Electronic Notes in Theoretical Computer Science 42 (2001)
URL: `http://www.elsevier.nl/locate/entcs/volume42.html` 23 pages

Computability and Decidability Issues in the Theory of Consistency Enforcement

Sebastian Link[1] and Klaus-Dieter Schewe[2]

Massey University,
Department of Information Systems,
Private Bag 11222
Palmerston North, New Zealand

Abstract

Consistency enforcement provides an alternative to common program verification within formal program specification languages. The existing approach uses a partial order on semantic equivalence classes of program specifications, called specialization, and aims to replace a given specification S by the greatest consistent specialization $S_{\mathcal{I}}$ which is provably consistent with respect to the given static invariant $\mathcal{I}$.

The underlying logic is arithmetic logic which allows computability and decidability issues in connection with the constructive generation of $S_{\mathcal{I}}$ to be investigated.

In this paper we justify the axiomatic approach to Dijkstra's calculus with respect to arithmetic logic and develop a new theory on top of that basis for which the construction problem of the greatest consistent specialization of a complex specification can be reduced to the involved basic commands and the investigation of a precondition. In addition, we are now even able to show that this construction is computable under mild restrictions concerning recursive program specifications and that the occuring precondition is decidable for common classes of invariants.

1 Introduction

It has become fairly standard within almost all formal specification languages to provide at least static invariants. On the one hand this allows to describe the semantics of a system in a sufficent way, but the other side of the coin is the task to guarantee consistency. For a program specification S and an invariant $\mathcal{I}$ this means that every execution of S starting in a state that satisfies $\mathcal{I}$ should always lead to a state satisfying $\mathcal{I}$, too. However, we can

[1] Email: `S.Link@massey.ac.nz`
[2] Email: `K.D.Schewe@massey.ac.nz`

restrict ourselves to terminating executions of S and can therefore handle the problems of termination and consistency separately.

For a general treatment we consider the axiomatic approach to program semantics using predicate transformers which lead to weakest (liberal) preconditions. It is well known in this context that we may express consistency by the proof obligation $\mathcal{I} \Rightarrow wlp(S)(\mathcal{I})$, but the verification efforts normally get out of hand.

The search for alternatives brings us to the idea of consistency enforcement. In particular, in the field of databases, where the complexity of the invariants—usually called integrity constraints in this context [9]—is much higher than the complexity of the programs themselves, the trigger approach has become very popular, but it can be shown that triggers cannot solve the problem in general [7].

A different approach, which indeed suggests itself, deals with *greatest consistent specializations* (GCSs) [6,8]. Considering a given program specification S and a given static invariant $\mathcal{I}$ we try to replace S by a slightly modified program specification $S_{\mathcal{I}}$ that is provably consistent with respect to $\mathcal{I}$. Furthermore, all "effects" of the original S should be preserved within $S_{\mathcal{I}}$. This can be formalized by the introduction of a specialization order on semantic equivalence classes of program specifications. The existing theory on GCS in [6] is based on infinitary logic $\mathcal{L}^{\omega}_{\omega\infty}$.

In order to shift the GCS approach from a somehow purely theoretical framework to an applicable theory we have to investigate computability of GCSs and decidability of preconditions that must be built. For these purposes it is preferable to obtain a tight connection with classical recursion theory [1]. This will be done in this paper.

At the beginning of Section 2 we introduce our notations for arithmetic logic. Then we are up to define what predicate transformers should be with respect to this logic, especially we want this axiomatical approach to become equivalent with the relational semantics. Therefore, we have to guarantee some slightly modified characteristic properties of predicate transformers, namely the universal conjunctivity and the pairing condition. Being so far, the standard semantics of guarded commands carries over and we are even able to show in Section 3 that recursion theory can be extended to the arithmetic case, at least, if we are restricted to certain WHILE-loops.

With this background we can develop the GCS approach on top of arithmetic logic. This will be done in Section 4. Many of the proofs follow the ideas of their classical counterparts in [6]. Computability cannot be achieved in general, since the building of least fixpoints requires to test for semantic equivalence, which is undecidable. If restricted to FOR-loops, however, GCSs become computable. This will be shown in Section 5. Furthermore, we show that preconditions that naturally occur in GCSs are not decidable in general, but we also indicate conditions which guarantee decidability.

We argue that at least for one application field, i.e. databases as already mentioned, the restrictions are tolerable. For the general case some other pragmatic solutions must be applied [5]. We conclude with a short summary and outlook.

For more details we refer the reader to the thesis [3].

2 Programming Semantics based on Arithmetic Logic

Our study is based on first-order arithmetic logic [1, Ch.7], i.e. our logical language contains just the function symbols 0, s, $+$ and $*$ of arity 0, 1, 2 and 2. The informal meaning is as usual: the constant 0, the succesor function, addition and multiplication. By convenience $+$ and $*$ are written as infix operators. The only predicate symbol is the equality symbol $=$. Variables in our language will be $x_1, x_2, x_3, \dots$.

We use the notation $\mathbb{T}$ for the set of terms and $\mathbb{F}$ for the set of formulae. In addition, let V denote the set of variables. We allow all standard abbreviations including formulae *true* and *false.*

Semantically, we fix a structure with domain $\mathbb{N}$, the set of non-negative integers. Then 0, s, $+$, $*$ and $=$ are interpreted in the usual way. For an interpretation it is then sufficient to consider a function $\sigma : V \to \mathbb{N}$. By the coincidence theorem it is even sufficient to be given the values $\sigma(x_i)$ for the free variables x_i in a term or a formula. In particular, we may always write σ as a k-tuple, if the number of free variables is k.

Finally, a k-ary relation $R \subseteq \mathbb{N}^k$ is called *arithmetical* iff it can be represented by a formula $Q \in \mathbb{F}$ in arithmetic logic (with free variables $x_1, \dots, x_k$), i.e. $(a_1, \dots, a_k) \in R$ holds iff $\models_\sigma Q$ holds for the interpretation defined by $\sigma(x_i) = a_i$ $(i = 1, \dots, k)$.

2.1 Axiomatic approach and its justification

In accordance with the existing theory on consistency enforcement in [6] each finite subset $X \subseteq V$ is called a *state space*. Each function $\sigma : X \to \mathbb{N}$ is called a *state* on X. Equivalently, a state is always representable by a k-tuple. For a fixed X let Σ $(= \Sigma(X))$ denote the *set of all states over* X.

A formula $\varphi \in \mathbb{F}$ with free variables $fr(\varphi)$ in X is then called an *X-formula* or an *invariant* on X. In order to emphasize the variables we sometimes write $\varphi(\boldsymbol{x})$ with a vector $\boldsymbol{x}$ of the state variables involved.

Then any pair of formulae $(\Delta(S), \Sigma_0(S))$ with $2k$ and k free variables, respectively, may be considered as defining the *relational semantics* of a program specification S. For convenience assume the first k free variables in $\Delta(S)$ to coincide with the free variables of $\Sigma_0(S)$.

According to our notation we sometimes write $\Delta(S)(\boldsymbol{x}, \boldsymbol{y})$ and $\Sigma_0(S)(\boldsymbol{x})$. So $\Delta(S)$ can be interpreted by state pairs, whereas $\Sigma_0(S)$ allows an interpretation by states. We interpret (σ, τ) with $\models_{(\sigma,\tau)} \Delta(S)$ as an *execution* of S with start state σ and a final state τ. Similarly, a state σ satisfying $\Sigma_0(S)$ is considered as a start state for S, in which a non-terminating execution of S exists.

Note that the model of relational semantics comprises daemonic non-determinism, non-termination and partial undefinedness.

In order to come to an axiomatic semantics based on the introduced logic of arithmetic, we associate with S two *predicate transformers* $wlp(S)$ and $wp(S)$—i.e., functions from (equivalence classes) of formulae to (equivalence classes) of formulae—with the standard informal meaning:

- $wlp(S)(\varphi)$ characterizes those initial states σ such that each terminating execution of S starting in σ results in a state τ satisfying φ.
- $wp(S)(\varphi)$ characterizes those initial states σ such that each execution of S starting in σ terminates and results in a state τ satisfying φ.

The notation $wlp(S)(\varphi)$ and $wp(S)(\varphi)$ corresponds to the usual *weakest (liberal) precondition* of S with respect to the postcondition φ. In order to save space we shall often use the notation $w(l)p(S)(\varphi)$ to refer to both predicate transformers at a time. If this occurs in an equivalence, then omitting everything in parentheses gives the wp-part, whereas omitting just the parentheses results in the wlp-part.

From our introduction of $\Delta(S)$ and $\Sigma_0(S)$ the following definition becomes straightforward.

Definition 2.1 The *predicate transformers* associated with a program specification S on a state space X are defined as

$$\begin{aligned} wlp(S)(\varphi(\boldsymbol{x})) &\Leftrightarrow \forall \boldsymbol{y}.\Delta(S)(\boldsymbol{x},\boldsymbol{y}) \Rightarrow \varphi(\boldsymbol{y}) \qquad \text{and} \\ wp(S)(\varphi(\boldsymbol{x})) &\Leftrightarrow (\forall \boldsymbol{y}.\Delta(S)(\boldsymbol{x},\boldsymbol{y}) \Rightarrow \varphi(\boldsymbol{y})) \wedge \neg\Sigma_0(S)(\boldsymbol{x}) \end{aligned}$$

for arbitrary X-formulae φ. □

The next step is to show that predicate transformers satisfying some nice conditions are sufficient for the definition of program specifications S. The conditions are the *pairing condition* and a slightly modified *universal conjunctivity* property. This gives the equivalence between the relational and the predicate transformer semantics, the so called *inversion theorem*.

We use the standard notation $w(l)p(S)^*(\varphi) \Leftrightarrow \neg w(l)p(S)(\neg\varphi)$ and refer to $wlp(S)^*$ and $wp(S)^*$ as the *dual predicate transformers*.

