \cdots \;\wedge\; MP_{s_n}(x_{s_n}, y_{s_n}) \;\wedge\; p(x_{s_1}, \ldots, x_{s_n}) \;\rightarrow\; (\sigma(p))(y_{s_1}, \ldots, y_{s_n}) \Big\}$$

- $VC_{\text{inv monotonic in P}}(MP, \sigma) :=$
  $$\bigcup_{s_1,\ldots,s_n \in S} \; \bigcup_{p \in P_{s_1 \cdots s_n}} \Big\{ MP_{s_1}(x_{s_1}, y_{s_1}) \;\wedge\; \cdots \;\wedge\; MP_{s_n}(x_{s_n}, y_{s_n}) \;\wedge\; (\sigma(p))(y_{s_1}, \ldots, y_{s_n}) \;\rightarrow\; p(x_{s_1}, \ldots, x_{s_n}) \Big\}$$

Lemma 4.6 (semantics of proof obligations)
Let $SIG = (S, F, P)$ a signature, $\sigma : SIG \rightarrow SIG'$ a signature morphism with $SIG \cap SIG' = \emptyset$, $\mathcal{A} \in Gen(SIG)$, $\mathcal{A}' \in Gen(SIG')$, and MP a mapping program for σ. Then holds:

(1) *$MP_{\mathcal{A}+\mathcal{A}'}$ is leftunique iff $\mathcal{A} + \mathcal{A}' \models VC_{\text{leftunique}}(MP, \sigma)$.*

(2) *$MP_{\mathcal{A}+\mathcal{A}'}$ is rightunique iff $\mathcal{A} + \mathcal{A}' \models VC_{\text{rightunique}}(MP, \sigma)$.*

(3) *$MP_{\mathcal{A}+\mathcal{A}'}$ is closed against F wrt $\mathcal{A}$ and $\mathcal{A}'|_\sigma$ iff $\mathcal{A} + \mathcal{A}' \models VC_{\text{closed against F}}(MP, \sigma)$.*

(4) *$MP_{\mathcal{A}+\mathcal{A}'} \supseteq GT_{\mathcal{A},\mathcal{A}'|_\sigma}$ iff $\mathcal{A} + \mathcal{A}' \models VC_{\text{supset of GT}}(MP, \sigma)$.*

(5) *$MP_{\mathcal{A}+\mathcal{A}'}$ is monotonic in P wrt $\mathcal{A}$ and $\mathcal{A}'|_\sigma$ iff $\mathcal{A} + \mathcal{A}' \models VC_{\text{monotonic in P}}(MP, \sigma)$.*

(6) *$MP^{-1}_{\mathcal{A}+\mathcal{A}'}$ is monotonic in P wrt $\mathcal{A}'|_\sigma$ and $\mathcal{A}$ iff $\mathcal{A} + \mathcal{A}' \models VC_{\text{inv monotonic in P}}(MP, \sigma)$.*

Proof. We make use of the fact that for all $s \in S$, $x_s \in X_s$, $y_s \in X_{\sigma(s)}$, and $v \in Val(X, \mathcal{A} + \mathcal{A}')$ holds

$$\mathcal{A} + \mathcal{A}', v \models MP_s(x_s, y_s) \quad \Leftrightarrow \quad (v(x_s), v(y_s)) \in MP(s)_{\mathcal{A}+\mathcal{A}'}$$

which can be proven as follows ($z_s \in X_{\sigma(s)} \setminus \{y_s\}$):

