# An Extension of Dynamic Logic for Modelling OCL's @*pre* Operator

Thomas Baar, Bernhard Beckert, and Peter H. Schmitt

Universität Karlsruhe, Fakultät für Informatik
Institut für Logik, Komplexität und Deduktionssysteme
Am Fasanengarten 5, D-76128 Karlsruhe
Fax: +49 721 608 4211, Email: {baar,beckert,pschmitt}@ira.uka.de

**Abstract.** We consider first-order Dynamic Logic (DL) with non-rigid functions, which can be used to model certain features of programming languages such as array variables and object attributes. We extend this logic by introducing an operator @*pre* on functions that makes a function after program execution refer to its old value before program execution. We show that formulas with this operator can be transformed into equivalent formulas of the non-extended logic. We briefly describe the motivation for this extension coming from a related operator in the Object Constraint Language (OCL).

## 1 Introduction

Since the Unified Modeling Language (UML) has been adopted as a standard of the Object Management Group (OMG) in 1997, many efforts have been made to underpin the UML—and the Object Constraint Language (OCL), which is an integral part of the UML, —with a formal semantics. Most approaches are based on providing a translation of UML/OCL into a language with a well-understood semantics, e.g., BOTL [3] and the Larch Shared Language (LSL) [4].

Within the KeY project (see the web site at i12www.ira.uka.de/~key for details), we follow the same line, translating UML/OCL into Dynamic Logic (DL). This choice is motivated by the fact that DL can cope with both the dynamic concepts of UML/OCL and real world programming languages used to implement UML models (e.g. Java Card [2]).

The OCL allows to enrich a UML model with additional constraints, e.g., invariants for UML classes, pre-/post-conditions for operations, guards for transitions in state-transition diagrams, etc. Although, at first glance, OCL is similar to an ordinary first-order language, closer inspection reveals some unusual concepts. Among them is the @*pre* operator. In OCL, this unary operator is applicable to attributes, associations, and side-effect-free operations (these are called "properties" in the OCL context [9, p. 7-11ff]). The @*pre* operator may only be used in post-conditions of UML operations. A property *prop* followed by @*pre* in the post-condition of an operation $m()$ evaluates to the value of *prop* before the execution of $m()$.

Dynamic Logic [5–8] can be seen as an extension of Hoare logic [1]. It is a first-order modal logic with modalities $[p]$ and $\langle p \rangle$ for every program $p$. These modalities refer to the worlds (called states in the DL framework) in which the program $p$ terminates when started in the current world. The formula $[p]\phi$ expresses that $\phi$ holds in *all* final states of $p$, and $\langle p \rangle \phi$ expresses that $\phi$ holds in *some* final state of $p$. In versions of DL with a non-deterministic programming language there can be several such final states (worlds). Here we use a deterministic *while* programming language. For deterministic programs there is exactly one final world (if $p$ terminates) or there is no final world (if $p$ does not terminate). The formula $\phi \rightarrow \langle p \rangle \psi$ is valid if, for every state $s$ satisfying pre-condition $\phi$, a run of the program $p$ starting in $s$ terminates, and in the terminating state the post-condition $\psi$ holds. The formula $\phi \rightarrow [p]\psi$ expresses the same, except that termination of $p$ is not required, i.e., $\psi$ must only hold *if* $p$ terminates. Thus, $\phi \rightarrow [p]\psi$ is similar to the Hoare triple $\{\phi\}p\{\psi\}$.

Here, we consider a version of first-order DL with non-rigid functions, i.e., functions whose interpretation can be changed by programs and, thus, can differ from state to state. Such non-rigid functions can be used to model features of real-world programming languages such as array variables and object attributes.

Moreover, to ease the translation of OCL into DL, we extend DL with an operator corresponding to OCL's @*pre*. The DL @*pre* operator makes a non-rigid function after program execution refer to its old value before program execution. This allows to easily express the relation between the old and the new interpretation. For example, $[p](c \doteq c^{@pre})$ expresses that the interpretation of the constant $c$ is not changed by the program $p$.

The main contribution of this paper is to show that formulas with the @*pre* operator can be transformed into equivalent formulas without @*pre*.

The @*pre* construct of OCL has already been investigated by other authors, e.g., for the purpose of translating OCL into BOTL [3]. However, to our knowledge, the work reported in this paper is the first treatment of @*pre* in the DL framework.

In Section 2, we briefly introduce DL with non-rigid functions. Section 3 extends DL with the @*pre* operator and gives its semantics formally. In Sections 4 resp. 5 we present two transformations of DL with @*pre* into DL without @*pre*. We close with a summary and a short discussion of possible extensions to non-deterministic programming languages in Section 7. The proofs are omitted from the main part of the paper; they are given in Appendix A.

## 2 Dynamic Logic with Non-rigid Functions

Although non-rigid functions are mostly ignored in the literature, the more specific concept of array assignments has been investigated in [5, 7]. In both papers their semantics is handled by adding to each state valuations of second-order array variables. We introduce, instead, non-rigid function symbols. This shift of attention comes naturally when we want to axiomatise the semantics of objectoriented languages in DL. In this setting non-static attributes of a class are best modelled by non-rigid functions.

Let $\Sigma = \Sigma_{nr} \cup \Sigma_r$ be a signature, where $\Sigma_{nr}$ contains the non-rigid function symbols and $\Sigma_r$ contains the rigid function symbols and the predicate symbols, which are all rigid ($\Sigma_r$ always contains the equality relation $\doteq$). The set $Term(\Sigma)$ of terms and the set $Fml_{FOL}(\Sigma)$ of first-order formulas are built as usual from $\Sigma$ and an infinite set $Var$ of object variables.

A term is called *non-rigid* if (a) it is a variable or (b) its leading function symbol is in $\Sigma_{nr}$. The programs in our DL are *while* programs with a generalised assignment command, reflecting the presence of non-rigid terms.

**Definition 1.** *The sets* $Fml_{DL}(\Sigma)$ *of* DL-formulas *and* $Prog_{DL}(\Sigma)$ *of* programs *are simultaneously defined as follows:*

- $Fml_{FOL}(\Sigma) \subseteq Fml_{DL}(\Sigma)$.
- *If* $\phi_1, \phi_2$ *are in* $Fml_{DL}(\Sigma)$*, then so are* $\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\phi_1 \rightarrow \phi_2$, $\forall x\, \phi_1$ *and* $\exists x\, \phi_1$ *for all* $x \in Var$.
- *If* $\phi$ *is in* $Fml_{DL}(\Sigma)$ *and* $p$ *is in* $Prog_{DL}(\Sigma)$*, then* $\langle p\rangle\phi$ *and* $[p]\phi$ *are in* $Fml_{DL}(\Sigma)$.
- *If* $t$ *is a non-rigid term and* $s$ *is a term, then* $t := s$ *is in* $Prog_{DL}(\Sigma)$.
- **fail** *and* **skip** *are in* $Prog_{DL}(\Sigma)$.
- *If* $p_1, p_2$ *are in* $Prog_{DL}(\Sigma)$*, then so is their sequential composition* $p_1; p_2$.
- *If* $\psi$ *is a quantifier-free first-order formula and* $p, q$ *are in* $Prog_{DL}(\Sigma)$*, then* **if** $\psi$ **then** $p$ **else** $q$ **fi** *and* **while** $\psi$ **do** $p$ **od** *are in* $Prog_{DL}(\Sigma)$.