Proposition 2.2 *The predicate transformers $w(l)p(S)$ satisfy the following conditions:*

$$\begin{aligned} & wp(S)(\varphi) \Leftrightarrow wlp(S)(\varphi) \wedge wp(S)(true) \qquad \textit{and} \\ & wlp(S)(\forall \boldsymbol{y}.Q(\boldsymbol{y}) \Rightarrow \varphi(\boldsymbol{x},\boldsymbol{y})) \Leftrightarrow \forall \boldsymbol{y}.Q(\boldsymbol{y}) \Rightarrow wlp(S)(\varphi(\boldsymbol{x},\boldsymbol{y})) \qquad . \end{aligned}$$

Conversely, any pair of predicate transformers satisfying these two conditions defines $\Delta(S)(\boldsymbol{x},\boldsymbol{y}) \Leftrightarrow wlp(S)^*(\boldsymbol{x}=\boldsymbol{y})$ and $\Sigma_0(\boldsymbol{x}) \Leftrightarrow wp(S)^*(false)$.

Proof. (sketch) It is straightforward to show that both conditions hold, i.e., it remains to show $w(l)p(S) = f_{(l)p}(S)$ for further predicate transformers $f_{(l)p}$. The main idea is use the equivalence of an arbitrary X-formula $\varphi(\boldsymbol{x})$ to $\forall \boldsymbol{y}.(\boldsymbol{x} = \boldsymbol{y} \Rightarrow \varphi(\boldsymbol{y}))$ and to apply the universal conjunctivity property of f_{lp} provided. For the wp-case we make use of pairing conditions as well as of the first equation already been proved. □

The next result gives a normal form representation of the predicate transformer $wlp(S)$, which will be useful in many proofs.

Lemma 2.3 *It is always possible to write $wlp(S)(\varphi)$ in the form*

$$wlp(S)(\varphi(\boldsymbol{x})) \Leftrightarrow \forall \boldsymbol{z}.wlp(S)^*(\boldsymbol{x} = \boldsymbol{z}) \Rightarrow \varphi(\boldsymbol{z}) \quad .$$

Proof. Obviously, we have $\varphi(\boldsymbol{x}) \Leftrightarrow \forall \boldsymbol{z}.\boldsymbol{x} = \boldsymbol{z} \Rightarrow \varphi(\boldsymbol{z}) \Leftrightarrow \forall \boldsymbol{z}.\neg\varphi(\boldsymbol{z}) \Rightarrow \boldsymbol{x} \neq \boldsymbol{z}$. Then the lemma follows immediately by applying the universal conjunctivity property. □

2.2 Guarded Commands afreshed

We now introduce the familiar language of guarded commands [4], for which we use standard basic commands and standard constructors. To define the semantics we simply have to define the predicate transformers since the inversion theorem is now available. These definitions are given as follows:

$$\begin{aligned}
w(l)p(skip)(\varphi) &\Leftrightarrow \varphi \\
w(l)p(fail)(\varphi) &\Leftrightarrow true \\
w(l)p(loop)(\varphi) &\Leftrightarrow false(\vee true) \\
w(l)p(x_{i_1} := t_{i_1} \| \dots \| x_{i_k} := t_{i_k})(\varphi) &\Leftrightarrow \{x_{i_1}/t_{i_1}, \dots, x_{i_k}/t_{i_k}\}.\varphi \\
w(l)p(S_1; S_2)(\varphi) &\Leftrightarrow w(l)p(S_1)(w(l)p(S_2)(\varphi)) \\
w(l)p(S_1 \Box S_2)(\varphi) &\Leftrightarrow w(l)p(S_1)(\varphi) \wedge w(l)p(S_2)(\varphi) \\
w(l)p(S_1 \boxtimes S_2)(\varphi) &\Leftrightarrow w(l)p(S_1)(\varphi) \wedge (wp(S_1)^*(true) \vee w(l)p(S_2)(\varphi)) \\
w(l)p(@x_j \bullet S)(\varphi) &\Leftrightarrow \forall x_j.w(l)p(S)(\varphi) \\
w(l)p(\mathcal{P} \to S)(\varphi) &\Leftrightarrow \mathcal{P} \Rightarrow w(l)p(S)(\varphi)
\end{aligned}$$

Here $\{x_{i_1}/t_{i_1}, \dots, x_{i_k}/t_{i_k}\}$ denotes the simultaneous substitution of the variables x_{i_j} by the terms t_{i_j}. It is easy to verify the pairing condition and the universal conjunctivity property for these predicate transformers.

We say that S is an X*-command* for some state space X iff $w(l)p(S)(\varphi) \Leftrightarrow \varphi$ hold for each Y-formulae φ, where $X \cap Y = \emptyset$, and X is minimal with this property.

3 Recursive Guarded Commands

In the last section we introduced the language of guarded commands together with an axiomatic semantics expressed via predicate transformers in arithmetic logic. So far, this language covers straightline non-deterministic partial programs extended by unbounded choice. We would like to go a bit further and investigate recursive programs expressed as least fixpoints $\mu T.f(T)$ with respect to a suitable order $\preceq$. This order will be the standard Nelson-order [4].

Unfortunately, we are not able to carry over the very general recursion theory from [4]. We have to restrict ourselves to simple WHILE-loops, i.e., $f(T) = \mathcal{P} \to S; T \Box \neg\mathcal{P} \to \mathit{skip}$, where the variable T does not occur within S. For convenience, we introduce command variables $T_1, T_2, \dots$.

3.1 The Nelson-Order

The idea of the Nelson-order is that whenever $S_1 \preceq S_2$ holds, then each terminating execution of S_1 is preserved within S_2, but a terminating execution in S_2 may be "approximated" in S_1 by a non-terminating execution. This leads to the following definition.

Definition 3.1 The *Nelson-order* is defined by

$$S_1 \preceq S_2 \Leftrightarrow (\mathit{wlp}(S_2)(\varphi) \Rightarrow \mathit{wlp}(S_1)(\varphi)) \wedge (\mathit{wp}(S_1)(\varphi) \Rightarrow \mathit{wp}(S_2)(\varphi))$$

for all φ. □

We are specially interested in the chain $\{f^i(\mathit{loop})\}_{i\in\mathbb{N}}$ with respect to $\preceq$. Therefore, we take a Gödel numbering g of guarded commands, which extends a Gödel numbering h of terms and formulae for our logic ([1, p.327f.]). With this Gödel numbering g we may express all formulae $w(l)p(f^i(\mathit{loop}))(\varphi)$ by two arithmetic predicates Q_1 and Q_2.

Lemma 3.2 *There are $k+2$-ary arithmetic predicates Q_1 and Q_2 such that*

$$Q_1(i,j,\boldsymbol{x}) \Leftrightarrow \mathit{wlp}(f^i(\mathit{loop}))(\varphi(\boldsymbol{x})) \quad \textit{and} \quad Q_2(i,j,\boldsymbol{x}) \Leftrightarrow \mathit{wp}(f^i(\mathit{loop}))(\varphi(\boldsymbol{x}))$$

hold for $h(\varphi(\boldsymbol{x})) = j$ and $\boldsymbol{x} = x_{i_1}, \dots, x_{i_k}$.

Proof. (sketch) For arbitrary program specifications S and arithmetic predicates φ, it is straightforward to define inductively formulae Q'_1 such that $Q'_1(i,j,\boldsymbol{x}) \Leftrightarrow \mathit{wlp}(S)(\varphi(\boldsymbol{x}))$ with $g(S) = i$ and $h(\varphi) = j$ holds. In particular, using our Gödel numbering g we obtain a primitve recursive function $\bar{g}$ such that

$$Q'_1(\bar{g}^k(g(\mathit{loop})), j, \boldsymbol{x}) \Leftrightarrow \mathit{wlp}(f^k(\mathit{loop}))(\varphi(\boldsymbol{x}))$$

is satisfied. Then, we define predicates $Q_1(k,j,\boldsymbol{x}) \Leftrightarrow Q'_1(\bar{g}^k(g(\mathit{loop})), j, \boldsymbol{x})$ and an extension of $\bar{Q}(h(\psi), h(\varphi), \boldsymbol{x}) \Leftrightarrow (\mathcal{P} \Rightarrow \mathit{wlp}(T)(\psi) \wedge \neg\mathcal{P} \Rightarrow \varphi)$ with $\psi, \varphi \in \mathbb{F}$. We conclude

$$Q_1(k,j,\boldsymbol{x}) \Leftrightarrow Q'_1(\bar{g}^k(g(\mathit{loop})), j, \boldsymbol{x}) \Leftrightarrow \bar{Q}(h(\psi), h(\varphi), \boldsymbol{x}) \quad ,$$

where $h(\varphi) = j$ and $\psi(\boldsymbol{x}) = wlp(f^{k-1}(loop))(\varphi(\boldsymbol{x})) = Q_1(k-1, j, \boldsymbol{x})$. In summary, we receive

$$Q_1(0, j, \boldsymbol{x}) \Leftrightarrow true \quad \text{and}$$
$$Q_1(k+1, j, \boldsymbol{x}) \Leftrightarrow \bar{Q}(h(Q_1(k, j, \boldsymbol{x})), j, \boldsymbol{x}).$$

Since $\bar{Q}$ is arithmetic, so Q_1 is as recursive predicates are closed under primitve recursion. The statement for Q_2 is completely analogous. □

We now define a *limit operator* $S = \lim_{i\in\mathbb{N}} f^i(loop)$ with help of the predicates Q_1 and Q_2:

$$\begin{aligned} wlp(S)(\varphi(\boldsymbol{x})) &\Leftrightarrow \forall i.Q_1(i, h(\varphi(\boldsymbol{x})), \boldsymbol{x}) \qquad \text{and} \\ wp(S)(\varphi(\boldsymbol{x})) &\Leftrightarrow \exists i.Q_2(i, h(\varphi(\boldsymbol{x})), \boldsymbol{x}) \quad . \end{aligned}$$

Lemma 3.3 *The limit $S = \lim_{i\in\mathbb{N}} f^i(loop)$ is well-defined.*

Proof. (sketch) By definition, it is obvious that

$$wlp(S)(\varphi(\boldsymbol{x})) \quad \Leftrightarrow \quad \forall i.i \in \mathbb{N} \Rightarrow wlp(f^i(loop))(\varphi(\boldsymbol{x})) \tag{1}$$

holds. From this we derive the universal conjunctivity property by direct calculation. Furthermore, the *wp*-part of the limes definition gives

$$wp(S)(\varphi(\boldsymbol{x})) \quad \Leftrightarrow \quad \exists i.i \in \mathbb{N} \wedge wp(f^i(loop))(\varphi(\boldsymbol{x})) \quad . \tag{2}$$

Under the assumption that $wp(S)(\varphi)$ holds we use equivalence (2), the pairing conditions of the $f^i(loop)$'s and the fact that $\{f^i(loop) \mid i \in \mathbb{N}\}$ is indeed a chain with respect to the Nelson-order in order to verify that $wlp(S)(\varphi) \wedge wp(S)(true)$ holds. For the reverse direction one can use the pairing condition of some $f^{i_0}(loop)$ and both equivalences (1) and (2). □

3.2 Least Fixpoints

Now, we are going to show how to obtain the semantics for WHILE-loops. It is easy to see that the function f on guarded commands is monotonic in the Nelson order [4]. Then an immediate consequence of the last lemma is the existence of a least upper bound, which is just given by the limit operator. To see this one only has to apply equations (1) and (2) as well as the idea of a least upper bound.