$$\begin{array}{ll} \multicolumn{2}{l}{\mathcal{A} + \mathcal{A}', v \models MP_s(x_s, y_s)} \\ \Leftrightarrow & \mathcal{A} + \mathcal{A}', v \models \langle MP(s)(x_s; z_s)\rangle\ z_s = y_s \\ \Leftrightarrow & \text{there is a } v' \in Val(X, \mathcal{A} + \mathcal{A}') \text{ with} \\ & v\ [\![MP(s)(x_s; z_s)]\!]_{\mathcal{A}+\mathcal{A}'}\ v' \text{ and } \mathcal{A} + \mathcal{A}', v' \models z_s = y_s \\ \Leftrightarrow & \text{there is a } v' \in Val(X, \mathcal{A} + \mathcal{A}') \text{ with} \\ & v\ [\![MP(s)(x_s; z_s)]\!]_{\mathcal{A}+\mathcal{A}'}\ v' \text{ and } v'(z_s) = v'(y_s) \\ \Leftrightarrow & \text{there is a } v' \in Val(X, \mathcal{A} + \mathcal{A}') \text{ with} \\ & v\ [\![MP(s)(x_s; z_s)]\!]_{\mathcal{A}+\mathcal{A}'}\ v' \text{ and } v'(z_s) = v(y_s) \\ \Leftrightarrow & (v(x_s), v(y_s)) \in MP(s)_{\mathcal{A}+\mathcal{A}'} \end{array}$$

With this in hand the proofs of (1) – (6) are straightforward. ■

Theorem 4.7 (monomorphicity obligations)
Let $SPEC = (SIG, X, Ax)$ a specification, $\sigma : SIG \rightarrow SIG'$ a bijective signature morphism with $SIG \cap SIG' = \emptyset$, and MP^u the uniform mapping program for σ. Then are equivalent:

(a) *$SPEC$ is monomorphic.*

(b) *there is a mapping program MP for σ such that:*

- $SPEC + \sigma(SPEC) \models VC_{\text{rightunique}}(MP, \sigma)$
- $SPEC + \sigma(SPEC) \models VC_{\text{closed against F}}(MP, \sigma)$
- $SPEC + \sigma(SPEC) \models VC_{\text{monotonic in P}}(MP, \sigma)$.

(c) *there is a mapping program MP for σ such that:*

- $SPEC + \sigma(SPEC) \models VC_{\text{rightunique}}(MP, \sigma)$
- $SPEC + \sigma(SPEC) \models VC_{\text{supset of GT}}(MP, \sigma)$
- $SPEC + \sigma(SPEC) \models VC_{\text{monotonic in P}}(MP, \sigma)$.

(d) *it holds:*

- $SPEC + \sigma(SPEC) \models VC_{\text{rightunique}}(MP^u, \sigma)$
- $SPEC + \sigma(SPEC) \models VC_{\text{monotonic in P}}(MP^u, \sigma)$.

(e) *it holds:*

- $SPEC + \sigma(SPEC) \models VC_{\text{leftunique}}(MP^u, \sigma)$
- $SPEC + \sigma(SPEC) \models VC_{\text{inv monotonic in P}}(MP^u, \sigma)$.

Proof. We start with a few preliminaries. Due to fact 2.34 and fact 2.37 is

$$SEM(\sigma(SPEC)) = (SEM(\sigma(SPEC))|_{\sigma})|_{\sigma^{-1}} = SEM(SPEC)|_{\sigma^{-1}}$$

and so, using fact 2.39, we get:

$$\begin{aligned} & SEM(SPEC + \sigma(SPEC)) \\ = \; & SEM(SPEC) + SEM(\sigma(SPEC)) \qquad (*) \\ = \; & SEM(SPEC) + SEM(SPEC)|_{\sigma^{-1}}. \end{aligned}$$

In the following we show that the single items of theorem 4.7 are equivalent to the single items in theorem 3.9:

$$\begin{tikzcd}
\text{theorem 4.7} & (a) \arrow[d, Leftrightarrow] & (b) \arrow[d, Rightarrow] & (c) \arrow[l, Rightarrow] & (d) \arrow[l, Rightarrow] & (e) \arrow[d, Leftrightarrow] \\
\text{theorem 3.9} & (a) \arrow[r, Leftrightarrow] & (b) \arrow[r, Leftrightarrow] & (c) \arrow[r, Leftrightarrow] & (d) \arrow[u, Rightarrow] \arrow[r, Leftrightarrow] & (e)
\end{tikzcd}$$

4.7(a) $\Leftrightarrow$ 3.9(a): trivial.