In the following, we often do not differentiate between the modalities $\langle p\rangle$ and $[p]$, and we use $\{\!|p|\!\}$ to denote that it may be of either form.

The Kripke structures used to evaluate formulas from $Fml_{DL}(\Sigma)$ and programs from $Prog_{DL}(\Sigma)$ are called *DL-Kripke structures*. The set of states of a DL-Kripke structure $\mathcal{K}$ is obtained as follows: Let $\mathcal{A}_0$ be a fixed first-order structure for the rigid signature $\Sigma_r$, and let $A$ denote the universe of $\mathcal{A}_0$. An $n$-ary function symbol $f \in \Sigma_r$ is interpreted as a function $f^{\mathcal{A}_0} : A^n \rightarrow A$ and every $n$-ary relation symbol $r \in \Sigma_r$ is interpreted as a set $R^{\mathcal{A}_0} \subseteq A^n$ of $n$-tuples. A *variable assignment* is a function $u : Var \rightarrow A$. We use $u[x/b]$ (where $b \in A$ and $x \in Var$) to denote the variable assignment such that $u[x/b](y) = b$ if $x = y$ and $u[x/b](x) = u(y)$ otherwise; moreover, if $V$ is a set of variables, then $u_{|V}$ denotes the *restriction* of $u$ to $V$. The set $S$ of all *states* of $\mathcal{K}$ consists of all pairs $(\mathcal{A}, u)$, where $u$ is a variable assignment and $\mathcal{A}$ is a first-order structure for the signature $\Sigma$, whose reduction to $\Sigma_r$, denoted with $\mathcal{A}_{|\Sigma_r}$, coincides with $\mathcal{A}_0$. We are now ready to define for each program $p$ its interpretation $\rho(p)$, which is a relation on $S$. Simultaneously, we define when a formula $\phi$ is true in a state $(\mathcal{A}, u)$, denoted by $(\mathcal{A}, u) \models \phi$.

**Definition 2.** *The* interpretation $\rho(p)$ *of programs* $p$ *and the* relation $\models$ *between* $S$ *and* $Fml_{DL}(\Sigma)$ *are simultaneously defined as follows:*

1. $(\mathcal{A}, u) \models \phi$ *is defined as usual in classical logic if* $\phi$ *is an atomic formula or its principal logical operator is one of the classical operators* $\wedge$, $\vee$, $\rightarrow$, $\neg$, *or*

*one of the quantifiers* $\forall$, $\exists$. *Also, the evaluation* $t^{(\mathcal{A},u)}$ *of terms* $t$ *is defined as usual.*

2. $(\mathcal{A},u) \models \langle p\rangle\phi$ *iff there exists a pair* $((\mathcal{A},u),(\mathcal{B},w))$ *of states in* $\rho(p)$ *such that* $(\mathcal{B},w) \models \phi$.
3. $(\mathcal{A},u) \models [p]\phi$ *iff* $(\mathcal{B},w) \models \phi$ *for all pairs* $((\mathcal{A},u),(\mathcal{B},w))$ *of states in* $\rho(p)$.
4. *If* $x$ *is a variable, then* $\rho(x := s) = \{((\mathcal{A},u),(\mathcal{A},u[x/s^{(\mathcal{A},u)}])) \mid (\mathcal{A},u) \in S\}$.
5. *If* $t = f(t_1,\ldots,t_n)$ *is a non-rigid term, then* $\rho(t := s)$ *consists of all pairs* $((\mathcal{A},u),(\mathcal{B},u))$ *such that* $\mathcal{B}$ *coincides with* $\mathcal{A}$ *except for the interpretation of* $f$, *which is given by*

$$f^{\mathcal{B}}(b_1,\ldots,b_n) = \begin{cases} s^{(\mathcal{A},u)} & \textit{if } (b_1,\ldots,b_n) = (t_1^{(\mathcal{A},u)},\ldots,t_n^{(\mathcal{A},u)}) \\ f^{\mathcal{A}}(b_1,\ldots,b_n) & \textit{otherwise} \end{cases}$$

6. $\rho(\mathbf{skip}) = \{((\mathcal{A},u),(\mathcal{A},u)) \mid (\mathcal{A},u) \in S\}$, *and* $\rho(\mathbf{fail}) = \emptyset$.
7. $\rho(\mathbf{while}\,\psi\,\mathbf{do}\,p\,\mathbf{od})$ *and* $\rho(\mathbf{if}\,\psi\,\mathbf{then}\,p\,\mathbf{else}\,q\,\mathbf{fi})$ *are defined as usual, e.g. [7].*

The particular choice of programs in $Prog_{DL}(\Sigma)$ (Def. 1) is rather arbitrary. The results being proved in this paper hold true for any choice of $Prog_{DL}(\Sigma)$, as long as Lemma 1 is guaranteed. Furthermore, we assume that all programs $p$ are deterministic, i.e., $(s,s_1) \in \rho(p)$ and $(s,s_2) \in \rho(p)$ implies $s_1 = s_2$.

**Lemma 1.** *Let* $\mathcal{K} = (\mathcal{S},\rho)$ *be a DL-Kripke structure over a signature* $\Sigma$, *let* $p$ *be a program, and let* $V_p$ *be the set of all variables occurring in* $p$.

1. *The program* $p$ *only changes variables in* $V_p$; *that is, if* $u(x) \neq w(x)$ *then* $x \in V_p$ *for all* $((\mathcal{A},u),(\mathcal{B},w)) \in \rho(p)$.
2. *The domain of the relation* $\rho(p)$ *is closed under changing variables not in* $V_p$ *in the sense that, if* $((\mathcal{A},u),(\mathcal{B},w)) \in \rho(p)$ *and* $u'_{|V_p} = u_{|V_p}$, *then there is a pair* $((\mathcal{A},u'),(\mathcal{B},w')) \in \rho(p)$ *with* $w'_{|V_p} = w_{|V_p}$ *and* $u'_{|Var\setminus V_p} = w'_{|Var\setminus V_p}$.

## 3 Dynamic Logic with the Operator @*pre*

We now define syntax and semantics of DL extended with the @*pre* operator, which can be attached to non-rigid function symbols. Intuitively, the semantics of $f^{@pre}$ within the scope of a modal operator $\langle\!|p|\!\rangle$ is that of $f$ before execution of $p$. If a formula contains nested modal operators, it may not be clear, to which state the @*pre* operator refers. To avoid confusion, we only allow @*pre* to be used in the Hoare fragment of DL, where formulas contain at most one modal operator.

**Definition 3.** *The set* $Term^{@}(\Sigma)$ *of* extended terms *over* $\Sigma = \Sigma_r \cup \Sigma_{nr}$ *consists of all terms* $t^{@}$ *that can be constructed from some* $t \in Term(\Sigma)$ *by attaching* @*pre to arbitrarily many occurrences of function symbols from* $\Sigma_{nr}$ *in* $t$. *Accordingly, the set* $Form^{@}_{FOL}(\Sigma)$ *of* extended first-order formulas *over* $\Sigma$ *consists of all formulas* $\phi^{@}$ *that can be constructed from some* $\phi \in Fml_{FOL}(\Sigma)$ *by attaching* @*pre to arbitrarily many occurrences of function symbols from* $\Sigma_{nr}$ *in* $\phi$.