Lemma 3.4 *The chain $\{f^i(loop) \mid i \in \mathbb{N}\}$ has a least upper bound, namely $\lim_{i\in\mathbb{N}} f^i(loop)$.* □

In the following we use the notation $\mu T_j.f(T_j)$ to denote the least fixpoint of f provided it exists.

Proposition 3.5 *Let $f(T_j) = \mathcal{P} \rightarrow S; T_j \Box \neg\mathcal{P} \rightarrow skip$. Then f has a least fixpoint with respect to $\preceq$, which is $\mu T_j.f(T_j) = \lim_{i\in\mathbb{N}} f^i(loop)$.*

Proof. (sketch) According to Lemma 3.4, $\{f^i(loop) \mid i \in \mathbb{N}\}$ has the least upper bound $S = \lim_{i\in\mathbb{N}} f^i(loop)$. Then we can make use of the universal conjunctivity property and apply equation (2) and the fact that $S \preceq f(S)$ holds to derive that $w(l)p(f(S))(\varphi) \Leftrightarrow w(l)p(S)(\varphi)$ are valid. Hence, S is indeed a fixpoint and each further fixpoint T is an upper bound for $\{f^i(loop) \mid i \in \mathbb{N}\}$, which gives $S \preceq T$. □

4 Greatest Consistent Specializations

Now the foundations are laid to develop the theory of consistency enforcement on top of first-order arithmetic logic. This will then form the basis for investigations concerning computability and decidability.

4.1 Consistency and Specialization

First we have to define consistency and the specialization preorder. This can be done in complete analogy to the classical case in [6].

Definition 4.1 Let $\mathcal{I}$ be an invariant on the state space X. Let S and T be commands on the state spaces Z and Y, respectively, with $Z \subseteq Y \subseteq X$.

- S is *consistent* with respect to $\mathcal{I}$ iff $\mathcal{I} \Rightarrow wlp(S)(\mathcal{I})$ holds.
- T *specializes* S (notation: $T \sqsubseteq S$) iff $w(l)p(S)(\varphi) \Rightarrow w(l)p(T)(\varphi)$ holds for all Z-formulae φ. □

Due to the pairing condition it is sufficient to consider only $\varphi = true$ for the *wp*-part in the specialization definition. The *wlp*-part can also be simplified in the known way.

Next we introduce the central notion for consistency enforcement, the GCS.

Definition 4.2 Let S be a Y-command and $\mathcal{I}$ an invariant on X with $Y \subseteq X$. The *greatest consistent specialization* (GCS) of S with respect to $\mathcal{I}$ is an X-command $S_{\mathcal{I}}$ with $S_{\mathcal{I}} \sqsubseteq S$, such that $S_{\mathcal{I}}$ is consistent with respect to $\mathcal{I}$ and each consistent specialization $T \sqsubseteq S$ satisfies $T \sqsubseteq S_{\mathcal{I}}$. □

First we show the existence of GCSs and their uniqueness up to semantic equivalence. Furthermore, GCSs with respect to conjunctions can be built successively and independently from the order of the given invariants. In both cases, the classical proofs from [8,6] carry over without significant changes. Nevertheless, we will give the proofs in Appendix A.

Proposition 4.3 *The GCS $S_{\mathcal{I}}$ of S with respect to $\mathcal{I}$ always exists and is unique up to semantic equivalence. We can always write*

$$S_{\mathcal{I}} = (\mathcal{I} \rightarrow S; @\boldsymbol{z}' \bullet \boldsymbol{z} := \boldsymbol{z}'; \mathcal{I} \rightarrow skip) \boxtimes (\neg\mathcal{I} \rightarrow S; @\boldsymbol{z}' \bullet \boldsymbol{z} := \boldsymbol{z}') \ ,$$

where $\boldsymbol{z}$ refers to the free variables in $\mathcal{I}$ not occurring in S.

Furthermore, for two invariants $\mathcal{I}$ and $\mathcal{J}$ we always obtain that $\mathcal{I} \wedge \mathcal{J} \rightarrow S_{\mathcal{I} \wedge \mathcal{J}}$ and $\mathcal{I} \wedge \mathcal{J} \rightarrow (S_{\mathcal{I}})_{\mathcal{J}}$ are semantically equivalent. □

From the form of the GCS Proposition 4.3 we may derive $wp(S_{\mathcal{I}})(true) \Leftrightarrow wp(S)(true)$, which allows to concentrate on the predicate transformer $wlp(S)$.

4.2 An Upper Bound for GCSs

For practical applications the form of the GCS derived in Proposition 4.3 is almost worth nothing, since it involves testing the invariant after non-deterministic selection of arbitrary values. However, the form is useful in proofs.

A suitable form of the GCS should be built from GCSs of the basic commands involved in S. Let the result of such a naive syntactic replacement be denoted by $S'_{\mathcal{I}}$. In general, however, $S'_{\mathcal{I}}$ is not the GCS. It may not even be a specialization of S, or it may be a consistent specialization, but not the greatest one. An example for the latter case is $S = x := x - a; x := x + a$ with some constant $a \geq 0$ and $\mathcal{I} \equiv x \geq 0$.

We now formulate a technical condition which allows to exclude this situation. Under this condition it will be possible to show that $S_{\mathcal{I}} \sqsubseteq S'_{\mathcal{I}}$ holds. The corresponding result will be called the *upper bound theorem*.

We need the notion of a *deterministic branch* S^+ of a command S, which requires $S^+ \sqsubseteq S$ and $wp(S)^*(true) \Leftrightarrow wp(S^+)^*(true)$ as well as $wlp(S^+)^*(\varphi) \Rightarrow wp(S^+)(\varphi)$ to hold for all φ.

Furthermore, we need the notion of a *δ-constraint* for an X-command S. This is an invariant $\mathcal{J}$ on $X \cup X'$ with a disjoint copy X' of X, for which $\{\boldsymbol{x}'/\boldsymbol{x}\}.wlp(S')(\mathcal{J})$ holds, where S' results from S by renaming all x_i to x'_i.

Finally, we write $\mathcal{P}_\sigma$ for the characterizing formula of state σ.

Definition 4.4 Let $S = S_1; S_2$ be an Y-command such that S_i is a Y_i-command for $Y_i \subseteq Y$ $(i = 1, 2)$. Let $\mathcal{I}$ be some X-invariant with $Y \subseteq X$. Let $X - Y_1 = \{y_1, \dots, y_m\}$, $Y_1 = \{x_1, \dots, x_l\}$ and assume that $\{x'_1, \dots, x'_l\}$ is a disjoint copy of Y_1 disjoint also from X. Then S is in *δ-$\mathcal{I}$-reduced form* iff for each deterministic branch S_1^+ of S_1 the following two conditions—with $\boldsymbol{x} = (x_1, \dots, x_l)$, $\boldsymbol{x'} = (x'_1, \dots, x'_l)$—hold:

- For all states σ with $\models_\sigma \neg\mathcal{I}$ we have, if $\mathcal{P}_\sigma \Rightarrow \{\boldsymbol{x}/\boldsymbol{x'}\}.(\forall y_1 \dots y_m.\mathcal{I})$ is a δ-constraint for S_1^+, then it is also a δ-constraint for $S_1^+ ; S_2$.
- For all states σ with $\models_\sigma \mathcal{I}$ we have, if $\mathcal{P}_\sigma \Rightarrow \{\boldsymbol{x}/\boldsymbol{x'}\}.(\forall y_1 \dots y_m.\neg\mathcal{I})$ is a δ-constraint for S_1^+, then it is also a δ-constraint for $S_1^+ ; S_2$. □

We have to extend this definition to arbitrary commands other than sequences.

Definition 4.5 Let S be an X-command and $\mathcal{I}$ some Y-invariant with $X \subseteq Y$. S is called *$\mathcal{I}$-reduced* iff the following holds:

- If S is one of $fail$, $skip$, $loop$ or an assignment, then S is always $\mathcal{I}$-reduced.
- If $S = S_1; S_2$, then S is $\mathcal{I}$-reduced iff S_1 and S_2 are $\mathcal{I}$-reduced and S is

δ-$\mathcal{I}$-reduced.

- If S is one of $\mathcal{P} \rightarrow T$, $@\, y :: \# y \bullet T$, $S_1 \Box S_2$ or $S_1 \boxtimes S_2$, then S is $\mathcal{I}$-reduced iff S_1 and S_2 or T respectively are $\mathcal{I}$-reduced.
- If $S = \mu T.f(T)$, then S is $\mathcal{I}$-reduced iff $f^n(loop)$ is $\mathcal{I}$-reduced for each $n \in \mathbb{N}$. $\Box$

With these technical preliminaries we may now state and prove the upper bound theorem. Notice, that a simple replacement of some $S_1 \boxtimes S_2$ by $(S_1)_{\mathcal{I}} \boxtimes (S_2)_{\mathcal{I}}$ within a recursive operation would destroy the required result.

