Reasoning on Procedural Programs using Description Logics with Concrete Domains

Ronald de Haan

Technische Universität Wien

Abstract. Existing approaches to assigning semantics to procedural programming languages do not easily allow automatic reasoning over programs. We assign a model theoretic semantics to programs of a simple procedural language, by encoding them into description logics with concrete domains. This allows us to flexibly express several reasoning problems over procedural programs, and to solve them efficiently using existing reasoning algorithms, for certain fragments of the programming language. Furthermore, it allows us to explore for what further fragments of the programming language reasoning problems are decidable.

1 Introduction

There has been much research investigating the formal semantics of (procedural) programs. This has resulted in several different approaches (see for instance [5, 6]). Such approaches include defining several notions of semantics for programs (e.g. operational semantics, denotational semantics, axiomatic semantics) and investigating the relations between these different notions. These approaches to investigating the semantics of programming languages are very useful for proving the correctness of compilers, for instance. However, it is very difficult to automatically solve reasoning problems on procedural programs (e.g. deciding whether two programs are equivalent) using such kinds of semantics. Also, there is no unified approach to express and automatically solve various different reasoning problems, based on these existing notions of semantics of programming languages.

We propose an approach that solves these problems. We assign a model theoretic semantics to procedural programs. In particular, we will encode procedural programs as description logic knowledge bases. Description logics are widely used formalisms to reason about large and complex knowledge bases [1], and there are many efficient reasoners available for description logics. We will show that by means of this encoding of programs into description logic knowledge bases, we can express reasoning problems over programs in the description logic language, and in this way reduce such reasoning problems to description logic reasoning. This allows us to leverage the performance of existing reasoning algorithms for our reasoning problems. Furthermore, we illustrate how this approach can be helpful in identifying fragments of programming languages for which these reasoning problems are decidable, and analyzing the computational complexity of such reasoning.

The paper is structured as follows. We begin with briefly repeating how the syntax and the semantics of the prototypical description logic $\mathcal{ALC}$ can be extended with concrete domains, resulting in the description logic $\mathcal{ALC}(\mathcal{D})$, before defining a simple

R.K. Rendsvig and S. Katrenko (Eds.): *ESSLLI 2012 Student Session Proceedings*, CEUR Workshop Proceedings vol.: see `http://ceur-ws.org/`, 2012, pp. 32–42.

representative procedural programming language *While*. Then, we show how programs of this programming language can be encoded into the description logic $\mathcal{ALC}(\mathcal{D})$ in a semantically faithful way. Also, we illustrate how this allows us to encode reasoning problems over *While* programs into $\mathcal{ALC}(\mathcal{D})$ reasoning problems. Finally, we will discuss the benefit of this method, in combination with suggestions for further research.

2 Preliminaries

A concrete domain $\mathcal{D}$ is a pair $(\Delta_\mathcal{D}, \Phi_\mathcal{D})$, where $\Delta_\mathcal{D}$ is a set and $\Phi_\mathcal{D}$ a set of predicate names. Each predicate name $P \in \Phi_\mathcal{D}$ is associated with an arity n and an n-ary predicate $P^\mathcal{D} \subseteq \Delta_\mathcal{D}^n$. A predicate conjunction of the form

$$c = \bigwedge_{i \leq k} (x_0^{(i)}, \ldots, x_{n_i}^{(i)}) : P_i,$$

where P_i is an n_i-ary predicate, for all $i \leq k$, and the $x_j^{(i)}$ are variables, is *satisfiable* iff there exists a function δ mapping the variables in c to elements of $\Delta_\mathcal{D}$ such that $(\delta(x_0^{(i)}), \ldots, \delta(x_{n_i}^{(i)})) \in P_i^\mathcal{D}$, for all $i \leq k$. A concrete domain is called *admissible* iff its set of predicate names is closed under negation and contains a name $\top_\mathcal{D}$ for $\Delta_\mathcal{D}$, and the satisfiability problem for finite conjunctions of predicates is decidable.

An example of an admissible concrete domain is $N = (\mathbb{N}, \Phi_N)$, where Φ_N contains unary predicates $\top_N$ and ρ_n, and binary predicates ρ, for $\rho \in \{=, \neq, <, \leq, >, \geq\}$, and ternary predicates $+, -, \star, \backslash$, and the required negations of predicates. All predicates are given the usual interpretation (here ρ_n holds for a value x iff $x \ \rho \ n$ holds).