**Definition 4.** *The* Hoare fragment $H(\Sigma) \subset Fml_{DL}(\Sigma)$ *over a signature* $\Sigma$ *consists of all formulas of the form* $\forall z_1 \ldots \forall z_d(\phi \rightarrow [p]\psi)$ *where* $p \in Prog_{DL}(\Sigma)$, $\phi, \psi \in Fml_{FOL}(\Sigma)$ *and* $z_1, \ldots, z_d \in Var$ $(d \geq 0)$*.*

*The* extended Hoare fragment $H^{@}(\Sigma)$ *consists of all formulas of the form* $\forall z_1 \ldots \forall z_d(\phi \rightarrow [p]\psi)$ *with* $p \in Prog_{DL}(\Sigma)$, $\phi \in Fml_{FOL}(\Sigma)$, $\psi \in Fml^{@}_{FOL}(\Sigma)$ *and* $z_1, \ldots, z_d \in Var$ $(d \geq 0)$*.*

**Definition 5.** *Let* $\mathcal{K}$ *be a DL-Kripke structure, let* $(\phi \rightarrow \langle p\rangle\psi) \in H^{@}$*, and let* $(\mathcal{A}, u)$ *be a state of* $\mathcal{K}$*. The relation* $(\mathcal{A}, u) \models \phi \rightarrow [p]\psi$ *is defined in the same way as in Def. 2 for formulas without @pre, except that, for any pair* $((\mathcal{A}, u), (\mathcal{B}, w))$ *in* $\rho(p)$*, the interpretation* $t^{(\mathcal{B},w)}$ *of the non-rigid terms in* $Term^{@}(\Sigma)$ *is given by:*

$$(f^{@pre}(t_1, \ldots, t_n))^{(\mathcal{B},w)} \quad = \quad f^{\mathcal{A}}(t_1^{(\mathcal{B},w)}, \ldots, t_n^{(\mathcal{B},w)}) \ .$$

In the following, we use notation like $(\mathcal{B}, w) \models \phi$ and $t^{(\mathcal{B},w)}$ for formulas $\phi$ resp. terms $t$ containing the $@pre$ operator if it is clear from the context which structure $\mathcal{A}$ is to be used for the interpretation of $@pre$.

## 4 Eliminating @*pre* Using Additional Functions

After the pre-requisites we now define a translation function $\tau_f$ on the extended Hoare fragment that eliminates the $@pre$ operator (the subscript $f$ indicates that $\tau_f$ uses new function symbols). The idea of $\tau_f$ is to introduce, for each function $f_i$ that occurs with the $@pre$ operator, an associated new function symbol $f^i_{pre}$ and to ensure that $f^i_{pre}$ is interpreted in the right way. For example, the translation of $\langle p\rangle r(f_i^{@pre}(a))$ is $\forall x(f^i_{pre}(x) \doteq f_i(x)) \rightarrow \langle p\rangle r(f^i_{pre}(a))$ (a more complex example is shown in Section 6). This (rather naive) translation preserves universal validity of formulas (Theorem 1).

**Definition 6.** *Let* $\Sigma' = \Sigma'_r \cup \Sigma_{nr}$ *be an extension of the signature* $\Sigma$ *where* $\Sigma'_r = \Sigma_r \cup \Sigma_{pre}$ *and* $\Sigma_{pre}$ *is disjoint from* $\Sigma$ *and, for every* $f \in \Sigma_{nr}$*, contains a function symbol* $f_{pre}$ *of the same arity as* $f$*. Then, the result of applying the* translation $\tau_f : H^{@}(\Sigma) \longrightarrow H(\Sigma')$ *to some* $\pi = \forall z_1 \ldots \forall z_d(\phi \rightarrow [p]\psi)$ *is*

$$\forall z_1 \ldots \forall z_d \\ \big((\phi \wedge \textstyle\bigwedge_{i=1}^{k} \forall x^i_1 \ldots \forall x^i_{n_i} f^i_{pre}(x^i_1, \ldots, x^i_{n_i}) \doteq f_i(x^i_1, \ldots, x^i_{n_i})) \rightarrow [p]\psi'\big)$$

*where*

- $f_1, \ldots, f_k \in \Sigma_{nr}$ *are the function symbols occurring in* $\psi$ *with attached @pre,*
- $f^1_{pre}, \ldots, f^k_{pre}$ *are the corresponding function symbols in* $\Sigma_{pre}$*,*
- *the* $x^i_j$ *are pairwise distinct variables not occurring in the original formula* $\pi$*,*
- $\psi'$ *is the result of replacing all occurrences of* $f_i^{@pre}$ *in* $\psi$ *by* $f^i_{pre}$ $(1 \leq i \leq k)$*.*

**Theorem 1.** *Let* $\pi \in H^{@}(\Sigma)$*. Then,* $\models_{\Sigma} \pi$ *iff* $\models_{\Sigma'} \tau_f(\pi)$*.*

Note, that the practical consequences of Theorem 1 are rather limited. Assume that $\Gamma$ is a DL formula without free variables and $\pi = \forall z_1 \ldots \forall z_d(\phi \to \langle p \rangle \psi)$ is a formula in the Hoare fragment for which we want to prove that $\Gamma \models_\Sigma \pi$. Because of the deduction theorem, that is equivalent to $\models_\Sigma \Gamma \to \pi$. Now, we would like to apply our translation $\tau_f$ to transform $\Gamma \to \pi$ into a formula without @*pre* and, making use of Theorem 1, prove the resulting non-extended formula instead. The translation $\tau_f$, however, is only applicable if $\Gamma \to \pi$ is in the Hoare fragment, which requires $\Gamma$ to be a pure first-order formula. This problem is avoided with our second translation presented in the following section.

## 5 Eliminating @*pre* Without Using Additional Functions

The translation $\tau_v$ does not only preserve validity but leads to a formula that is fully equivalent to the original one. Instead of introducing new function symbols, it solely relies on introducing new variables.

The basic idea of $\tau_v$ is to "flatten" all terms in a formula containing @*pre*. For example, $[p]r(f^{@pre}(a))$ is equivalent to $[p]\forall y(y \doteq f^{@pre}(a) \to r(y))$. This in turn is equivalent to $[p]\forall y_1 \forall y_2((y_1 \doteq f^{@pre}(y_2) \wedge y_2 \doteq a) \to r(y_1))$. Since $y_1, y_2$ are new variables and do not occur in $p$, the quantification can be moved to the front, and we get $\forall y_1 \forall y_2 [p]((y_1 \doteq f^{@pre}(y_2) \wedge y_2 \doteq a) \to r(y_1))$. For the $\langle \cdot \rangle$ modality, this is only possible if the program $p$ is deterministic (cf. Section 7). Finally, we have arrived at a point where we can eliminate the occurrence of @*pre* by moving the "definition" $y_1 \doteq f^{@pre}(y_2)$ of $y_1$ in front of the modal operator: $\forall y_1 \forall y_2 (y_1 \doteq f(y_2) \to ([p](y_2 \doteq a \to r(y_1))))$. Note, that the "definition" $y_2 \doteq a$ of $y_2$ remains behind the modal operator because no @*pre* is attached to $a$.