Theorem 4.6 *Let $\mathcal{I}$ be an invariant on X and let S be some $\mathcal{I}$-reduced Y-command with $Y \subseteq X$. Let $S'_{\mathcal{I}}$ result from S as follows:*

- *Each restricted choice $S_1 \boxtimes S_2$ occurring within S will be replaced by $S_1 \Box\, wlp(S_1)(false) \rightarrow S_2$.*
- *Then each basic command will be replaced by their GCSs with respect to $\mathcal{I}$.*

Then $T \sqsubseteq S'_{\mathcal{I}}$ holds for each consistent specialization $T \sqsubseteq S$ with respect to $\mathcal{I}$.

Proof. (sketch) The proof is done by (lengthy) structural induction on S. There are no great ideas hidden behind the cases for preconditioning, choice, restricted choice and unbounded choice, but the proofs require a huge technical effort, nevertheless. Therefore, we refer to the classical proofs [6, p.120-122] which only have to be changed slightly.

For sequences we can also keep the classical proof [6, App.C] with minor changes. The most important idea is to exploit the normal form of the GCS from Proposition 4.3 for $S = S_1; S_2$ to compute both $wlp(S_{\mathcal{I}})(\boldsymbol{x} = \boldsymbol{a})$ and $wlp((S_1)_{\mathcal{I}}; (S_2)_{\mathcal{I}})(\boldsymbol{x} = \boldsymbol{a})$. Then we distinguish the two cases $\boldsymbol{x} = \boldsymbol{a} \Rightarrow \neg\mathcal{I}$ and $\boldsymbol{x} = \boldsymbol{a} \Rightarrow \mathcal{I}$. We may assume without loss of generality that S_1 is deterministic such that we are able to use an advanced technical characterization of δ-$\mathcal{I}$-reduceness.

In the first case we assume that the required implication for specialization is violated in at least one state $\boldsymbol{x} \mapsto \boldsymbol{b}$. From this we obtain a δ-constraint for S_1 and by δ-$\mathcal{I}$-reducedness a δ-constraint for S in the form $\boldsymbol{x} = \boldsymbol{b} \Rightarrow \{\boldsymbol{x}/\boldsymbol{x}'\}.\mathcal{I}$. This leads to a contradiction. The second case is quite similar.

For recursive guarded commands—restricted to WHILE-loops, for which we know the existence of least fixpoints—we shift the proof to Appendix B. $\Box$

4.3 The general form of a GCS

Theorem 4.6 has a flavour of compositionality, but it does not yet give the GCS. The idea of the main theorem on GCSs is to cut out from $S'_{\mathcal{I}}$ those executions that are not allowed to occur in a specialization of S. This leads to the following theorem. The proof is shifted to Appendix C.

Theorem 4.7 *Let $\mathcal{I}$, S and $S'_{\mathcal{I}}$ be as in Theorem 4.6. Let Z be a disjoint* *copy of the state space Y. With the formulae*

$$\mathcal{P}(S,\mathcal{I},\boldsymbol{x}') \equiv \{\boldsymbol{z}/\boldsymbol{y}\}.wlp(S''_{\mathcal{I}};\boldsymbol{z}=\boldsymbol{x}' \rightarrow skip)(wlp(S)^*(\boldsymbol{z}=\boldsymbol{y})) \quad ,$$

where $S''_{\mathcal{I}}$ results from $S'_{\mathcal{I}}$ by renaming the Y to Z, the GCS $S_{\mathcal{I}}$ is semantically equivalent to

$$@\boldsymbol{x}' \bullet \mathcal{P}(S,\mathcal{I},\boldsymbol{x}') \rightarrow S'_{\mathcal{I}};\boldsymbol{y}=\boldsymbol{x}' \rightarrow skip \quad .\Box$$

Note that if we consider deterministic branches as a pragmatic approach suggested in [6], then the unbounded choice in Theorem 4.7 disappears. We omit further details.

The charaterization of GCSs according to Theorem 4.7 makes it formally possible to reduce consistency enforcement to a simple syntactical replacement (the forming of $S'_{\mathcal{I}}$) and to an investigation of a guard, namely $P(S,\mathcal{I},\boldsymbol{x}')$. The following results from section 5 will heavily depend on this reduction and will support the theorys practical relevance.

5 Computability and Decidability

We have now reached the stage, where we can say that the GCS approach could have been succesfully developed with respect to arithmetic logic. Thus, we can turn to the original intention of this paper: computability and decidability issues.

Taking the general form of the GCS in Theorem 4.7 we may now ask, whether we can find an algorithm to compute the GCS. We may further ask, whether the result is effective. The answer to both questions is negative in general, but we will identify subcases, for which effective GCSs can be computed.

5.1 The Computability of GCSs

First consider the computability problem. Taking our Gödel numberings h for terms and formulae and g for commands, we have already exploited their inversibility. From this we obtain the following immediate consequence.

Lemma 5.1 *For each $n \in \mathbb{N}$ it is decidable, whether n is the Gödel number of a term, a formula or a guarded command.* $\Box$

Next we consider the upper bound $S'_{\mathcal{I}}$ that occurs in the GCS. Since this is only a syntactic transformation, we may now conclude that $(S,\mathcal{I}) \mapsto S'_{\mathcal{I}}$ is computable. Hence it is sufficient to investigate the computability for the precondition $\mathcal{P}(S,\mathcal{I},\boldsymbol{x}')$ for arbitrary $\boldsymbol{x}'$.

These conditions involve the predicate transformers $wlp(S)$ and $wlp(S'_{\mathcal{I}})$. According to our definition of axiomatic semantics for commands, we know that building these predicate transformers is simple done by syntactic replacement operations. By exploiting our Gödel numbering h again, we conclude that for recursion-free S the mapping

$$(S, \mathcal{I}, \boldsymbol{x}') \mapsto \mathcal{P}(S, \mathcal{I}, \boldsymbol{x}')$$

—and hence $(S, \mathcal{I}) \mapsto S_{\mathcal{I}}$, too—is computable.

However, if S involves a loop, then $S'_{\mathcal{I}}$ also involves a loop. In order to determine $wlp(S)$ and $wlp(S'_{\mathcal{I}})$ we have to use the limit operator. For a loop $\mu T_j.f(T_j)$ this means to build $wlp(f^i(loop))$ for all $i \in \mathbb{N}$. This is only possible, if there is some $n \in \mathbb{N}$ such that $wlp(f^n(loop)) = wlp(f^m(loop))$ holds for all $m \geq n, m \in \mathbb{N}$. This means that we have a bounded loop (or equivalently a FOR-loop).

Proposition 5.2 *If recursive guarded commands are restricted to bounded loops, then GCSs are computable, i.e., the function $(S, \mathcal{I}) \mapsto S_{\mathcal{I}}$ is computable. In general, howver, the GCS cannot be computed.* $\square$

5.2 An Undecidability Result

Even, if the GCS $S_{\mathcal{I}}$ can be computed from the given command S and the invariant $\mathcal{I}$, the result still contains the preconditions $\mathcal{P}(S, \mathcal{I}, \boldsymbol{x}')$. If such a precondition is undecidable, then the GCS will not be effective. In the proof of the next proposition we exploit the arithmetic hierarchy [1, Ch.8].

Proposition 5.3 *In general, the preconditions $\mathcal{P}(S, \mathcal{I}, \boldsymbol{x}')$ involved in the general form of a GCS $S_{\mathcal{I}}$ are not decidable.*

Proof. (sketch) As a simple counterexample we take

$$S \equiv \underbrace{x_1 := t_1}_{S_1}; R(x_1, x_2) \to @y \bullet Q(x_1, x_2, y) \to \underbrace{x_2 := t_2}_{S_2}$$

with an underlying state space $X = \{x_1, x_2\}$ and consider a decidable $\mathcal{I}$ such that S is $\mathcal{I}$-reduced. Then we use Proposition 4.3 in order to compute the GCS $S_{\mathcal{I}}$. After a complex calculation and some simplifying steps we receive

$$\begin{gathered} \mathcal{P}(S, \mathcal{I}, (u_1, u_2)) \qquad \Leftrightarrow \\ \forall z'_2.\forall y'.\exists y.((\{x_1/t_1, x_2, z'_2\}.(\mathcal{I} \wedge R \wedge Q) \wedge \{x_1/u_1, x_2/u_2\}.\mathcal{I}) \vee \\ (\neg\mathcal{I} \wedge \{x_1/t_1, x_2/z'_2\}.(\neg\mathcal{I} \wedge R \wedge \{y/y'\}.Q))) \Rightarrow (\{x_1/u_1\}.(R \wedge Q) \wedge u_2 = t_2) \quad . \end{gathered}$$

This result is a formula in Π^0_2 (provided R and Q are recursive). $\square$

It is easy to see that undecidability results from the unbounded choice operator used in the counterexample. This produced the universal quantifier for the predicate transformer $wlp(S''_{\mathcal{I}})$ and the existential quantifier for the dual predicate transformer $wlp(S)^*$. Consequently we had to climb up two levels in the arithmetic hierarchy.

However, all classes Σ^0_i, Π^0_i and Δ^0_i in the arithmetic hierarchy are closed under bounded quantification. So, if we restrict unbounded choice to *bounded* *selection*, i.e., $@y \bullet \mathcal{P}(y) \rightarrow S$ with finite $\{y \mid\models \mathcal{P}(y)\}$, then we achieve decidability.

Proposition 5.4 *If S is a command, in which unbounded choice only occurs in the form of bounded selection, then for $\mathcal{I} \in \Sigma_0^0$ the preconditions $\mathcal{P}(S, \mathcal{I}, \boldsymbol{x}')$ involved in the general form of a GCS $S_{\mathcal{I}}$ are decidable.* □

6 Conclusion

In this article we considered the GCS approach to consistency enforcement presented in [6]. We could show that the underlying theory of predicate transformers could be carried over from an infinitary logic to first-order arithmetic logic. We were even able to do this for recursive program specifications by exploiting Gödel numberings for terms, formulae and guarded commands. However, the used recursive program specifications are slightly restricted with respect to the more general classical theory in [4].

Then we could show that the existence and uniqueness of GCSs, the commutativity result from [8] and the fundamental compositionality result carry over to the new logic. This allows to study computability and decidability issues. We could show that both properties do not hold in general, but for reasonable subclasses of program specifications.