We will use the description logic $\mathcal{ALC}(\mathcal{D})$ [2], which is the extension of the prototypical description logic $\mathcal{ALC}$ [7] with concrete domains. For an overview of description logics with concrete domains, see [3].

We get the logic $\mathcal{ALC}(\mathcal{D})$, for a given concrete domain, by augmenting $\mathcal{ALC}$ with *abstract features* (roles interpreted as functional relations), *concrete features* (interpreted as a partial function from the logical domain into the concrete domain), and a new concept constructor that allows to describe constraints on concrete values using predicates from the concrete domain. More concretely, we can construct concepts $\exists u_1, \ldots, u_k.P$ and $u\uparrow$, for $u, u_1, \ldots, u_k$ paths and $P \in \Psi^\mathcal{D}$ a k-ary predicate. A path is a sequence $f_1 \ldots f_n g$, where $f_1, \ldots, f_n$ (for $n \geq 0$) are abstract features and g is a concrete feature.

Paths are interpreted as (partial) functions from the logical domain into the concrete domain, by taking the composition of the interpretation of their components. Concepts $\exists u_1, \ldots, u_k.P$ are interpreted as the set of objects that are in the domain of the interpretation of all u_i, such that the resulting concrete objects satisfy the predicate P. Concepts $u\uparrow$ are interpreted as those objects that are not in the domain of the interpretation of u. For a complete, formal definition of the syntax and semantics of $\mathcal{ALC}(\mathcal{D})$, see [2, 3], for instance.

3 The Programming Language *While*

3.1 Syntax

We define the syntax of the simple representative procedural programming language *While* (defined and used for similar purposes in [5, 6]) with the following grammar

(we use right-associative bracketing). Let $\mathfrak{X}$ be a countably infinite set of variables. We let n range over values in $\mathbb{N}$[1], x over $\mathfrak{X}$, a over expressions of category **AExp**, b over expressions of category **BExp**, and p over expressions of category **Prog**.

$$\begin{aligned} a &::= n \mid x \mid a + a \mid a \star a \mid a - a \\ b &::= \top \mid \bot \mid a = a \mid a \leq a \mid \neg b \mid b \ \wedge b \\ p &::= x := a \mid \mathit{skip} \mid p; p \mid \textbf{if } b \textbf{ then } p \textbf{ else } p \mid \textbf{while } b \textbf{ do } p \end{aligned}$$

We consider programs as expressions of category **Prog**. We denote the set of variables occurring in a program p with $Var(p)$, the set of subterms of p of category **BExp** with $Bool(p)$, and the set of subterms of p of category **AExp** with $Arith(p)$. Furthermore, with $cl(p)$ we denote the smallest set of programs such that:

- $cl(p)$ is closed under subterms, i.e., if $p_1 \in cl(p)$, $p_2 \in Sub(p)$ and $p_2 \in \textbf{Prog}$, then $p_2 \in cl(p)$; and
- if **while** b **do** $p_1 \in cl(p)$, then also $p_1; \textbf{while } b \textbf{ do } p_1 \in cl(p) \in cl(p)$.

3.2 Operational Semantics

Given a finite subset of variables $X \subseteq \mathfrak{X}$, we define the set of states over X, denoted with $State_X$, as the set of total mappings $\mu : X \to \mathbb{N}$.

We define the function $\mathcal{B}^X$ that interprets expressions of category **BExp** as a function from $State_X$ to $\mathbb{B}$.

$$\begin{aligned} \mathcal{B}^X(s, a_1 \ \sigma \ a_2) &= \mathcal{A}^X(s, a_1) \ \sigma \ \mathcal{A}^X(s, a_2) \text{ for } \sigma \in \{=, \leq\} \\ \mathcal{B}^X(s, \neg b) &= \neg \mathcal{B}^X(s, b) \\ \mathcal{B}^X(s, b_1 \wedge b_2) &= \mathcal{B}^X(s, b_1) \wedge \mathcal{B}^X(s, b_2) \end{aligned}$$

We define the function $\mathcal{A}^X$ that interprets expressions of category **AExp** as a function from $State_X$ to $\mathbb{N}$.