The idea that has been illustrated with this small example is generalised in the following definition of the translation $\tau_v$ (a more complex example for the application of $\tau_v$ is shown in Section 6).

**Definition 7.** *The result of applying the* translation $\tau_v : H^{@}(\Sigma) \longrightarrow H(\Sigma)$ *to some formula* $\pi = \forall z_1 \ldots \forall z_d(\phi \to [p]\psi)$ *from* $H^{@}(\Sigma)$ *is defined as follows: Let* $t_1, \ldots, t_l, \ldots, t_m, \ldots, t_k \in Term^{@}(\Sigma)$ $(1 \leq l \leq m \leq k)$ *be all (sub-)terms occurring in* $\psi$, *where for* $1 \leq i \leq l$ *the term* $t_i = f_i^{@pre}(s_1^i, \ldots, s_{n_i}^i)$ *is not a variable and has the @pre operator attached to its leading function symbol, for* $l < i \leq m$ *the term* $t_i = f_i(s_1^i, \ldots, s_{n_i}^i)$ *is not a variable and does not have the @pre operator attached to its leading function symbol, and for* $m < i \leq k$ *the term* $t_i$ *is a variable. Then,*

$$\begin{aligned}\tau_v(\pi) \;=\; & \forall z_1 \ldots \forall z_d \forall y_1 \ldots \forall y_k \\ & \quad \Big( \big(\phi \wedge \textstyle\bigwedge_{i=1}^{l} y_i \doteq f_i(x_1^i, \ldots, x_{n_i}^i)\big) \;\to \\ & \qquad [p]\big((\textstyle\bigwedge_{i=l+1}^{m} y_i \doteq f_i(x_1^i, \ldots, x_{n_i}^i) \wedge \bigwedge_{i=m+1}^{k} y_i \doteq t_i) \;\to\; \psi'\big)\Big) \;,\end{aligned}$$ [1]

*where*

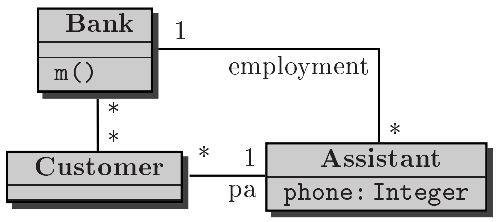
[1] If one of the variables $y_i$ occurs in $\tau_v(\pi)$ on only one side of $[p]$, then $\tau_v(\pi)$ can be simplified by omitting the equality "defining" $y_i$ and replacing all occurrences of $y_i$ by the right side of that equality.

– *for all* $1 \le i \le m$ *and* $1 \le j \le n_i$*, the variable* $x^i_j$ *is identical to* $y_{ind}$ *where* $ind \in \{1, \dots, k\}$ *is the index such that* $t_{ind} = s^i_j$*,*
– $\psi'$ *is the result of replacing all occurrences of terms* $t_i$ *in* $\psi$ *on the top-level (i.e., not the sub-term occurrences) by* $y_i$ *(*$1 \le i \le k$*).*

**Theorem 2.** *Let* $\pi \in H^{@}(\Sigma)$*. Then,* $\models \pi \leftrightarrow \tau_v(\pi)$*.*

Theorem 2 states the strongest result one could wish for. It implies that $\pi$ can be substituted by $\tau_v(\pi)$ in any context. However, $\tau_v$ is only defined on the Hoare fragment. To eliminate occurrences of *@pre* from more complex DL formulas, one has to translate the Hoare fragment sub-formulas. For instance, even if $\Gamma$ is not a pure first-order formula, $\Gamma \to \pi$ can be translated into $\Gamma \to \tau_v(\pi)$.

## 6 An Illustrating Example




The UML class diagram on the right models the following scenario: To better serve their customers, a bank names for every customer one of its employees as a personal assistant.

Now assume, the bank moves to a new building. The phone numbers may change and also the association of the customers with their personal assistants is reconsidered on this occasion. Operation $m()$ effects all these changes but must ensure that for every customer the phone number of his or her personal assistant does not change. In OCL this constraint is expressed as:

**context** *Bank::m()*
  **post:** `customer->forAll(c| c.pa.phone = c.pa@pre.phone@pre)`

By converting this constraint into extended DL, we get the following Hoare fragment formula, assuming that the program $p_m$ implements $m()$:

$$\pi \quad = \quad \forall z \big( customer(z) \to \langle p_m \rangle\, phone(pa(z)) \doteq phone^{@pre}(pa^{@pre}(z)) \big)$$

The application of $\tau_f$ resp. $\tau_v$ to $\pi$ yields:

$$\begin{aligned}
\tau_f(\pi) \quad = \quad & \forall z \big( \; ( \; customer(z) \land \\
& \qquad \forall x^1_1\, pa_{pre}(x^1_1) \doteq pa(x^1_1) \;\land\; \forall x^2_1\, phone_{pre}(x^2_1) \doteq phone(x^2_1) \\
& \quad ) \;\to \langle p_m \rangle\, phone(pa(z)) \doteq phone_{pre}(pa_{pre}(z)) \big) \\
\tau_v(\pi) \quad = \quad & \forall z \forall y_1 \dots \forall y_5 \big( \\
& \quad (customer(z) \land y_1 \doteq phone(y_2) \land y_2 \doteq pa(y_5)) \to \\
& \quad \langle p_m \rangle\, ((y_3 \doteq phone(y_4) \land y_4 \doteq pa(y_5) \land y_5 \doteq z) \;\to\; y_3 \doteq y_1) \big)
\end{aligned}$$

## 7 Summary

This paper demonstrates how the semantics of the OCL construct *@pre* can be integrated into an extended DL with non-rigid function symbols. Since the *@pre* operator is rather unusual, for practical reasons, it is useful to translate formulas with @*pre* into formulas without @*pre*. Our first translation $\tau_f$ only preserves validity of formulas, which in practice is often not sufficient. The second translation $\tau_v$ is more complex but leads to a fully equivalent formula. Both translations stay within the Hoare fragment, i.e., transform Hoare fragment formulas into Hoare fragment formulas. The translation $\tau_v$ can also be used to remove @*pre* from a non-Hoare formula $\pi$ by applying it to all Hoare sub-formulas of $\pi$.

Both translations are independent of the actual form of the program $p$ that is part of the translated formula; it remains unchanged and can be anonymous. Only the variables occurring in $p$ have to be known, as they may be affected by program execution.