There are at least three more problems we would like to approach next. Firstly, we would like to study the Goldfarb classification [2] and its impact to GCS construction. Secondly, we would like to look at weakened approaches to consistency enforcement, e.g., the one presented in [5] and to discuss computability and decidability for this approach as well. Thirdly and finally, we would like to address the problems of GCSs—and weakened approaches—with respect to basic commands. In particular, it would be nice to see how GCSs for various classes of relational constraints (see [9]) would look like.

Acknowledgement

The authors would like to thank Cris Calude and Rod Downey for helping to correct a proof in an earlier version of the paper.

References

[1] J. Bell, M. Machover. *A Course in Mathematical Logic*. North-Holland 1977.

[2] E. Börger, E. Grädel, Y. Gurevich. *The Classical Decision Problem*. Springer 1997.

[3] S. Link. *Eine Theorie der Konsistenzerzwingung auf der Basis arithmetischer Logik*. M.Sc. Thesis (in German). TU Clausthal 2000.

[4] G. Nelson. A Generalization of Dijkstra's Calculus. *ACM TOPLAS*. vol. 11 (4): 517-561. 1989.

[5] K.-D. Schewe. Fundamentals of Consistency Enforcement. In H. Jaakkola, H. Kangassalo, E. Kawaguchi (eds.). *Information Modelling and Knowledge Bases* X: 275-291. IOS Press 1999.

[6] K.-D. Schewe, B. Thalheim. Towards a Theory of Consistency Enforcement. *Acta Informatica*. vol. 36: 97-141. 1999.

[7] K.-D. Schewe, B. Thalheim. Limitations of Rule Triggering Systems for Integrity Maintenance in the Context of Transition Specifications. *Acta Cybernetica*. vol. 13: 277-304. 1998.

[8] K.-D. Schewe, B. Thalheim, J. Schmidt, I. Wetzel. Integrity Enforcement in Object Oriented Databases. In U. Lipeck, B. Thalheim (eds.). *Modelling Database Dynamics*: 174-195. Workshops in Computing. Springer 1993.

[9] B. Thalheim. *Dependencies in Relational Databases*. Teubner 1991.

A Existence, normal form representation and commutativity of GCSs

In the appendix we give a detailed proof of Proposition 4.3.
Proposition 4.3 *The GCS $S_{\mathcal{I}}$ of S with respect to $\mathcal{I}$ always exists and is unique up to semantic equivalence. We can always write*

$$S_{\mathcal{I}} = (\mathcal{I} \rightarrow S; @\boldsymbol{z}' \bullet \boldsymbol{z} := \boldsymbol{z}'; \mathcal{I} \rightarrow skip) \boxtimes (\neg\mathcal{I} \rightarrow S; @\boldsymbol{z}' \bullet \boldsymbol{z} := \boldsymbol{z}') ,$$

where $\boldsymbol{z}$ refers to the free variables in $\mathcal{I}$ not occurring in S.

Furthermore, for two invariants $\mathcal{I}$ and $\mathcal{J}$ we always obtain that $\mathcal{I} \wedge \mathcal{J} \rightarrow S_{\mathcal{I}\wedge\mathcal{J}}$ and $\mathcal{I} \wedge \mathcal{J} \rightarrow (S_{\mathcal{I}})_{\mathcal{J}}$ are semantically equivalent.

Proof. First we show the existence and uniqueness up to semantic equivalence of GCS. We set

$$\mathcal{T} = \{T \mid T \sqsubseteq S \text{ and } T \text{ ist consistent with respect to } \mathcal{I}\} \quad .$$

If the least upper bound $S_{\mathcal{I}}$ of $\mathcal{T}$ with respect to the specialization $\sqsubseteq$ exists, then this must be the GCS. Therefore, we have the uniqueness up to semantic equivalence.
We now verify the conditions from Definition 4.2 for the program specification

$S_{\mathcal{I}}$ above. Let φ be an arbitrary state formula on Y. Then we receive

$$\begin{aligned}
wlp(S_{\mathcal{I}})^*(\varphi) \;\Leftrightarrow\; & (\mathcal{I} \wedge wlp(S)^*(\exists \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.(\mathcal{I} \wedge \varphi))) \vee \\
& (\neg\mathcal{I} \wedge wlp(S)^*(\exists \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\varphi)) \\
\Leftrightarrow\; & (\mathcal{I} \wedge wlp(S)^*((\exists \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\mathcal{I}) \wedge \varphi)) \vee (\neg\mathcal{I} \wedge wlp(S)^*(\varphi)) \\
\Rightarrow\; & (\mathcal{I} \wedge wlp(S)^*(\varphi)) \vee (\neg\mathcal{I} \wedge wlp(S)^*(\varphi)) \\
\Leftrightarrow\; & wlp(S)^*(\varphi) \quad .
\end{aligned}$$

Doing this we have made use of the dual predicate transformers' monotonicity property and the fact that variables z_i do not occur within φ. Then the asserted specialization $S_{\mathcal{I}} \sqsubseteq S$ follows from the same computation for *wp* instead of *wlp*.
Next we consider

$$\begin{aligned}
wlp(S_{\mathcal{I}})(\mathcal{I}) \quad \Leftrightarrow \quad & (\mathcal{I} \Rightarrow wlp(S)(\forall \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.(\mathcal{I} \Rightarrow \mathcal{I}))) \wedge \\
& (\neg\mathcal{I} \Rightarrow wlp(S)(\forall \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\mathcal{I})) \\
\Leftrightarrow \quad & (\neg\mathcal{I} \Rightarrow wlp(S)(\forall \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\mathcal{I})) \quad .
\end{aligned}$$

Due to $\neg wlp(S_{\mathcal{I}})(\mathcal{I}) \Leftrightarrow \neg\mathcal{I} \wedge \neg wlp(S)(\forall \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\mathcal{I})$ we obtain $\mathcal{I} \Rightarrow wlp(S_{\mathcal{I}})(\mathcal{I})$ which means that the above $S_{\mathcal{I}}$ is indeed consistent with respect to $\mathcal{I}$.
Let $\boldsymbol{x} = \boldsymbol{y}$ be a characterizing state formula and $T \sqsubseteq S$ an arbitrary, but $\mathcal{I}$-consistent specialization of S. Then we ditinguish two cases.
Case 1. We assume $\boldsymbol{x} = \boldsymbol{y} \Rightarrow \neg\mathcal{I}$ and therefore, we conclude $wlp(T)^*(\boldsymbol{x} = \boldsymbol{y}) \Rightarrow wlp(T)^*(\neg\mathcal{I}) \Rightarrow \neg\mathcal{I}$ using the monotonicity of $wlp(S)^*$ and consistency of T. Moreover, it follows

$$\begin{aligned}
wlp(T)^*(\boldsymbol{x} = \boldsymbol{y}) \quad \Rightarrow \quad & \neg\mathcal{I} \wedge wlp(S)^*(\boldsymbol{x} = \boldsymbol{y}) \\
\Rightarrow \quad & \neg\mathcal{I} \wedge wlp(S)^*(\exists \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\boldsymbol{x} = \boldsymbol{y}) \\
\Rightarrow \quad & wlp(S_{\mathcal{I}})^*(\boldsymbol{x} = \boldsymbol{y}) \quad .
\end{aligned}$$

For the first implication we simply use the specialization $T \sqsubseteq S$, for the second we refer to the monotonicity applied to $\boldsymbol{x} = \boldsymbol{y} \Rightarrow \exists \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\boldsymbol{x} = \boldsymbol{y}$ and the last one follows from the first line of the computation of $wlp(S_{\mathcal{I}})^*$
Case 2. Now we start out from $\boldsymbol{x} = \boldsymbol{y} \Rightarrow \mathcal{I}$ and we derive $wlp(T)^*(\boldsymbol{x} = \boldsymbol{y}) \Leftrightarrow wlp(T)^*(\mathcal{I} \wedge \boldsymbol{x} = \boldsymbol{y})$, consequentely. With $T \sqsubseteq S$ and the monotonicity of $wlp(S)^*$ we conclude

$$wlp(S)^*(\exists \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.(\mathcal{I} \wedge \boldsymbol{x} = \boldsymbol{y})) \wedge wlp(S)^*(\exists \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.(\boldsymbol{x} = \boldsymbol{y})) \quad . \qquad (*)$$

Obviously, we have $(wlp(T)^*(\boldsymbol{x} = \boldsymbol{y}) \wedge \mathcal{I}) \vee (wlp(T)^*(\boldsymbol{x} = \boldsymbol{y}) \Rightarrow \neg\mathcal{I})$ and together with $(*)$ we receive

$$\begin{aligned} wlp(T)^*(\boldsymbol{x}=\boldsymbol{y}) &\Leftrightarrow (\mathcal{I}\wedge wlp(S)^*(\exists \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.(\mathcal{I}\wedge \boldsymbol{x}=\boldsymbol{y})))\vee \\ &\quad\ (\neg\mathcal{I}\wedge wlp(S)^*(\exists \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\boldsymbol{x}=\boldsymbol{y})) \\ &\Leftrightarrow wlp(S_{\mathcal{I}})^*(\boldsymbol{x}=\boldsymbol{y}) \quad . \end{aligned}$$

This first step has brought us to $wlp(T)^*(\boldsymbol{x}=\boldsymbol{y}) \Rightarrow wlp(S_{\mathcal{I}})^*(\boldsymbol{x}=\boldsymbol{y})$, i.e., $wlp(S_{\mathcal{I}})(\boldsymbol{x}\neq\boldsymbol{y}) \Rightarrow wlp(T)(\boldsymbol{x}\neq\boldsymbol{y})$. For arbitrary state formula φ we have $\varphi(\boldsymbol{x}) \Leftrightarrow \forall\boldsymbol{y}.\neg\varphi(\boldsymbol{y}) \Rightarrow \boldsymbol{x}\neq\boldsymbol{y}$ and therefore

$$\begin{aligned} wlp(S_{\mathcal{I}})(\varphi(\boldsymbol{x})) &\Leftrightarrow \forall\boldsymbol{y}.\neg\varphi(\boldsymbol{y}) \Rightarrow wlp(S_{\mathcal{I}})(\boldsymbol{x}\neq\boldsymbol{y}) \\ &\Rightarrow \forall\boldsymbol{y}.\neg\varphi(\boldsymbol{y}) \Rightarrow wlp(T)(\boldsymbol{x}\neq\boldsymbol{y}) \\ &\Leftrightarrow wlp(T)(\varphi(\boldsymbol{x})) \quad , \end{aligned}$$

using the universal conjunctivity property of wlp. We obtain $wlp(T)^*(\varphi) \Rightarrow wlp(S_{\mathcal{I}})^*(\varphi)$ for all φ and on top of that $wp(T)^*(false) \Rightarrow wp(S)^*(false) \Rightarrow wp(S_{\mathcal{I}})^*(false)$ holds as well, due to the specialization $T \sqsubseteq S$ and the first line of the computation of $wlp(S_{\mathcal{I}})^*$ above. Indeed, we have proved that T is a specialization of $S_{\mathcal{I}}$.