$$\begin{aligned} \mathcal{A}^X(s, n) &= n && \text{for } n \in \mathbb{N} \\ \mathcal{A}^X(s, x) &= \mathit{state}(x) && \text{for } x \in X \\ \mathcal{A}^X(s, a_1 \ \rho \ a_2) &= \mathcal{A}^X(s, a_1) \ \rho \ \mathcal{A}^X(s, a_2) && \text{for } \rho \in \{+, \star\} \\ \mathcal{A}^X(s, a_1 - a_2) &= \mathcal{A}^X(s, a_1) - \mathcal{A}^X(s, a_2) && \text{if } \mathcal{A}^X(s, a_1) - \mathcal{A}^X(s, a_2) \geq 0 \\ \mathcal{A}^X(s, a_1 - a_2) &= 0 && \text{if } \mathcal{A}^X(s, a_1) - \mathcal{A}^X(s, a_2) < 0 \end{aligned}$$

For $s \in State_X$, $x \in X$ and $n \in \mathbb{N}$, we define $s[x \mapsto n](y) = n$, if $x = y$, and $s[x \mapsto n](y) = s(y)$ if $x \neq y$.

For a program p and a set X such that $var(p) \subseteq X \subseteq \mathfrak{X}$, we define the operational semantics as follows. We consider the transition system $(\Gamma, T, \Rightarrow)$, where $\Gamma = \{(q, s) \mid q \in cl(p), s \in State_X\}$, $T = State_X$, and $\Rightarrow \subseteq \Gamma \times (\Gamma \cup T)$.

We define the relation $\Rightarrow$ as the smallest relation such that for each $s \in State_X$, for each $a \in \textbf{AExp}$, and for each $b \in \textbf{BExp}$

- we have $(\mathit{skip}, s) \Rightarrow s$;
- we have $(x := a, s) \Rightarrow s[x \mapsto \mathcal{A}^X(a, s)]$;
- $(p_1, s) \Rightarrow (p_1', s')$ implies $(p_1; p_2, s) \Rightarrow (p_1'; p_2, s')$;
- $(p_1, s) \Rightarrow s'$ implies $(p_1; p_2, s) \Rightarrow (p_2, s')$;

[1] In this paper, we restrict ourselves to natural numbers, but the approach can be extended straightforwardly to other concrete domains.

– we have (**if** b **then** p_1 **else** $p_2, s) \Rightarrow (p_1, s)$, if $\mathcal{P}^X(b, s) = \top$;
– we have (**if** b **then** p_1 **else** $p_2, s) \Rightarrow (p_2, s)$, if $\mathcal{P}^X(b, s) = \bot$; and
– we have (**while** b **do** $p, s) \Rightarrow$ (**if** b **then** $(p;$ **while** b **do** $p)$ **else** $skip, s)$.

Note that $\Rightarrow$ is deterministic, i.e., for any s, t, t', if $s \Rightarrow t$ and $s \Rightarrow t'$, then $t = t'$. We say that p terminates on s with outcome t if $(p, s) \Rightarrow^* t$ for $t \in T$. We say that p does not terminate on s if there is no $t \in T$ such that $(p, s) \Rightarrow^* t$. We take notice of the fact that if p does not terminate on s, then there is an infinite sequence $(p, s) \Rightarrow (p', s') \Rightarrow \ldots$ starting from (p, s).

3.3 Normal Form

In order to simplify our encoding of *While* programs into the description logic $\mathcal{ALC}(\mathcal{D})$ later, we define a notion of normal forms for *While* programs. Consider the following substitutions, preserving the operational semantics of programs, for a fresh variable x, for $\rho \in \{+, \star, -\}$, and $\pi \in \{=, \leq\}$:

$$\begin{array}{rl}
\varphi[x := a_1\ \rho\ a_2] \rightsquigarrow x_1 := a_1\ ;\ \varphi[x := x_1\ \rho\ a_2] & \text{if } a_1 \notin \mathcal{X} \\
\varphi[x := a_1\ \rho\ a_2] \rightsquigarrow x_2 := a_2\ ;\ \varphi[x := a_1\ \rho\ x_2] & \text{if } a_2 \notin \mathcal{X} \\
\varphi[a_1\ \pi\ a_2] \rightsquigarrow x_1 := a_1\ ;\ \varphi[x_1\ \pi\ a_2] & \text{if } a_1 \notin \mathcal{X} \\
\varphi[a_1\ \pi\ a_2] \rightsquigarrow x_2 := a_2\ ;\ \varphi[a_1\ \pi\ x_2] & \text{if } a_2 \notin \mathcal{X} \\
p \rightsquigarrow p; skip & \text{if } p \text{ not of the form } q; skip
\end{array}$$

Using the transformations on programs given by the above substitutions, we can transform any program p to an operationally equivalent program p' such that the following holds:

– each subexpression of p' of category **AExp** is either of the form $x\ \rho\ y$, for $x, y \in \mathcal{X}$ and $\rho \in \{+, \star, -\}$, or of the form n for $n \in \mathbb{N}$;
– for each subexpression of p' of category **BExp** of the form $t\ \rho\ s$, for $\rho \in \{=, \leq\}$, holds $t, s \in \mathcal{X}$; and
– either $p' = skip$, or p' is of the form $e; skip$.