4.7(b) $\Rightarrow$ 3.9(b): Let $\mathcal{A}$ and $\mathcal{B}$ any two algebras from $SEM(SPEC)$, and set $\mathcal{A}' := \mathcal{B}|_{\sigma^{-1}}$. Then is (see (*)) $\mathcal{A} + \mathcal{A}' \in SEM(SPEC + \sigma(SPEC))$. From assumption, together with lemma 4.6, follows that $MP_{\mathcal{A}+\mathcal{A}'}$ is rightunique, closed against F wrt $\mathcal{A}$ and $\mathcal{A}'|_{\sigma}$, and monotonic in P wrt $\mathcal{A}$ and $\mathcal{A}'|_{\sigma}$. By choosing $R := MP_{\mathcal{A}+\mathcal{A}'}$ we get a rightunique family $R = (R_s)_{s \in S}$ of relations $R_s \subseteq A_s \times B_s$, which is closed against F wrt $\mathcal{A}$ and $\mathcal{B}$ and monotonic in P wrt $\mathcal{A}$ and $\mathcal{B}$. (The identity $\mathcal{A}'|_{\sigma} = \mathcal{B}$ is due to fact 2.34.)

4.7(c) $\Rightarrow$ 4.7(b): From assumption, together with lemma 4.6, follows for all $\mathcal{A} \in SEM(SPEC)$, $\mathcal{A}' \in SEM(\sigma(SPEC))$ that $MP_{\mathcal{A}+\mathcal{A}'}$ is rightunique and $MP_{\mathcal{A}+\mathcal{A}'} \supseteq GT_{\mathcal{A},\mathcal{A}'|_{\sigma}}$. So (fact 3.2(2)) $MP_{\mathcal{A}+\mathcal{A}'}$ is a total function, and thus, due to lemma 3.4 closed against F wrt $\mathcal{A}$ and $\mathcal{A}'|_{\sigma}$. Applying lemma 4.6 again, leads to $\mathcal{A} + \mathcal{A}' \models VC_{\text{closed against F}}(MP, \sigma)$. Because of (*) we get $SPEC + \sigma(SPEC) \models VC_{\text{closed against F}}(MP, \sigma)$.

4.7(d) $\Rightarrow$ 4.7(c): We choose as mapping program $MP := MP^u$. Then, the proof obligations in $VC_{\text{supset of GT}}(MP, \sigma)$ degenerate to tautologies. The other conditions of 4.7(c) are just parts of the assumption.

3.9(d) $\Rightarrow$ 4.7(d): Let us assume that 3.9(d) holds. Together with the above preliminaries follows that for any two algebras $\mathcal{A} \in SEM(SPEC)$, $\mathcal{A}' \in SEM(\sigma(SPEC))$ is $GT_{\mathcal{A},\mathcal{A}'|_\sigma}$ rightunique and monotonic in P wrt $\mathcal{A}$ and $\mathcal{A}'|_\sigma$. From theorem 4.4 and lemma 4.6 we get that $\mathcal{A} + \mathcal{A}' \models VC_{\text{rightunique}}(MP^u, \sigma)$ and $\mathcal{A} + \mathcal{A}' \models VC_{\text{monotonic in P}}(MP^u, \sigma)$. By (*) this is sufficient for $SPEC + \sigma(SPEC) \models VC_{\text{rightunique}}(MP^u, \sigma)$ and $SPEC + \sigma(SPEC) \models VC_{\text{monotonic in P}}(MP^u, \sigma)$.

4.7(e) $\Leftrightarrow$ 3.9(e): The implication 3.9(e) $\Rightarrow$ 4.7(e) can be proven just as 3.9(d) $\Rightarrow$ 4.7(d). On the other hand, from 4.7(e) follows (by (*)) that for any two algebras $\mathcal{A}, \mathcal{B} \in SEM(SPEC)$ holds (we abbreviate $\mathcal{B}|_{\sigma^{-1}}$ by $\mathcal{A}'$) $\mathcal{A} + \mathcal{A}' \models VC_{\text{leftunique}}(MP^u, \sigma) \cup VC_{\text{inv monotonic in P}}(MP^u, \sigma)$. Due to lemma 4.6 we get that $MP^u_{\mathcal{A}+\mathcal{A}'}$ is leftunique and that ${MP^u}^{-1}_{\mathcal{A}+\mathcal{A}'}$ is monotonic in P wrt $\mathcal{A}'|_\sigma$ and $\mathcal{A}$. Since $MP^u_{\mathcal{A}+\mathcal{A}'} = GT_{\mathcal{A},\mathcal{A}'|_\sigma}$ (theorem 4.4), ${MP^u}^{-1}_{\mathcal{A}+\mathcal{A}'} = GT_{\mathcal{A}'|_\sigma,\mathcal{A}}$ (fact 3.2(1)), and $\mathcal{A}'|_\sigma = \mathcal{B}$ (fact 2.34) this is exactly what 3.9(e) states.