Let us now consider the asserted commutativity result. Since $(S_{\mathcal{I}_1})_{\mathcal{I}_2}$ is $\mathcal{I}_2$-consistent of definition we have

$$\mathcal{I}_2 \Rightarrow wlp((S_{\mathcal{I}_1})_{\mathcal{I}_2})(\mathcal{I}_2) \quad .$$

On the other side we can use the definition of GCS and consistency as well as $(S_{\mathcal{I}_1})_{\mathcal{I}_2} \sqsubseteq S_{\mathcal{I}_1}$ in order to receive

$$\mathcal{I}_1 \Rightarrow wlp(S_{\mathcal{I}_1})(\mathcal{I}_1) \Rightarrow wlp((S_{\mathcal{I}_1})_{\mathcal{I}_2})(\mathcal{I}_1) \quad .$$

In summary, this results in

$$\mathcal{I}_1\wedge\mathcal{I}_2 \Rightarrow wlp((S_{\mathcal{I}_1})_{\mathcal{I}_2})(\mathcal{I}_1) \wedge wlp((S_{\mathcal{I}_1})_{\mathcal{I}_2})(\mathcal{I}_2) \Leftrightarrow wlp((S_{\mathcal{I}_1})_{\mathcal{I}_2})(\mathcal{I}_1\wedge\mathcal{I}_2) \quad ,$$

so we have proved the consistency of $(S_{\mathcal{I}_1})_{\mathcal{I}_2}$ with respect to $\mathcal{I}_1\wedge\mathcal{I}_2$. From $S_{\mathcal{I}_1} \sqsubseteq S$ and $(S_{\mathcal{I}_1})_{\mathcal{I}_2} \sqsubseteq S_{\mathcal{I}_1}$ we derive

$$wlp(S)(\varphi) \Rightarrow wlp(S_{\mathcal{I}_1})(\varphi) \Rightarrow wlp((S_{\mathcal{I}_1})_{\mathcal{I}_2})(\varphi) \quad ,$$

i.e., the specialization $(S_{\mathcal{I}_1})_{\mathcal{I}_2} \sqsubseteq S$. Consequentely, this together with definition 4.2 yields $(S_{\mathcal{I}_1})_{\mathcal{I}_2} \sqsubseteq S_{\mathcal{I}_1\wedge\mathcal{I}_2}$ and we obtain

$$\begin{aligned} wlp(\mathcal{I}_1\wedge\mathcal{I}_2 \rightarrow S_{\mathcal{I}_1\wedge\mathcal{I}_2})(\varphi) &\Leftrightarrow \mathcal{I}_1\wedge\mathcal{I}_2 \Rightarrow wlp(S_{\mathcal{I}_1\wedge\mathcal{I}_2})(\varphi) \\ &\Rightarrow \mathcal{I}_1\wedge\mathcal{I}_2 \Rightarrow wlp((S_{\mathcal{I}_1})_{\mathcal{I}_2})(\varphi) \\ &\Leftrightarrow wlp(\mathcal{I}_1\wedge\mathcal{I}_2 \rightarrow (S_{\mathcal{I}_1})_{\mathcal{I}_2})(\varphi) \end{aligned}$$

for arbitrary φ which means $(\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow (S_{\mathcal{I}_1})_{\mathcal{I}_2}) \sqsubseteq (\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2})$. Thus, it remains to show the reverse specialization.
From $S_{\mathcal{I}_1 \wedge \mathcal{I}_2} \sqsubseteq S$ follows

$$(\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2}) \sqsubseteq S \qquad . \tag{A.1}$$

In addition, $S_{\mathcal{I}_1 \wedge \mathcal{I}_2}$ is consistent with respect to $\mathcal{I}_1 \wedge \mathcal{I}_2$ of definition, so we have not only $\mathcal{I}_1 \wedge \mathcal{I}_2 \Rightarrow wlp(S_{\mathcal{I}_1 \wedge \mathcal{I}_2})(\mathcal{I}_1)$ but also $\mathcal{I}_1 \wedge \mathcal{I}_2 \Rightarrow wlp(S_{\mathcal{I}_1 \wedge \mathcal{I}_2})(\mathcal{I}_2)$. Next we consider

$$(\mathcal{I}_1 \Rightarrow wlp(\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2})(\mathcal{I}_1)) \Leftrightarrow (\mathcal{I}_1 \wedge \mathcal{I}_2 \Rightarrow wlp(S_{\mathcal{I}_1 \wedge \mathcal{I}_2})(\mathcal{I}_1)) \tag{A.2}$$

and

$$(\mathcal{I}_2 \Rightarrow wlp(\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2})(\mathcal{I}_2)) \Leftrightarrow (\mathcal{I}_1 \wedge \mathcal{I}_2 \Rightarrow wlp(S_{\mathcal{I}_1 \wedge \mathcal{I}_2})(\mathcal{I}_2)) \quad . \tag{A.3}$$

From equation (A.2) we obtain the consistency of $\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2}$ with respect to $\mathcal{I}_1$ and using equation (A.1) yields

$$(\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2}) \sqsubseteq S_{\mathcal{I}_1} \quad . \tag{A.4}$$

From equation (A.3) follows the consistency of $\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2}$ with respect to $\mathcal{I}_2$ and using equation (A.4) we conclude

$$(\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2}) \sqsubseteq (S_{\mathcal{I}_1})_{\mathcal{I}_2} \quad . \tag{A.5}$$

Finally, we compute

$$\begin{aligned}
w(l)p(\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow (S_{\mathcal{I}_1})_{\mathcal{I}_2})(\varphi) &\Leftrightarrow \mathcal{I}_1 \wedge \mathcal{I}_2 \Rightarrow w(l)p((S_{\mathcal{I}_1})_{\mathcal{I}_2})(\varphi) \\
&\Rightarrow \mathcal{I}_1 \wedge \mathcal{I}_2 \Rightarrow w(l)p(\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2})(\varphi) \\
&\Leftrightarrow \mathcal{I}_1 \wedge \mathcal{I}_2 \Rightarrow (\mathcal{I}_1 \wedge \mathcal{I}_2 \Rightarrow w(l)p(S_{\mathcal{I}_1 \wedge \mathcal{I}_2})(\varphi)) \\
&\Leftrightarrow \mathcal{I}_1 \wedge \mathcal{I}_2 \Rightarrow w(l)p(S_{\mathcal{I}_1 \wedge \mathcal{I}_2})(\varphi) \\
&\Leftrightarrow w(l)p(\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2})(\varphi)
\end{aligned}$$

the specialization $\mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow S_{\mathcal{I}_1 \wedge \mathcal{I}_2} \sqsubseteq \mathcal{I}_1 \wedge \mathcal{I}_2 \rightarrow (S_{\mathcal{I}_1})_{\mathcal{I}_2}$, where we just make use of equation (A.5) in the appearing implication. This completes the proof.$\Box$

B Proof of the upper bound theorem in the recursive case

In this appendix we prove the upper bound theorem for recursive operations restricted to simple WHILE-loops in the form of $f(S) = \mathcal{P} \rightarrow T; S \Box \neg\mathcal{P} \rightarrow$

skip for which we know the existence of least fixpoints according to subsection 3.2. For this we need some additional lemmata.

For recursive guarded commands the monotonicity of all operation constructors with respect to the Nelson-order $\preceq$ is fundamental [4]. Unfortunately, a similiar result does not hold for the specialization order $\sqsubseteq$. More precisely, the result is false for the $\boxtimes$-constructor in its first component.

Lemma B.1 *Let $f(S)$ be a guarded-command expression with the program variable S in which restricted choice $\boxtimes$ does not occur. Then f is monotonic with respect to the specialization order $\sqsubseteq$.*

Proof. (sketch) The proof is done by structural induction. For each constructor it is completely analogous to the corresponding proof for the Nelson-order in [4]. We omit the details. $\square$

In [6, Proposition 20, p.120] we have seen that $S'_{\mathcal{I}}$ may contain the choice-constructor instead of restricted choice, provided we include some guard. Replacing within a recursive operation some $S_1 \boxtimes S_2$ by $(S_1)_{\mathcal{I}} \boxtimes (S_2)_{\mathcal{I}}$ would destroy the required result.

The next lemma follows from taking together the cases in the upper bound theorem for preconditionings $\rightarrow$, choices $\square$, unbounded choices @ and restricted choices $\boxtimes$.

Lemma B.2 *Let T be a consistent specialization of some $\mathcal{I}$-reduced $f(S')$ with respect to $\mathcal{I}$, where $f(S)$ is an expression built from the constructors of guarded commands. Construct $f_{\mathcal{I}}(S)$ from $f(S)$ as follows:*

(i) Each restricted choice $S_1 \boxtimes S_2$ occuring within $f(S)$ will be replaced by $S_1 \square wlp(S_1)(false) \rightarrow S_2$.

(ii) Then each basic operation, i.e., skip and assignments will be replaced by their GCSs with respect to $\mathcal{I}$.