We will say that programs that satisfy this particular condition are in normal form.

4 Encoding Programs into $\mathcal{ALC}(\mathcal{D})$

We model programs in the language *While* using description logic and its model-theoretic semantics. The concrete (i.e. numerical) values in the programming language correspond to concrete values in the description logic. States are represented by objects, and programs are represented by concepts. We represent the execution of programs on states by a (functional) role $\mathsf{nextState}$. In particular, for a given program p with $Var(p) = \{x_1, \ldots, x_n\}$, we denote states $s \in State_{Var(p)}$ with objects that have concrete features $\mathsf{valueOf}_{x_i}$.

An execution of a program p will then be modelled by means of a $\mathsf{nextState}$ sequence of objects. The objects in this sequence represent the states occurring in the particular execution of the program. Whenever in this execution of the program there occurs a state s such that the execution is continued from this state s with the intermediate program p', the object corresponding to s is an element of the concept $C_{p'}$. Also, whenever this execution terminates, the final object in this sequence is an element of the concept C_{skip}.

4.1 Encoding

Take an arbitrary program p, i.e., an expression of category **Prog**. W.l.o.g., we assume p is in normal form. We define an $\mathcal{ALC}(\mathcal{D})$ TBox $\mathcal{T}^p$ as follows. We use concept names C_q for each $q \in cl(p)$, and concept names D_b for each $b \in Bool(p)$.

For each variable $x \in Var(p)$, we create a concrete feature $\mathsf{valueOf}_x$, and we require

$$\top \sqsubseteq \neg \mathsf{valueOf}_x\uparrow \tag{1}$$

We let $\mathsf{nextState}$ be an abstract feature and for C_{skip} we require:

$$C_{skip} \sqsubseteq \neg\exists \mathsf{nextState}.\top \tag{2}$$

For each $b \in Bool(p)$, we require the following, where x_1, x_2 range over X, and b_1, b_2 range over $Bool(p)$:

$$D_{x_1=x_2} \equiv \exists(\mathsf{valueOf}_{x_1})(\mathsf{valueOf}_{x_2}).= \tag{3}$$
$$D_{x_1\leq x_2} \equiv \exists(\mathsf{valueOf}_{x_1})(\mathsf{valueOf}_{x_2}).\leq \tag{4}$$
$$D_{\neg b_1} \equiv \neg D_{b_1} \tag{5}$$
$$D_{b_1\wedge b_2} \equiv D_{b_1} \sqcap D_{b_2} \tag{6}$$

Furthermore, we let $D_\top$ denote $\top$ and $D_\bot$ denote $\bot$. Then, for each $q \in cl(p)$ of the form $p_1; p_2$ we require the following for C_q, where p_1, p_2, q_1, q_2 range over $cl(p)$, x, y_1, y_2 range over X, a ranges over $Arith(p)$,