This finishes the proof (cf. diagram above). ■

The criteria (b) – (e) of the above theorem suggest four approaches to the verification of monomorphicity of a specification. All the proof obligations are accessible to deduction, especially they can be dealt with in the KIV system. It remains the question which of the criteria (b) – (e) is the best with respect to tractability. To apply criterion (b) or (c) the existence of a mapping program has to be proved. This is done constructively by explicitly providing a program, checking that it is a mapping program (which can be done automatically) and proving the obligations. In (d) and (e) no program has to be made up, and the proof obligations can be generated completely automatically. So one might prefer the latter two criteria. However, on the other hand, providing a program in (b) or (c) adds further information to the proof task, which may be valuable while doing the proof. In some sense the same information must be "generated" during a proof of criterion (d) (provided the program is chosen appropriately). We believe that programming is easier than proving.[20] Thus, for reasons of tractability, we give preference to (b) and (c) (and not to (d) or (e)). Fortunately, in doing so, we do not lose anything, since we get (d) as a special case of (b) or (c) by choosing $MP := MP^u$. (We suspect that (d) and (e) are of equal tractability since they are in a sense symmetric.)

Comparing (b) and (c), the difference in $VC_{\text{closed against F}}(MP, \sigma)$ and $VC_{\text{supset of GT}}(MP, \sigma)$ can be seen in their granularity. In the former there

[20]This argument is also a criticism of some approaches to program synthesis. We believe that it is (in general) more tractable to explicitly provide a program and then verify it, instead of (implicitly) constructing the program while doing the proof.

is a formula for every function symbol of the signature, in the latter there is a formula for every sort symbol of the signature, where in some sense the formula for sort s captures all the formulas for function symbols with target sort s. On the first sight, following a divide and conquer principle, one might prefer (b) over (c). Unfortunately, in general some of the proof obligations in $VC_{\text{closed against F}}(MP, \sigma)$ mutually depend on each other, i.e. they have to be proven simultaneously (by induction). In this case the greater granularity in (b) is more of a hindrance than a help. However, if there are sorts mutually depending on each other, even some of the proof obligations in $VC_{\text{supset of GT}}(MP, \sigma)$ may have to be proven simultaneously. Thus it seems to be the best to syntactically analyze the sort dependencies in order to generate a set proof obligations without any mutually dependencies, but with optimal granularity. Essentially this will be a mixture of (conjunctions of) formulas from $VC_{\text{supset of GT}}(MP, \sigma)$ and $VC_{\text{closed against F}}(MP, \sigma)$. In addition, if there is information provided about which function symbols are constructors or not, it can be exploited to further optimize the definition of the uniform mapping program and the proof obligations.

5 Conclusion and Future Work

We have presented proof obligations, which are sufficient and necessary for the monomorphicity of a given specification. These proof obligations express certain properties of potential indeterministic procedures and are formulated in dynamic logic. So the task of proving monomorphicity can be directly dealt with in the KIV system [4] which was originally designed for program verification.

Currently, we investigate the question about the tractability of such proofs. We have reasons to hope that proving monomorphicity using the approach presented here, i.e. by (well-established) program verification methods, is much easier than using the meta-reasoning approach, which we have pursued formerly [5]. In the next future we will work on generalizing our results to parameterized specifications.

Acknowledgments

I am indebted to Thomas Fuchß, Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel for valuable discussions during the course of this research. Especially, the idea of copying signatures in order to reason about relations *between* algebras is due to Wolfgang Reif. Finally, I would like to thank Ralph Groß for carrying out some of the more technical proofs, and Thomas Fuchß for proof-reading an earlier draft of this paper.