The correctness proofs for $\tau_f$ and $\tau_v$ make use of the fact that the programs are deterministic. Nevertheless, we assert that the translation $\tau_f$ works just as well for non-deterministic programming languages. For $\tau_v$ the situation is more difficult. Intuitively, $\tau_v$ moves a universal quantification from behind the modal operator $\{\!|p|\!\}$ to the front of $\{\!|p|\!\}$. That is not a problem as long as the programs are deterministic. If the programs are non-deterministic, however, $\{\!|p|\!\}$ contains an implicit quantification over states. If $\{\!|p|\!\} = [p]$, that quantification is universal, and $\tau_v$ should still work. If, however, $\{\!|p|\!\} = \langle p \rangle$, the translation $\tau_v$ intuitively moves a universal quantification over an implicit existential quantification, which is not correct. Appendix B contains an example demonstrating that Theorem 2 (which states the correctness of $\tau_v$) does not hold for non-deterministic programs and the $\langle \cdot \rangle$ modality. Nevertheless, even if $p$ is non-deterministic, $\tau_v$ can be used to remove the @*pre* operator from a formula $\pi$ of the form $\phi \to \langle p \rangle \psi$ because $\pi$ is equivalent to $\phi \to \neg[p]\neg\psi$ and, thus, to $\phi \to \neg\tau_v(true \to [p]\neg\psi)$. Then, however, the resulting formula is not in the Hoare fragment.

## References


1. K. R. Apt. Ten years of Hoare logic: A survey – part I. *ACM Transactions on Programming Languages and Systems*, 1981.
2. B. Beckert. A Dynamic Logic for Java Card. In *Proceedings, 2nd ECOOP Workshop on Formal Techniques for Java Programs, Cannes, France*, 2000.
3. D. Distefano, J.-P. Katoen, and A. Rensink. Towards model checking OCL. In *Proceedings, ECOOP Workshop on Defining a Precise Semantics for UML*, 2000.
4. A. Hamie, J. Howse, and S. Kent. Interpreting the Object Constraint Language. In *Proceedings, Asia Pacific Conference in Software Engineering*. IEEE Press, 1998.
5. D. Harel. Dynamic Logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Volume II: Extensions of Classical Logic*. Reidel, 1984.
6. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.
7. D. Kozen and J. Tiuryn. Logic of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 14, pages 89–133. Elsevier, 1990.
8. V. R. Pratt. Semantical considerations on Floyd-Hoare logic. In *Proceedings, 18th Annual IEEE Symposium on Foundation of Computer Science*, 1977.
9. Rational Software Corp. et al. *Unified Modelling Language Semantics, version 1.3*, June 1999. Available at: `www.rational.com/uml/index.jtmpl`.

## A Appendix: Proofs for the Correctness Theorems

For the purpose of proving the theorems it is useful to introduce the notion of a restriction of a DL-Kripke structure:

**Definition 8.** *Let $\mathcal{K}' = (\mathcal{S}', \rho')$ be a DL-Kripke structure for a signature $\Sigma'$. Let $\Sigma$ be a signature obtained from $\Sigma'$ by omitting some rigid function symbols. We define the* restriction $\mathcal{K}'_{|\Sigma} = \mathcal{K} = (\mathcal{S}, \rho)$ *of $\mathcal{K}'$ to $\Sigma$:*

1. $\mathcal{S} = \{(\mathcal{A}'_{|\Sigma}, u) \mid (\mathcal{A}', u) \in \mathcal{S}'\}$
2. $\rho(p) = \{((\mathcal{A}'_{|\Sigma}, u), (\mathcal{B}'_{|\Sigma}, w)) \mid ((\mathcal{A}', u), (\mathcal{B}', w)) \in \rho'(p)\}$

**Lemma 2.** *We use the notation from Definition 6. For all (sub-)terms $t$ of $\psi$, let $\tau_f^{Term}(t)$ be the result of replacing all occurrences of $f_i^{@pre}$ in $t$ by $f_{pre}^i$ $(1 \leq i \leq k)$.*

*Moreover, let $\mathcal{K}' = (\mathcal{S}', \rho')$ be a Kripke structure over $\Sigma'$, let $(\mathcal{A}', u)$ and $(\mathcal{B}', w)$ be states of $\mathcal{K}'$, and let $p \in Prog_{DL}(\Sigma)$ be a program.*

*Then, for all (sub-)terms $t$ of $\psi$, the following holds: If*

1. $(\mathcal{A}', u) \models_{\Sigma'} \forall x_1^i \ldots \forall x_{n_i}^i f_{pre}^i(x_1^i, \ldots, x_{n_i}^i) \doteq f(x_1^i, \ldots, x_{n_i}^i)$ *for every non-rigid function symbol $f_i$ that occurs in $t$ with attached @pre, and*
2. $((\mathcal{A}', u), (\mathcal{B}', w)) \in \rho'(p)$,

*then*

$$(\tau_f^{Term}(t))^{(\mathcal{B}', w)} = t^{(\mathcal{B}, w)}$$

*where $\mathcal{B} = \mathcal{B}'_{|\Sigma}$.*

*Proof.* First, we derive from clause 1 that $(f_{pre}^i)^{\mathcal{A}'} = (f^i)^{\mathcal{A}'}$. By clause 2 and applying Definition 5, we obtain

$$(f_i^{@pre})^{\mathcal{B}'} = (f_{pre}^i)^{\mathcal{A}'} \tag{1}$$

Now, the proof proceeds by induction on the complexity of the term $t$.

*Induction base.* In the base case, $t$ is of form $c$ or $c^{@pre}$ where $c \in \Sigma$ is a constant symbol or of form $x$ where $x \in Var$.

*Case 1: $t = x$.*

$$(\tau_f^{Term}(x))^{(\mathcal{B}', w)} = x^{(\mathcal{B}', w)} = x^{(\mathcal{B}, w)} = w(x)$$

*Case 2: $t = c$.*

$$(\tau_f^{Term}(c))^{(\mathcal{B}', w)} = c^{\mathcal{B}'} = c^{\mathcal{B}}$$

Remember, that $c \in \Sigma$ and $\mathcal{B} = \mathcal{B}'_{|\Sigma}$.

*Case 3:* $t = c^{@pre}$.

$$\begin{aligned}
(\tau_f^{Term}(c^{@pre}))^{(\mathcal{B}',w)} &= c_{pre}^{\mathcal{B}'} \\
&= c_{pre}^{\mathcal{A}'} && \text{as } c_{pre} \in \Sigma_{pre} \text{ is rigid} \\
&= (c^{@pre})^{\mathcal{B}'} && \text{by (1)} \\
&= (c^{@pre})^{\mathcal{B}} && \text{as } c \in \Sigma \text{ and } \mathcal{B} = \mathcal{B}'_{|\Sigma} \\
&= (c^{@pre})^{(\mathcal{B},w)}
\end{aligned}$$

*Induction step.*
*Case 1:* $t = f(t_1, \dots, t_n)$. Trivial, by applying the definition of $\tau_f^{Term}$.
*Case 2* $(t = f^{@pre}(s_1, \dots, s_n)$.

$$\begin{aligned}
&(\tau_f^{Term}(f^{@pre}(s_1, \dots, s_n)))^{(\mathcal{B}',w)} \\
&= (f_{pre}(\tau_f^{Term}(s_1), \dots, \tau_f^{Term}(s_n)))^{(\mathcal{B}',w)} \\
&= f_{pre}^{\mathcal{B}'}((\tau_f^{Term}(s_1))^{(\mathcal{B}',w)}, \dots, (\tau_f^{Term}(s_n))^{(\mathcal{B}',w)}) \\
&= f_{pre}^{\mathcal{B}'}(s_1^{(\mathcal{B},w)}, \dots, s_n^{(\mathcal{B},w)}) \quad \text{by the induction hypothesis} \\
&= f_{pre}^{\mathcal{A}'}(s_1^{(\mathcal{B},w)}, \dots, s_n^{(\mathcal{B},w)}) \quad \text{as } f_{pre} \in \Sigma_{pre} \text{ is rigid} \\
&= (f^{@pre})^{\mathcal{B}'}(s_1^{(\mathcal{B},w)}, \dots, s_n^{(\mathcal{B},w)}) \quad \text{by (1)} \\
&= (f^{@pre})^{\mathcal{B}}(s_1^{(\mathcal{B},w)}, \dots, s_n^{(\mathcal{B},w)}) \quad \text{as } f \in \Sigma \text{ and } \mathcal{B} = \mathcal{B}'_{|\Sigma} \\
&= f^{@pre}(s_1, \dots, s_n)^{(\mathcal{B},w)}
\end{aligned}$$