$$C_{(x:=a);p_2} \sqsubseteq \exists \mathsf{nextState}.\top \sqcap \exists \mathsf{nextState}.C_{p_2} \tag{7}$$
$$C_{skip;p_2} \sqsubseteq C_{p_2} \tag{8}$$
$$C_{(x:=n);p_2} \sqsubseteq \exists(\mathsf{nextState}\ \mathsf{valueOf}_x).=_n \tag{9}$$
$$C_{(x:=y);p_2} \sqsubseteq \exists(\mathsf{nextState}\ \mathsf{valueOf}_x)(\mathsf{valueOf}_y).= \tag{10}$$
$$C_{(x:=y_1+y_2);p_2} \sqsubseteq \exists(\mathsf{nextState}\ \mathsf{valueOf}_x)(\mathsf{valueOf}_{y_1})(\mathsf{valueOf}_{y_2}).+ \tag{11}$$
$$\begin{aligned} C_{(x:=y_1-y_2);p_2} \sqsubseteq\ & (\neg\exists(\mathsf{valueOf}_{y_2})(\mathsf{valueOf}_{y_1}).\leq \sqcup \\ & \exists(\mathsf{valueOf}_{y_1})(\mathsf{nextState}\ \mathsf{valueOf}_x)(\mathsf{valueOf}_{y_2}).+) \sqcap \\ & (\neg\exists(\mathsf{valueOf}_{y_2})(\mathsf{valueOf}_{y_1}).> \sqcup \\ & \exists(\mathsf{nextState}\ \mathsf{valueOf}_x).=_0) \end{aligned} \tag{12}$$
$$C_{(x:=y_1\star y_2);p_2} \sqsubseteq \exists(\mathsf{nextState}\ \mathsf{valueOf}_x)(\mathsf{valueOf}_{y_1})(\mathsf{valueOf}_{y_2}).\star \tag{13}$$
$$C_{(x:=a);p_2} \sqsubseteq \exists(\mathsf{valueOf}_y)(\mathsf{nextState}\ \mathsf{valueOf}_y).= \quad \text{for } y \neq x \tag{14}$$
$$C_{(\textbf{if } b \textbf{ then } q_1 \textbf{ else } q_2);p_2} \sqsubseteq (\neg D_b \sqcup C_{q_1;p_2}) \sqcap (D_b \sqcup C_{q_2;p_2}) \tag{15}$$
$$C_{(\textbf{while } b \textbf{ do } q);p_2} \sqsubseteq (\neg D_b \sqcup C_{q;(\textbf{while } b \textbf{ do } q);p_2}) \sqcap (D_b \sqcup C_{p_2}) \tag{16}$$

Notice that, in general, the TBox $\mathcal{T}^p$ is not acyclic, since Axioms (7), (8), (15) and (16) can together induce a cycle.

4.2 Correctness

In order to use the above encoding of a program p into an $\mathcal{ALC}(\mathcal{D})$ TBox $\mathcal{T}^p$ to reduce reasoning problems over programs into $\mathcal{ALC}(\mathcal{D})$ reasoning, we show the following correspondence between the operational semantics of p and the model theoretic semantics of $\mathcal{T}^p$.

Lemma 1. *For any program p, any X such that $Var(p) \subseteq X \subseteq \mathfrak{X}$, any state $s \in State_X$, any $b \in Bool(p)$, and for any model $\mathfrak{I} = (\Delta^{\mathfrak{I}}, \cdot^{\mathfrak{I}})$ of $\mathcal{T}^p$, we have that $d \in \Delta^{\mathfrak{I}}$ and $(d, s(x_i)) \in \mathsf{valueOf}^{\mathfrak{I}}_{x_i}$ for all $1 \leq i \leq n$ implies that $d \in C^{\mathfrak{I}}_b$ iff $\mathcal{B}^X(b, s) = \top$.*

Proof (sketch). By induction on the structure of b. All cases follow directly from the fact that Axioms (3)-(6) hold.

Theorem 1. *For any program p, any X such that $Var(p) \subseteq X \subseteq \mathfrak{X}$, any state $s \in State_X$ such that p terminates on s with outcome t, and for any model $\mathfrak{I} = (\Delta^{\mathfrak{I}}, \cdot^{\mathfrak{I}})$ of $\mathcal{T}^p$ we have that $d \in C^{\mathfrak{I}}_p$ and $(d, s(x_i)) \in \mathsf{valueOf}^{\mathfrak{I}}_{x_i}$ for all $1 \leq i \leq n$ implies that $e \in C^{\mathfrak{I}}_{skip}$ and $(e, t(x_i)) \in \mathsf{valueOf}^{\mathfrak{I}}_{x_i}$ for all $1 \leq i \leq n$, for some $e \in \Delta^{\mathfrak{I}}$.*

Proof. By induction on the length of the $\Rightarrow$-derivation $(p, s) \Rightarrow^k t$. Assume $d \in C^{\mathfrak{I}}_p$ and $(d, s(x_i)) \in \mathsf{valueOf}^{\mathfrak{I}}_{x_i}$ for all $1 \leq i \leq n$, for some $d \in \Delta^{\mathfrak{I}}$. The base case $k = 0$ holds vacuously. In the case for $k = 1$, we know $p = skip$, since p is in normal form. Therefore, we know $s = t$, and thus $e = d$ witnesses the implication.

In the inductive case, we distinguish several cases. Case $p = skip; q$. We know $(p, s) \Rightarrow (q, s) \Rightarrow^{k-1} t$. Since $\mathfrak{I}$ satisfies $\mathcal{T}^p$, by Axiom (8), we know $d \in C^{\mathfrak{I}}_q$. The result now follows directly by the induction hypothesis.