Then we have $T \sqsubseteq f_{\mathcal{I}}(S'_{\mathcal{I}})$. $\square$

As in the classical we must now face the main difficulty to bring together two different partial orders, namely the specialization order $\sqsubseteq$ which is fundamental for GCSs and the Nelson-order $\preceq$ required for recursion.

Lemma B.3 *Let T und S be Y-operations and $\{f^i(loop) \mid i \in \mathbb{N}\}$ the chain of Y-operations with respect to the Nelson-order. Furthermore, let $\mathcal{I}$ be an invariant on X for $Y \subseteq X$. Then we have:*

(i) If $T \preceq S$ holds, then $T_{\mathcal{I}} \preceq S_{\mathcal{I}}$ follows.

(ii) $(\lim_{i\in\mathbb{N}} f^i(loop))_{\mathcal{I}} \sqsubseteq \lim_{i\in\mathbb{N}}(f^i(loop))_{\mathcal{I}}$.

Proof. (i) Here we use the normal form of a GCS given in Proposition 4.3. The first result follows immediately, because all constructors are monotonic in the Nelson-order $\preceq$.

(ii) First, $\lim_{i\in\mathbb{N}} f^i(loop)$ is the least upper bound of $\{f^i(loop) \mid i \in \mathbb{N}\}$ with respect to the Nelson-order according to Lemma 3.4, i.e., especially

$f^i(\mathit{loop}) \preceq \lim_{i\in\mathbb{N}} f^i(\mathit{loop})$ holds for arbitrary $i \in \mathbb{N}$. From this and (i) we get $(f^i(\mathit{loop}))_{\mathcal{I}} \preceq (\lim_{i\in\mathbb{N}} f^i(\mathit{loop}))_{\mathcal{I}}$. Using Lemma 3.4 again, $\lim_{i\in\mathbb{N}}(f^i(\mathit{loop}))_{\mathcal{I}}$ is obviously the least upper bound of the chain $\{(f^i(\mathit{loop}))_{\mathcal{I}} \mid i \in \mathbb{N}\}$ which means that $\lim_{i\in\mathbb{N}}(f^i(\mathit{loop}))_{\mathcal{I}} \preceq (\lim_{i\in\mathbb{N}} f^i(\mathit{loop}))_{\mathcal{I}}$ must hold. Therefore, we receive

$$wp\left(\lim_{i\in\mathbb{N}}(f^i(\mathit{loop}))_{\mathcal{I}}\right)(\varphi) \Rightarrow wp\left(\left(\lim_{i\in\mathbb{N}} f^i(\mathit{loop})\right)_{\mathcal{I}}\right)(\varphi)$$

according to the definition of the Nelson-order.
Once again we make use of Proposition 4.3 in order to compute

$$\begin{aligned}
& wlp\left(\lim_{i\in\mathbb{N}}(f^i(\mathit{loop}))_{\mathcal{I}}\right)(\varphi) \Leftrightarrow \\
& \forall i. i \in \mathbb{N} \Rightarrow wlp\left((f^i(\mathit{loop}))_{\mathcal{I}}\right)(\varphi) \Leftrightarrow \\
& \forall i. i \in \mathbb{N} \Rightarrow wlp((\mathcal{I} \rightarrow f^i(\mathit{loop}); @\boldsymbol{z}' \bullet \boldsymbol{z} := \boldsymbol{z}'; \mathcal{I} \rightarrow \mathit{skip})\boxtimes \\
& \qquad (\neg\mathcal{I} \rightarrow f^i(\mathit{loop}); @\boldsymbol{z}' \bullet \boldsymbol{z} := \boldsymbol{z}'))(\varphi) \Leftrightarrow \\
& \forall i. i \in \mathbb{N} \Rightarrow ((\mathcal{I} \Rightarrow wlp(f^i(\mathit{loop}))(\forall \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\mathcal{I} \Rightarrow \varphi))\wedge \\
& \qquad (\neg\mathcal{I} \Rightarrow wlp(f^i(\mathit{loop}))(\forall \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\varphi))) \Leftrightarrow \\
& (\mathcal{I} \Rightarrow \forall i. i \in \mathbb{N} \Rightarrow wlp(f^i(\mathit{loop}))(\forall \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\mathcal{I} \Rightarrow \varphi))\wedge \\
& \qquad (\neg\mathcal{I} \Rightarrow \forall i. i \in \mathbb{N} \Rightarrow wlp(f^i(\mathit{loop}))(\forall \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\varphi)) \Leftrightarrow \\
& (\mathcal{I} \Rightarrow wlp\left(\lim_{i\in\mathbb{N}} f^i(\mathit{loop})\right)(\forall \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\mathcal{I} \Rightarrow \varphi))\wedge \\
& \qquad (\neg\mathcal{I} \Rightarrow wlp\left(\lim_{i\in\mathbb{N}} f^i(\mathit{loop})\right)(\forall \boldsymbol{z}'.\{\boldsymbol{z}/\boldsymbol{z}'\}.\varphi)) \Leftrightarrow \\
& wlp((\mathcal{I} \rightarrow \lim_{i\in\mathbb{N}} f^i(\mathit{loop}); @\boldsymbol{z}' \bullet \boldsymbol{z} := \boldsymbol{z}'; \mathcal{I} \rightarrow \mathit{skip})\boxtimes \\
& \qquad (\neg\mathcal{I} \rightarrow \lim_{i\in\mathbb{N}} f^i(\mathit{loop}); @\boldsymbol{z}' \bullet \boldsymbol{z} := \boldsymbol{z}'))(\varphi) \Leftrightarrow \\
& wlp\left(\left(\lim_{i\in\mathbb{N}} f^i(\mathit{loop})\right)_{\mathcal{I}}\right)(\varphi) \quad ,
\end{aligned}$$

i.e.,

$$wlp\left(\lim_{i\in\mathbb{N}}(f^i(\mathit{loop}))_{\mathcal{I}}\right)(\varphi) \Rightarrow wlp\left(\left(\lim_{i\in\mathbb{N}} f^i(\mathit{loop})\right)_{\mathcal{I}}\right)(\varphi) \quad ,$$

supplies the asserted specialization. □

We are now able to give the main proof.

Proposition B.4 Let $S' = \mu T_j.f(T_j)$ with $f(T_j) = \mathcal{P} \rightarrow T; T_j \Box \neg\mathcal{P} \rightarrow \mathit{skip}$ be an $\mathcal{I}$-reduced Y-operation and $T \sqsubseteq S'$ a consistent specialization with respect to some X-invariant $\mathcal{I}$ with $Y \subseteq X$. Then we have $T \sqsubseteq \mu T_j.f_{\mathcal{I}}(T_j)$, where $f_{\mathcal{I}}(T_j)$ is built as in Lemma B.2.

Proof. Since S' is a fixpoint we have $S' = f(S')$. T is an $\mathcal{I}$-reduced consistent specialization of S' by assumption, so the specialization

$$T \sqsubseteq f_{\mathcal{I}}(S'_{\mathcal{I}}) = f_{\mathcal{I}}\left(\left(\lim_{i\in\mathbb{N}} f^i(loop)\right)_{\mathcal{I}}\right) \quad .$$

follows by Lemma B.2. Due to the monotonicity of $f_{\mathcal{I}}$ and because of Lemma B.3 (ii) we derive further

$$f_{\mathcal{I}}\left(\left(\lim_{i\in\mathbb{N}} f^i(loop)\right)_{\mathcal{I}}\right) \sqsubseteq f_{\mathcal{I}}\left(\underbrace{\lim_{i\in\mathbb{N}} \overbrace{\left(f^i(loop)\right)_{\mathcal{I}}}^{T_{1i}}}_{T_1}\right) \quad .$$

We set $T_{2i} = f^i_{\mathcal{I}}(loop)$ and show $T_{1i} \sqsubseteq T_{2i}$ for all $i \in \mathbb{N}$ by induction. The case $i = 0$ gives $T_{10} = loop_{\mathcal{I}} = loop = T_{20}$. In the case $i > 0$ we can assume $T_{1j} \sqsubseteq T_{2j}$ for all $j < i$. T_{1i} is an $\mathcal{I}$-consistent specialization of $f^i(loop) = f(f^{i-1}(loop))$, hence we conclude

$$T_{1i} \sqsubseteq f_{\mathcal{I}}\left(\left(f^{i-1}(loop)\right)_{\mathcal{I}}\right) = f_{\mathcal{I}}\left(T_{1(i-1)}\right).$$

by Lemma B.2. Now, we apply the induction hypothesis and the monotonicity of $f_{\mathcal{I}}$ in order to obtain $f_{\mathcal{I}}\left(T_{1(i-1)}\right) \sqsubseteq f_{\mathcal{I}}\left(T_{2(i-1)}\right) = T_{2i}$, i.e., together $T_{1i} \sqsubseteq T_{2i}$ as asserted.
For $T_2 = \lim_{i\in\mathbb{N}} f^i_{\mathcal{I}}(loop)$ follows

$$\begin{aligned} wlp(T_2)(\varphi) \quad &\Leftrightarrow \quad (\forall i.i \in \mathbb{N} \Rightarrow wlp(T_{2i})(\varphi)) \\ &\Rightarrow \quad (\forall i.i \in \mathbb{N} \Rightarrow wlp(T_{1i})(\varphi)) \\ &\Leftrightarrow \quad wlp(T_1)(\varphi) \end{aligned}$$

and

$$\begin{aligned} wp(T_2)(\varphi) \quad &\Leftrightarrow \quad (\exists i.i \in \mathbb{N} \wedge wp(T_{2i})(\varphi)) \\ &\Rightarrow \quad (\exists i.i \in \mathbb{N} \wedge wp(T_{1i})(\varphi)) \\ &\Leftrightarrow \quad wp(T_1)(\varphi) \quad , \end{aligned}$$

thus the specialization $T_1 \sqsubseteq T_2$. Finally, we receive by applying Lemma B.1

$$T \sqsubseteq f_{\mathcal{I}}(T_1) \sqsubseteq f_{\mathcal{I}}(T_2) = T_2 = \mu T_j.f_{\mathcal{I}}(T_j) \quad ,$$

where we use the fact that T_2 is a fixpoint. □

C A detailed proof of the compositionality result

We still need an additional proposition which gives a normal form for specialization. The proof is completely the same as in the classical case [6, App.B] and omitted, therefore.