□

**Lemma 3.** *We use the same notation as in Definition 6 and Lemma 2; and we assume that the same pre-conditions as in Lemma 2 are true. Then*

$$(\mathcal{B}', w') \models_{\Sigma'} \psi' \quad \textit{iff} \quad (\mathcal{B}, w) \models_{\Sigma} \psi \quad .$$

*Proof.* Simple, by applying the definition of $\psi'$ and Lemma 2. □

**Theorem 1.** *Let* $\pi \in H^{@}(\Sigma)$. *Then*

$$\models_{\Sigma} \pi \quad \textit{iff} \quad \models_{\Sigma'} \tau_f(\pi)$$

*Proof.* We use the same notation as in Definition 6. Since the variables $z_1, \dots, z_d$ are universally quantified in both $\pi$ and $\tau_f(\pi)$, it suffices to show that

$$\models_{\Sigma} \phi \to \{\!|p|\!\}\psi \quad \text{iff} \quad \models_{\Sigma'} (\phi \wedge \mathit{prefix}) \to \{\!|p|\!\}\psi'$$

where

$$\mathit{prefix} \ = \ \bigwedge_{i=1}^{k} \forall x_1^i \dots \forall x_{n_i}^i f_{pre}^i(x_1^i, \dots, x_{n_i}^i) \doteq f_i(x_1^i, \dots, x_{n_i}^i)$$

Let $\mathcal{K} = (\mathcal{S}, \rho)$ be a DL-Kripke structure for the signature $\Sigma$, and let $(\mathcal{A}, u)$ and $(\mathcal{B}, w)$ be states in $\mathcal{S}$. Analogous definitions are made for $\mathcal{K}'$.

*First part.* We assume

$$\models_{\Sigma} \phi \to \{\!|p|\!\}\psi$$

and aim at showing

$$(\mathcal{A}', u) \models_{\Sigma'} (\phi \wedge \mathit{prefix}) \to \{\!|p|\!\}\psi' \ .$$

The argument is only non-trivial if

$$(\mathcal{A}', u) \models_{\Sigma'} \phi \wedge \mathit{prefix} \ . \tag{1}$$

It remains to be shown that

$$(\mathcal{A}', u) \models_{\Sigma'} \{\!|p|\!\}\psi' \ . \tag{2}$$

Since $\models_{\Sigma} \phi \to \{\!|p|\!\}\psi$, we have in particular for $\mathcal{K} = \mathcal{K}'_{|\Sigma}$ and $\mathcal{A} = \mathcal{A}'_{|\Sigma}$ that

$$(\mathcal{A}, u) \models_{\Sigma} \phi \to \{\!|p|\!\}\psi \ . \tag{3}$$

By construction and (1), $(\mathcal{A}, u) \models_{\Sigma} \phi$. Thus, by (3),

$$(\mathcal{A}, u) \models_{\Sigma} \{\!|p|\!\}\psi \ . \tag{4}$$

*Case 1: The program $p$ does not terminate when started in $(\mathcal{A}, u)$.* In this case, the only way that (4) can hold is that $\{\!|p|\!\} = [p]$. Since $(\mathcal{A}, u)$ and $(\mathcal{A}', u)$ only differ in the interpretation of symbols that do not occur in $p$, the program $p$ does also not terminate when started in $(\mathcal{A}', u)$. Therefore, (2) holds.

*Case 2: The program $p$ terminates when started in $(\mathcal{A}, u)$.* Because our programming language is deterministic, there is exactly one state $(\mathcal{B}, w))$ with $((\mathcal{A}, u), (\mathcal{B}, w)) \in \rho(p)$ and

$$(\mathcal{B}, w) \models_{\Sigma} \psi \ . \tag{5}$$

By Definition 8, Clause 2, there exists a $\mathcal{B}'$ such that

$$\mathcal{B} = \mathcal{B}'_{|\Sigma} \quad \text{and} \quad ((\mathcal{A}', u), (\mathcal{B}', w)) \in \rho'(p) \ .$$

Lemma 3 and (5) yield

$$(\mathcal{B}', w) \models_{\Sigma'} \psi'$$

which finally proves (2).

*Second part.* We assume

$$\models_{\Sigma'} (\phi \wedge \mathit{prefix}) \to \{\!|p|\!\}\psi' \tag{6}$$

and aim at showing

$$(\mathcal{A}, u) \models_{\Sigma} \phi \to \{\!|p|\!\}\psi \ .$$

The argument is only non-trivial if

$$(\mathcal{A}, u) \models_\Sigma \phi \ . \tag{7}$$

It remains to be shown that

$$(\mathcal{A}, u) \models_\Sigma \{\!|p|\!\}\psi \ . \tag{8}$$

Since (6), we have for every $\mathcal{K}'$ and $(\mathcal{A}', u)$ that

$$(\mathcal{A}', u) \models_{\Sigma'} (\phi \wedge \mathit{prefix}) \to \{\!|p|\!\}\psi' \ . \tag{9}$$

We choose $\mathcal{K}'$ and $(\mathcal{A}', u)$ in such a way that

$$\mathcal{K}'_{|\Sigma} = \mathcal{K} \ , \quad \mathcal{A}'_{|\Sigma} = \mathcal{A} \ , \quad (f^i_{pre})^{\mathcal{A}'} = f^{\mathcal{A}'} \quad (1 \leq i \leq k) \ .$$

Lemma 1 implies that that this choice is possible. Thus,

$$(\mathcal{A}', u) \models_{\Sigma'} \phi \wedge \mathit{prefix} \ ,$$

and by (9) we get

$$(\mathcal{A}', u) \models_{\Sigma'} \{\!|p|\!\}\psi' \ . \tag{10}$$

*Case 1: The program $p$ does not terminate when started in* $(\mathcal{A}', u)$. In this case, the only way that (10) can hold is that $\{\!|p|\!\} = [p]$. Since $(\mathcal{A}, u)$ and $(\mathcal{A}', u)$ only differ in the interpretation of symbols that do not occur in $p$, the program $p$ does also not terminate when started in $(\mathcal{A}, u)$. Therefore, (8) holds.