Case $p = (x := a); q$. We know $(p, s) \Rightarrow (q, s') \Rightarrow^{k-1} t$, and $s' = s[x \mapsto \mathcal{A}^X(a, s)]$. Since $\mathfrak{I}$ satisfies $\mathcal{T}^p$, by Axioms (7), (9)-(13) and (14), we know there must exist a $d' \in C^{\mathfrak{I}}_q$ such that $(d', s'(x_i)) \in \mathsf{valueOf}^{\mathfrak{I}}_{x_i}$ for all $1 \leq i \leq n$. Then by the induction hypothesis, the result follows directly.

Case $p = (\mathbf{if}\, b\, \mathbf{then}\, p_1\, \mathbf{else}\, p_2); q$. Assume $\mathcal{B}^X(b, s) = \top$. Then $(p, s) \Rightarrow (p_1; q, s) \Rightarrow^{k-1} t$. By Lemma 1, we know $d \in D^{\mathfrak{I}}_b$. Then, by the fact that Axiom (15) holds, we know $d \in C^{\mathfrak{I}}_{p_1;q}$. The result now follows directly by the induction hypothesis. For $\mathcal{B}^X(b, s) = \bot$ an analogous argument holds.

Case $p = (\mathbf{while}\, b\, \mathbf{do}\, p_1); q$. Assume $\mathcal{B}^X(b, s) = \top$. Then $(p, s) \Rightarrow^2 (p_1; p, s) \Rightarrow^{k-2} t$. By Lemma 1, we know $d \in D^{\mathfrak{I}}_b$. Then, by the fact that Axiom (16) holds, we know $d \in C_{p_1;p}$. The result now follows directly by the induction hypothesis.

If, however, in the same case holds $\mathcal{B}^X(b, s) = \bot$, then $(p, s) \Rightarrow^3 (q, s) \Rightarrow^{k-3} t$. By Lemma 1, we know $d \notin D^{\mathfrak{I}}_b$. By the fact that Axiom (16) holds, we know $d \in C^{\mathfrak{I}}_q$. The result now follows directly by the induction hypothesis.

Theorem 2. *For any program p, any X such that $Var(p) \subseteq X \subseteq \mathfrak{X}$ any state $\{x_1 \mapsto c_1, \ldots, x_n \mapsto c_n\} = s \in State_X$ such that p does not terminate on s, there exists a model $\mathfrak{I} = (\Delta^{\mathfrak{I}}, \cdot^{\mathfrak{I}})$ of $\mathcal{T}^p$ such that for some $d \in \Delta^{\mathfrak{I}}$ we have $d \in C^{\mathfrak{I}}_p$, $(d, c_i) \in \mathsf{valueOf}^{\mathfrak{I}}_{x_i}$, for all $1 \leq i \leq n$, and $C^{\mathfrak{I}}_{skip} = \emptyset$.*

Proof. Since p does not terminate on s, we know there exists an infinite $\Rightarrow$-sequence $(p_i, s_i) \Rightarrow (p_{i+1}, s_{i+1})$, for $i \in \mathbb{N}$, where $(p_1, s_1) = (p, s)$. Consider the following interpretation $\mathfrak{I} = (\Delta^{\mathfrak{I}}, \cdot^{\mathfrak{I}})$, where $\Delta^{\mathfrak{I}} = \{(p_i, s_i) \mid i \in \mathbb{N}\}$. For $q \in cl(p)$, we let $C^{\mathfrak{I}}_q = \{(p_i, s_i) \mid p_i = q\}$. For $b \in Bool(p)$, we let $D^{\mathfrak{I}}_b = \{(p_i, s_i) \mid i \in \mathbb{N}, \mathcal{P}^X(b, s) = \top\}$. For each $x \in X$, we let $\mathsf{valueOf}^{\mathfrak{I}}_x = \{((p_i, s_i), s_i(x)) \mid i \in \mathbb{N}\}$. We let $\mathsf{nextState}^{\mathfrak{I}} = \{((p_i, s_i), (p_{i+1}, s_{i+1})) \mid i \in \mathbb{N}\}$.

The definition of $\mathfrak{I}$ implies that $C^{\mathfrak{I}}_{skip} = \emptyset$. Assume $(p_k, s_k) \in C^{\mathfrak{I}}_{skip}$. Then $p_k = skip$, and thus $(p_k, s_k) \Rightarrow s_k$, which contradicts our assumption of non-termination.