Proposition C.1 *Let S and T be commands on the state spaces Z and Y, respectively, with $Z \subseteq Y$. Then $wlp(S)(\varphi) \Rightarrow wlp(T)(\varphi)$ holds for all Z-formulae iff*

$$\{\boldsymbol{z}'/\boldsymbol{z}\}.wlp(T')(wlp(S)^*(\boldsymbol{z} = \boldsymbol{z}'))$$

holds, where $\boldsymbol{z}'$ is a disjoint copy of $\boldsymbol{z}$ and T' results from T by renaming each z_i into z_i'. □

Now we can show the main result with respect to the GCS-construction.

Theorem 4.7 *Let $\mathcal{I}$, S and $S'_{\mathcal{I}}$ be as in Theorem 4.6. Let Z be a disjoint copy of the state space Y. With the formulae*

$$\mathcal{P}(S, \mathcal{I}, \boldsymbol{x}') \equiv \{\boldsymbol{z}/\boldsymbol{y}\}.wlp(S''_{\mathcal{I}}; \boldsymbol{z} = \boldsymbol{x}' \to skip)(wlp(S)^*(\boldsymbol{z} = \boldsymbol{y})) \quad ,$$

where $S''_{\mathcal{I}}$ results from $S'_{\mathcal{I}}$ by renaming the Y to Z, the GCS $S_{\mathcal{I}}$ is semantically equivalent to

$$@\boldsymbol{x}' \bullet \mathcal{P}(S, \mathcal{I}, \boldsymbol{x}') \to S'_{\mathcal{I}}; \boldsymbol{y} = \boldsymbol{x}' \to skip \quad .$$

Proof. We take the form claimed in the theorem as a definition and verify the conditions in the definition of the GCS. First let φ be an arbitrary Y-formula. For this we get by assistance of Proposition C.1

$$wlp(S_{\mathcal{I}})^*(\varphi) \Leftrightarrow (\exists \boldsymbol{x}'.\mathcal{P}(S, \mathcal{I}, \boldsymbol{x}') \wedge wlp(S'_{\mathcal{I}})^*(\boldsymbol{y} = \boldsymbol{x}' \wedge \varphi)) \Rightarrow wlp(S)^*(\varphi) \quad ,$$

which gives the specialization condition $S_{\mathcal{I}} \sqsubseteq S$. The wp-part follows analogously.

Consistency can be verified easily, since $S'_{\mathcal{I}}$ is already consistent with respect to $\mathcal{I}$, namely

$$\begin{aligned}
\mathcal{I} \quad &\Rightarrow \quad wlp(S'_{\mathcal{I}})(\mathcal{I}) \\
&\Rightarrow \quad wlp(S'_{\mathcal{I}})(\boldsymbol{y} = \boldsymbol{x}' \Rightarrow wlp(skip)(\mathcal{I})) \\
&\Leftrightarrow \quad wlp(S'_{\mathcal{I}})(wlp(\boldsymbol{y} = \boldsymbol{x}' \to skip)(\mathcal{I})) \\
&\Leftrightarrow \quad wlp(S'_{\mathcal{I}}; \boldsymbol{y} = \boldsymbol{x}' \to skip)(\mathcal{I}) \\
&\Rightarrow \quad \forall \boldsymbol{x}'.P(S, \mathcal{I}, \boldsymbol{x}') \Rightarrow wlp(S'_{\mathcal{I}}; \boldsymbol{y} = \boldsymbol{x}' \to skip)(\mathcal{I}) \\
&\Leftrightarrow \quad wlp(@\boldsymbol{x}' \bullet P(S, \mathcal{I}, \boldsymbol{x}') \to S'_{\mathcal{I}}; \boldsymbol{y} = \boldsymbol{x}')(\mathcal{I}) \\
&\Leftrightarrow \quad wlp(S_{\mathcal{I}})(\mathcal{I}) \quad .
\end{aligned}$$

Therefore, we have the consistency of $S_{\mathcal{I}}$ with respect to $\mathcal{I}$. Note, that the second implication in the computation above holds due to the monotonicity of $wlp(S'_{\mathcal{I}})$ applied to $\mathcal{I} \Rightarrow (\boldsymbol{y} = \boldsymbol{x}' \Rightarrow \mathcal{I})$.

Finally, let T be an arbitrary consistent specialization of S. We assume without loss of generality that $wp(T)(true) \Leftrightarrow true$ holds. From Theorem 4.6 we already get $T \sqsubseteq S'_{\mathcal{I}}$. From this we compute

$$\begin{aligned} w(l)p(\underbrace{S'_{\mathcal{I}}; \boldsymbol{y} = \boldsymbol{x}' \rightarrow skip}_{S_{\mathcal{I}}^{\boldsymbol{x}'}})(\varphi) &\Leftrightarrow w(l)p(S'_{\mathcal{I}})(w(l)p(\boldsymbol{y} = \boldsymbol{x}' \rightarrow skip)(\varphi)) \\ &\Rightarrow w(l)p(T)(w(l)p(\boldsymbol{y} = \boldsymbol{x}' \rightarrow skip)(\varphi)) \\ &\Leftrightarrow w(l)p(\underbrace{T; \boldsymbol{y} = \boldsymbol{x}' \rightarrow skip}_{T^{\boldsymbol{x}'}})(\varphi) \quad , \end{aligned}$$

i.e., $T^{\boldsymbol{x}'} \sqsubseteq S_{\mathcal{I}}^{\boldsymbol{x}'}$. At this point it suffices to show $wp(T^{\boldsymbol{x}'})^*(true) \Rightarrow P(S, \mathcal{I}, \boldsymbol{x}')$, because

$$\begin{aligned} w(l)p(P(S, \mathcal{I}, \boldsymbol{x}') \rightarrow S_{\mathcal{I}}^{\boldsymbol{x}'})(\varphi) &\Leftrightarrow P(S, \mathcal{I}, \boldsymbol{x}') \Rightarrow w(l)p(S_{\mathcal{I}}^{\boldsymbol{x}'})(\varphi) \\ &\Rightarrow wp(T^{\boldsymbol{x}'})^*(true) \Rightarrow w(l)p(S_{\mathcal{I}}^{\boldsymbol{x}'})(\varphi) \\ &\Rightarrow wp(T^{\boldsymbol{x}'})^*(true) \Rightarrow w(l)p(T^{\boldsymbol{x}'})(\varphi) \\ &\Leftrightarrow w(l)p(\underbrace{wp(T^{\boldsymbol{x}'})^*(true) \rightarrow T^{\boldsymbol{x}'}}_{\Leftrightarrow T^{\boldsymbol{x}'}})(\varphi) \end{aligned}$$

implies immediately $T^{\boldsymbol{x}'} \sqsubseteq P(S, \mathcal{I}, \boldsymbol{x}') \rightarrow S_{\mathcal{I}}^{\boldsymbol{x}'}$ and we obtain $\forall \boldsymbol{x}' \bullet T^{\boldsymbol{x}'} \sqsubseteq \forall \boldsymbol{x}' \bullet P(S, \mathcal{I}, \boldsymbol{x}') \rightarrow S_{\mathcal{I}}^{\boldsymbol{x}'}$, consequently. The formula on the left-hand side is equivalent to T, whereas the one on the right-hand side is equivalent to $S_{\mathcal{I}}$. Assume there is a state $\boldsymbol{a}$, in which $\mathcal{P}(S, \mathcal{I}, \boldsymbol{x}')$ does not hold. From Proposition C.1 we get the existence of a state $\boldsymbol{b}$ with

$$\models_{\boldsymbol{a}} \neg\,(wlp(S)(\boldsymbol{y} \neq \boldsymbol{b}) \Rightarrow wlp(S'_{\mathcal{I}}; \boldsymbol{y} = \boldsymbol{x}' \rightarrow skip)(\boldsymbol{y} \neq \boldsymbol{b})) \quad ,$$

which is equivalent to

$$\models_{\boldsymbol{a}} wlp(S)(\boldsymbol{y} \neq \boldsymbol{b}) \wedge \neg wlp(S'_{\mathcal{I}})(\boldsymbol{y} = \boldsymbol{x}' \Rightarrow \boldsymbol{y} \neq \boldsymbol{b})$$

and this, finally, to

$$\models_{\boldsymbol{a}} wlp(S)(\boldsymbol{y} \neq \boldsymbol{b}) \wedge wlp(S'_{\mathcal{I}})^*(\boldsymbol{y} = \boldsymbol{x}' \wedge \boldsymbol{y} = \boldsymbol{b}) \quad .$$

Hence $\boldsymbol{x}' = \boldsymbol{b}$ must hold by definition of characterizing state formulae. On the other hand we receive $\models_{\boldsymbol{a}} wlp(T)(\boldsymbol{y} \neq \boldsymbol{b})$ due to $T \sqsubseteq S$ and together with

$$\begin{aligned} wlp(T^{\boldsymbol{x}'})(false) &\Leftrightarrow wlp(T)(\boldsymbol{y} = \boldsymbol{x}' \Rightarrow false) \\ &\Leftrightarrow wlp(T)(\boldsymbol{y} \neq \boldsymbol{x}') \\ &\Leftrightarrow wlp(T)(\boldsymbol{y} \neq \boldsymbol{b}) \end{aligned}$$

we conclude $\models_{\boldsymbol{a}} wlp(T^{\boldsymbol{x}'})(false)$. The pairing condition $wp(T^{\boldsymbol{x}'})(false) \Leftrightarrow wlp(T^{\boldsymbol{x}'})(false) \wedge wp(T^{\boldsymbol{x}'})(true)$ and

$$wp(T^{\boldsymbol{x}'})(true) \Leftrightarrow wp(T)(\boldsymbol{y} = \boldsymbol{x}' \Rightarrow true) \Leftrightarrow wp(T)(true) \Leftrightarrow true$$

give $\models_{\boldsymbol{a}} wp(T^{\boldsymbol{x}'})(false)$, which is equivalent to $\models_{\boldsymbol{a}} \neg wp(T^{\boldsymbol{x}'})^*(true)$. $\square$