*Case 2: The program $p$ terminates when started in* $(\mathcal{A}', u)$. Because our programming language is deterministic, there is exactly one state $(\mathcal{B}', w)$ with $((\mathcal{A}', u), (\mathcal{B}', w)) \in \rho'(p)$ and

$$(\mathcal{B}', w) \models_{\Sigma'} \psi' \ . \tag{11}$$

Thus, $((\mathcal{A}, u), (\mathcal{B}, w)) \in \rho(p)$ by the choice of $\mathcal{K}'$ (see Def. 8, Clause 2, where $\mathcal{B} = \mathcal{B}'_{|\Sigma}$). Lemma 3 and (11) yield

$$(\mathcal{B}, w) \models_\Sigma \psi \ .$$

which finally proves (8). □

**Theorem 2.** *Let $\pi \in H^{@}(\Sigma)$. Then $\models \pi \leftrightarrow \tau_v(\pi)$.*

*Proof.* We use the same notation as in Definition 7. Since the variables $z_1, \ldots, z_d$ are universally quantified in both $\pi$ and $\tau_v(\pi)$, it suffices to show that

$$\models \ (\phi \to \{\!|p|\!\}\psi) \ \leftrightarrow \ \forall y_1 \ldots \forall y_k(\mathit{prefix}_1 \to \{\!|p|\!\}(\mathit{prefix}_2 \to \psi'))$$

where

$$\mathit{prefix}_1 = \phi \wedge \bigwedge_{i=1}^{l} y_i \doteq f_i(x^i_1, \ldots, x^i_{n_i}))$$

$$\mathit{prefix}_2 = \bigwedge_{i=l+1}^{m} y_i \doteq f_i(x^i_1, \ldots, x^i_{n_i}) \wedge \bigwedge_{i=m+1}^{k} y_i \doteq t_i$$

*First part.* We first consider the easier implication from left to right.

Let $\mathcal{K} = (\mathcal{S}, \rho)$ be a Kripke structure for $\Sigma$, and let $(\mathcal{A}, u) \in \mathcal{S}$ such that

$$(\mathcal{A}, u) \models \phi \to \{p\}\psi \tag{1}$$

We have to show that

$$(\mathcal{A}, u) \models \forall y_1 \ldots \forall y_k(\mathit{prefix}_1 \to \{p\}(\mathit{prefix}_2 \to \psi')) \ .$$

Let $a_1, \ldots, a_k$ arbitrary elements of the universe of $\mathcal{A}$, and let

$$u' = u[y_1/a_1, \ldots, y_k/a_k] \ .$$

Now, it suffices to show that

$$(\mathcal{A}, u') \models \mathit{prefix}_1 \to \{p\}(\mathit{prefix}_2 \to \psi') \ .$$

The argument is only non-trivial if

$$(\mathcal{A}, u') \models \mathit{prefix}_1 \ . \tag{2}$$

We therefore obtain

$$(\mathcal{A}, u) \models \{p\}\psi \tag{3}$$

using (1) and the fact that $y_1, \ldots, y_k$ do not occur in $\phi$.

*Case 1: The program $p$ does not terminate when started in $(\mathcal{A}, u)$.* In this case, the only way that (3) can hold is that $\{p\} = [p]$. Since $u$ and $u'$ only differ in the assignment of values to variables not occurring in $p$, the program $p$ does also not terminate when started in $(\mathcal{A}, u')$. Therefore,

$$(\mathcal{A}, u') \models \{p\}(\mathit{prefix}_2 \to \psi')$$

holds trivially.

*Case 2: The program $p$ terminates when started in $(\mathcal{A}, u)$.* Because our programming language is deterministic, there is exactly one state $(\mathcal{B}, w)$ with

$$((\mathcal{A}, u), (\mathcal{B}, w)) \in \rho(p)$$

and

$$(\mathcal{B}, w) \models \psi \tag{4}$$

Since $y_1, \dots, y_k$ do not occur in $p$ that implies

$$(\mathcal{B}, w') \models \psi \tag{5}$$

where $w' = w[y_1/a_1, \dots, y_k/a_k]$, and $((\mathcal{A}, u'), (\mathcal{B}, w')) \in \rho(p)$. It remains to be shown that

$$(\mathcal{B}, w') \models \mathit{prefix}_2 \rightarrow \psi' \ .$$

Again, the argument is only non-trivial in case

$$(\mathcal{B}, w') \models \mathit{prefix}_2 \ . \tag{6}$$

Now, by (5) and the definition of $\psi'$, it suffices to prove

$$t_i^{(\mathcal{B}, w')} = y_i^{(\mathcal{B}, w')} \qquad \text{for } 1 \leq i \leq k \ . \tag{7}$$

We prove (7) by induction on the complexity of $t_i$.
*Case 1: $t_i$ is a variable.* We get from (6) that

$$(\mathcal{B}, w') \models y_i \doteq t_i \ .$$

Therefore, (7) holds trivially.
*Case 2: $t_i = f(s_1^i, \dots, s_{n_i}^i)$.* We get from (6) that

$$(\mathcal{B}, w') \models y_i \doteq f_i(x_1^i, \dots, x_{n_i}^i) \ .$$

Thus, it suffices to show that

$$(s_j^i)^{(\mathcal{B}, w')} = (x_j^i)^{(\mathcal{B}, w')} \qquad \text{for } 1 \leq j \leq n_i \ . \tag{8}$$

Because $x_j^i = y_{ind}$ and $s_j^i = t_{ind}$ for some $1 \leq ind \leq k$, (8) follows from the induction hypothesis as $s_j^i = t_{ind}$ is of lesser complexity than $t_i$.
*Case 3: $t_i = f_i^{@pre}(s_1^i, \dots, s_{n_i}^i)$.* In this case, we get from (2) that

$$(\mathcal{A}, u') \models y_i \doteq f_i(x_1^i, \dots, x_{n_i}^i) \ .$$

Since

$$(f_i)^{\mathcal{A}} = (f_i^{@pre})^{\mathcal{B}}$$

and, by Lemma 1, the variable assignments $u'$ and $w'$ do not differ in the valuation of the variables $y_i$ resp. $x_i^j$ (recall that each $x_i^j$ is identical to some $y_{i'}$), it again suffices to show (8), which can be done in the same way as in Case 2.

*Second part.* Now, we consider the implication from right to left.

We assume

$$(\mathcal{A}, u) \models \tau_v(\pi) \tag{9}$$

and aim to show

$$(\mathcal{A}, u) \models \pi \ .$$

Thus, we assume

$$(\mathcal{A}, u) \models \phi \tag{10}$$

and aim to prove

$$(\mathcal{A}, u) \models \{\!|p|\!\}\psi \ . \tag{11}$$

*Case 1: The program $p$ does not terminate when started in state $(\mathcal{A}, u)$.* Since (a) each of the variables $y_1, \ldots, y_l$ occurs in $\mathit{prefix}_1$ exactly once on the left side of an equation $y_i \doteq f_i(x_1^i, \ldots, x_{n_i}^i)$, and (b) if one of the argument variables $x_j^i$ is identical to some $y_{i'}$ then the term $t_{i'}$ is of lesser complexity than $t_i$, it is possible to inductively define a variable assignment $u'$ that differs from $u$ only on the variables $y_1, \ldots, y_l$ and has the property that

$$(\mathcal{A}, u') \models \bigwedge_{i=1}^{l} y_i \doteq f_i(x_1^i, \ldots, x_{n_i}^i)$$

Since $u$ and $u'$ differ only on the variables $y_1, \ldots, y_l$, which do not occur in $\phi$, we also have

$$(\mathcal{A}, u') \models \phi$$

and, thus,

$$(\mathcal{A}, u') \models \mathit{prefix}_1 \ .$$

From this and (9) we obtain

$$(\mathcal{A}, u') \models \{\!|p|\!\}(\mathit{prefix}_2 \to \psi') \ . \tag{12}$$

Now, since the variables $y_1, \ldots, y_k$ do not occur in $p$, the program $p$ does also not terminate when started in $u'$. Only if $\{\!|p|\!\} = [p]$ does this not contradict (12). Then, however, (11) holds trivially.