Clearly, $\mathfrak{I}$ satisfies Axiom (1). Since $C^{\mathfrak{I}}_{skip} = \emptyset$, $\mathfrak{I}$ also satisfies Axiom (2). It is easy to verify, that by the definition of $D^{\mathfrak{I}}_b$ we get that $\mathfrak{I}$ satisfies Axioms (3)-(6).

To see that $\mathfrak{I}$ satisfies Axioms (7)-(16), we take an arbitrary object (p_j, s_j) in the interpretation an arbitrary class $C^{\mathfrak{I}}_q$, and we distinguish several cases. Consider

$p_j = skip; q$. Then by the constraints on $\Rightarrow$, $p_{j+1} = q$ and $s_{j+1} = s_j$. This witnesses that the subsumption in Axiom (8) holds.

Consider $p_j = (x := a); q$. Then by the constraints on $\Rightarrow$, $p_{j+1} = q$ and $s_{j+1} = s_j[x \mapsto \mathcal{A}^X(a, s_j)]$. By definition of $\mathfrak{I}$, we know $((p_j, s_j), (p_{j+1}, s_{j+1})) \in \mathsf{nextState}^{\mathfrak{I}}$. It is now easy to verify that the subsumptions in Axioms (7), (9)-(13) and (14) are satisfied.

Consider $p_j = (\textbf{if } b \textbf{ then } p'_1 \textbf{ else } p'_2); q$. Assume $\mathcal{B}^X(b, s_j) = \top$. Then $(p_j, s_j) \in D_b^{\mathfrak{I}}$. Also, by the constraints on $\Rightarrow$, $p_{j+1} = p'_1; q$ and $s_{j+1} = s_j$. It is easy to verify that, in this case, the subsumption in Axiom (15) holds. The case for $\mathcal{B}^X(b, s_j) = \bot$ is completely analogous.

Consider $p_j = (\textbf{while } b \textbf{ do } p'); q$. If $\mathcal{B}^X(b, s_j) = \top$, then $(p_j, s_j) \in D_b^{\mathfrak{I}}$ and $p_{j+1} = p'; p_j$ and $s_{j+1} = s_j$. If $\mathcal{B}^X(b, s_j) = \bot$, then $(p_j, s_j) \notin D_b^{\mathfrak{I}}$ and $p_{j+1} = skip; q$ and $s_{j+1} = s_j$. It is easy to verify that, in either case, the subsumption in Axiom (16) holds.

Theorem 3. *For any program p, any X such that $Var(p) \subseteq X \subseteq \mathfrak{X}$ any state $\{x_1 \mapsto c_1, \ldots, x_n \mapsto c_n\} = s \in State_X$ such that p terminate on s, there exists a model $\mathfrak{I} = (\Delta^{\mathfrak{I}}, \cdot^{\mathfrak{I}})$ of $\mathcal{T}^p$ such that for some $d \in \Delta^{\mathfrak{I}}$ we have $d \in C_p^{\mathfrak{I}}$, $(d, c_i) \in \mathsf{valueOf}_{x_i}^{\mathfrak{I}}$, for all $1 \leq i \leq n$.*

Proof (Sketch). We know there exists $(p, s) \Rightarrow^k (p', s') \Rightarrow t$. Analogously to the proof of Theorem 2, we can construct a model $\mathfrak{I}$ from the sequence $(p, s) \Rightarrow^k (p', s')$. By similar arguments to those in the proof of Theorem 2 it follows that $\mathfrak{I} \models \mathcal{T}^p$. Then, $(p, s) \in C_p^{\mathfrak{I}}$ witnesses the further constraints on $\mathfrak{I}$.

Note that the syntax and operational semantics of *While* can be adapted to various concrete domains with varying operators. The encoding into $\mathcal{ALC}(\mathcal{D})$ can be adapted correspondingly, and a corresponding correlation between the operational semantics and the model theoretic semantics can be proven.