References

[1] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1, Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, 1985.

[2] W. Reif. *Korrektheit von Spezifikationen und generischen Moduln*. PhD thesis, Universität Karlsruhe, Fakultät für Informatik, 1991.

[3] W. Reif. Correctness of full first-order specifications. In *4th Conference on Software Engineering and Knowledge Engineering*. Capri, Italy, IEEE Press, 1992.

[4] W. Reif. The KIV-system: Systematic construction of verified software. In *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*. Springer Verlag, 1992.

[5] W. Reif and A. Schönegge. Meta-level reasoning: Proving monomorphicity of specifications. In Deduktionstreffen 1994. Technical report. Technische Hochschule Darmstadt, Fachbereich Informatik, October 1994.

[6] A. Schönegge. Would you ever risk a non-monomorphic specification? Technical Report 33/95, Universität Karlsruhe, Fakultät für Informatik, 1995.

[7] M. Wirsing. *Algebraic Specification*, volume B of *Handbook of Theoretical Computer Science*, chapter 13, pages 675–788. Elsevier Science Publishers B. V., 1990.

A An Instructive Example

In the process of finding and proving the theorems in this paper we had formulated the following (faulty) lemma.

(Faulty) Lemma A.1
Let $SIG = (S, F, P)$, $\mathcal{A}, \mathcal{B} \in Gen(SIG)$, and $R = (R_s)_{s \in S}$ a family of relations $R_s \subseteq A_s \times B_s$. If R is lefttotal, righttotal and $R \subseteq GT_{\mathcal{A},\mathcal{B}}$, then is R closed against F wrt $\mathcal{A}$ and $\mathcal{B}$.

This faulty lemma says that every lefttotal and righttotal subset of $GT_{\mathcal{A},\mathcal{B}}$ is $GT_{\mathcal{A},\mathcal{B}}$ itself (cf. lemma 3.3). To see that this does *not* hold, look at a simple example specification:

sorts	s
functions	$a, b, c : s$
axioms	$a = b \wedge a \neq c \vee a \neq b \wedge a = c$

This specification has exactly two generated models (up to isomorphicity), namely one, which we denote by $\mathcal{A}$, with $a_{\mathcal{A}} = b_{\mathcal{A}}$ and $a_{\mathcal{A}} \neq c_{\mathcal{A}}$, and the other, which we denote by $\mathcal{B}$, with $a_{\mathcal{B}} \neq b_{\mathcal{B}}$ and $a_{\mathcal{B}} = c_{\mathcal{B}}$. The relation

$$R \ := \ \{(b_{\mathcal{A}}, b_{\mathcal{B}}), (c_{\mathcal{A}}, c_{\mathcal{B}})\}$$

is a lefttotal and righttotal (leftunique and rightunique) homomorphic relation between $\mathcal{A}$ and $\mathcal{B}$. However, since $(a_{\mathcal{A}}, a_{\mathcal{B}}) \notin R$, it is not closed against $\{a, b, c\}$ wrt $\mathcal{A}$ and $\mathcal{B}$.

Looking at the proof obligations, this result suggests that in general in the involved uniform mapping programs (cf. definition 4.2) the program fragments for the single function symbols cannot be combined in a fixed order with **if-then-else** constructs instead of nondeterministic choice ($\bigcup$) (which is the case for uniform restrictions for modules [2]). For example, the input-output relation (under $\mathcal{A} + \mathcal{A}'$) of

```
map($x$; var $y$). if $x = b$ then $y := b'$
                   else if $x = c$ then $y := c'$
                   else if $x = a$ then $y := a'$
                   else abort
```

(where $a' = \sigma(a)$, $b' = \sigma(b)$, $c' = \sigma(c)$) is only a proper subset of, but not equal to $GT_{\mathcal{A},\mathcal{A}'|_\sigma}$ (cf. theorem 4.4). If one would take this procedure as uniform mapping program, the proof obligations listed in theorem 4.7(d) could be shown (though the specification is not monomorphic).