*Case 2: The program $p$ terminates when started in state $(\mathcal{A}, u)$.* Because our programming language is deterministic, there is a single state $(\mathcal{B}, w)$ with

$$((\mathcal{A}, u), (\mathcal{B}, w)) \in \rho(p) \ .$$

As in the previous case, we can inductively define values for the variables in $Y = \{y_1, \ldots, y_k\}$ in such a way that we get variable assignments $u'$ and $w'$ with

$$u_{|\,Var \setminus Y} = u'_{|\,Var \setminus Y} \quad \text{and} \quad w_{|\,Var \setminus Y} = w'_{|\,Var \setminus Y} \tag{13}$$

$$u'_{|\,Y} = w'_{|\,Y} \tag{14}$$

$$((\mathcal{A}, u'), (\mathcal{B}, w')) \in \rho(p) \tag{15}$$

$$(\mathcal{A}, u') \models \textstyle\bigwedge_{i=1}^{l} y_i \doteq f_i(x_1^i, \ldots, x_{n_i}^i) \tag{16}$$

$$(\mathcal{B}, w') \models \textstyle\bigwedge_{i=l+1}^{m} y_i \doteq f_i(x_1^i, \ldots, x_{n_i}^i) \wedge \textstyle\bigwedge_{i=m+1}^{k} y_i \doteq t_i \tag{17}$$

Moreover, since $u$ and $u'$ differ only on the variables $y_1, \ldots, y_k$, which do not occur in $\phi$, we have

$$(\mathcal{A}, u') \models \phi \tag{18}$$

Therefore, and by (16) we have

$$(\mathcal{A}, u') \models \mathit{prefix}_1 \tag{19}$$

By (19) and (9) we obtain

$$(\mathcal{A}, u') \models \{p\}(\mathit{prefix}_2 \rightarrow \psi')$$

and, thus, by (15)

$$(\mathcal{B}, w') \models \mathit{prefix}_2 \rightarrow \psi' \ ,$$

which finally with (17) gives us

$$(\mathcal{B}, w') \models \psi' \ . \tag{20}$$

By induction on the complexity of the term $t_i$, we prove that

$$y_i^{(\mathcal{B}, w')} = t_i^{(\mathcal{B}, w')} \tag{21}$$

*Case 1: $t_i$ is a variable.* We get from (17) that

$$(\mathcal{B}, w') \models y_i \doteq t_i \ ,$$

which trivially implies (21).
*Case 2: $t_i = f_i(s_1^i, \ldots, s_{n_i}^i)$.* We get from (17) that

$$(\mathcal{B}, w') \models y_i \doteq f_i(x_1^i, \ldots, x_{n_i}^i) \ .$$

Thus, it suffices to show that

$$(s_j^i)^{(\mathcal{B}, w')} = (x_j^i)^{(\mathcal{B}, w')} \qquad \text{for } 1 \leq j \leq n_i \ . \tag{22}$$

Because $x_j^i = y_{ind}$ and $s_j^i = t_{ind}$ for some $1 \leq ind \leq k$, (22) follows from the induction hypothesis as $s_j^i = t_{ind}$ is of lesser complexity than $t_i$.
*Case 3: $t_i = f_i^{@pre}(s_1^i, \ldots, s_{n_i}^i)$.* In this case, we get from (16) that

$$(\mathcal{A}, u') \models y_i \doteq f_i(x_1^i, \ldots, x_{n_i}^i) \ .$$

Since

$$(f_i)^{\mathcal{A}} = (f_i^{@pre})^{\mathcal{B}}$$

and $u'$ and $w'$ do not differ in the valuation of the variables $y_i$ resp. $x_i^j$ (recall that each $x_i^j$ is identical to some $y_{i'}$), it again suffices to show (22), which can be done in the same way as in Case 2.

Now, since $w$ and $w'$ only differ on the variables $y_1, \ldots, y_k$, which do not occur in $t_i$, we get from (21) that

$$y_i^{(\mathcal{B},w')} = t_i^{(\mathcal{B},w)} \tag{23}$$

By (20) and (23) and the construction of $\psi'$ from $\psi$, we get

$$(\mathcal{B}, w) \models \psi \ .$$

Therefore, and since $((\mathcal{A}, u), (\mathcal{B}, w)) \in \rho(p)$, (11) holds.

## B Appendix: Counterexample

We present a counterexample to Theorem 2 for non-deterministic programs and the modal operator $\langle\cdot\rangle$. Consider the formula

$$\pi \quad = \quad \mathit{true} \to \langle p \rangle r(f^{@pre}(c))$$

over the signature $\Sigma = \{r, \doteq, c\} \cup \{f\}$. The result of applying the translation $\tau_v$ is:

$$\tau_v(\pi) \ = \ \forall y_1 \forall y_2((\mathit{true} \wedge y_1 \doteq f(y_2)) \to \langle p \rangle (y_2 \doteq c \to r(y_1))) \ .$$

We consider the DL-Kripke structure $\mathcal{K} = (\mathcal{S}, \rho)$, an arbitrary state $(\mathcal{A}, u)$, and stipulate that $\{a_1, a_2\}$ is the universe of $\mathcal{A}$. Furthermore, the relation symbol $r$ is interpreted by the empty set in every state. Thus,

$$(\mathcal{A}, u) \not\models \pi \ .$$

Let $p$ be a program that, when started in $(\mathcal{A}, u)$, non-deterministically changes the interpretation of $c$ to $a_1$ or $a_2$ and does not change the state in any other way. Thus, there are states $(\mathcal{B}_1, u)$ and $(\mathcal{B}_2, u)$ in $\mathcal{S}$ such that $((\mathcal{A}, u), (\mathcal{B}_i, u)) \in \rho(p)$ for $i = 1, 2$. We stipulate $c^{\mathcal{B}_1} = a_1$ and $c^{\mathcal{B}_2} = a_2$. The interpretation of the function symbol $f$ does not matter, say $f^{\mathcal{B}_1} = f^{\mathcal{B}_2}$ is the identity function.

Now, for every $a \in \{a_1, a_2\}$, the condition $y_2 \doteq c$ is false in one of the states $(\mathcal{B}_1, u[y_2/a])$ and $(\mathcal{B}_2, u[y_2/a])$. Thus, $y_2 \doteq c \to r(y_1)$ is true in one of the two states, and so $\langle p \rangle (y_2 \doteq c \to r(y_1))$ is true in $(\mathcal{A}, u[y_2/a])$ (as $y_2$ is not affected by $p$). Therefore, $(\mathcal{A}, u) \models \tau_v(\pi)$, contradicting Theorem 2.


### Acknowledgement

We thank Wolfgang Ahrendt for fruitful discussions on the topic of this paper.