4.3 Example

To illustrate the method described above, of encoding *While* programs into $\mathcal{ALC}(\mathcal{D})$ TBoxes, we will consider an example. Let p_0 be the following program in normal form, that computes the factorial of the value stored in variable x and outputs this in variable y. Note that variable z is simply used to refer to the constant value 1.

$$p_0 = \quad (y := 1; z := 1;\ \textbf{while } x > z \textbf{ do } (y := y \star x;\ x := x - z); skip)$$

Furthermore, we will use the following abbreviations to refer to subprograms of p_0.

$$\begin{aligned} p_1 &= (z := 1;\ \textbf{while } x > z \textbf{ do } (y := y \star x;\ x := x - z); skip) \\ p_2 &= (\textbf{while } x > z \textbf{ do } (y := y \star x;\ x := x - z); skip) \\ p_3 &= (y := y \star x;\ x := x - z) \\ p_4 &= (x := x - z) \end{aligned}$$

We can now construct the $\mathcal{ALC}(\mathcal{D})$ TBox $\mathcal{T}^{p_0}$, in the fashion described above.

$$
\begin{aligned}
\mathcal{T}^{p_0} = \{ \quad & \top \sqsubseteq \neg\mathsf{valueOf}_x\!\uparrow, \\
& \top \sqsubseteq \neg\mathsf{valueOf}_y\!\uparrow, \\
& \top \sqsubseteq \neg\mathsf{valueOf}_z\!\uparrow, \\
& C_{skip} \sqsubseteq \neg\exists\mathsf{nextState}.\top, \\
& D_{x<z} \equiv \exists(\mathsf{valueOf}_x)(\mathsf{valueOf}_z).<, \\
& C_{(y:=1);p_1} \sqsubseteq \exists\mathsf{nextState}.\top \sqcap \exists\mathsf{nextState}.C_{p_1}, \\
& C_{(z:=1);p_2} \sqsubseteq \exists\mathsf{nextState}.\top \sqcap \exists\mathsf{nextState}.C_{p_2}, \\
& C_{(y:=1);p_1} \sqsubseteq \exists(\mathsf{nextState}\ \mathsf{valueOf}_y).=_1, \\
& C_{(y:=1);p_1} \sqsubseteq \exists(\mathsf{valueOf}_x)(\mathsf{nextState}\ \mathsf{valueOf}_x).=, \\
& C_{(y:=1);p_1} \sqsubseteq \exists(\mathsf{valueOf}_z)(\mathsf{nextState}\ \mathsf{valueOf}_z).=, \\
& C_{(z:=1);p_2} \sqsubseteq \exists(\mathsf{nextState}\ \mathsf{valueOf}_z).=_1, \\
& C_{(z:=1);p_2} \sqsubseteq \exists(\mathsf{valueOf}_x)(\mathsf{nextState}\ \mathsf{valueOf}_x).=, \\
& C_{(z:=1);p_2} \sqsubseteq \exists(\mathsf{valueOf}_y)(\mathsf{nextState}\ \mathsf{valueOf}_y).=, \\
& C_{p_2} \sqsubseteq (\neg D_{x<z} \sqcup C_{p_3;p_2}) \sqcap (D_{x<z} \sqcup C_{skip}), \qquad (\dagger) \\
& C_{(y:=y\star x);p_4;p_2} \sqsubseteq \exists\mathsf{nextState}.\top \sqcap \exists\mathsf{nextState}.C_{p_4;p_2}, \qquad (\dagger) \\
& C_{(x:=x-z);p_2} \sqsubseteq \exists\mathsf{nextState}.\top \sqcap \exists\mathsf{nextState}.C_{p_2}, \qquad (\dagger) \\
& C_{(y:=y\star x);p_4;p_2} \sqsubseteq \exists(\mathsf{nextState}\ \mathsf{valueOf}_y)(\mathsf{valueOf}_y)(\mathsf{valueOf}_x).\star, \\
& C_{(y:=y\star x);p_4;p_2} \sqsubseteq \exists(\mathsf{valueOf}_x)(\mathsf{nextState}\ \mathsf{valueOf}_x).=, \\
& C_{(y:=y\star x);p_4;p_2} \sqsubseteq \exists(\mathsf{valueOf}_z)(\mathsf{nextState}\ \mathsf{valueOf}_z).=, \\
& C_{(x:=x-z);p_2} \sqsubseteq (\neg\exists(\mathsf{valueOf}_z)(\mathsf{valueOf}_x).\leq \sqcup \\
& \qquad \exists(\mathsf{valueOf}_x)(\mathsf{nextState}\ \mathsf{valueOf}_x)(\mathsf{valueOf}_z).+) \sqcap \\
& \qquad (\neg\exists(\mathsf{valueOf}_z)(\mathsf{valueOf}_x).> \sqcup \\
& \qquad \exists(\mathsf{nextState}\ \mathsf{valueOf}_x).=_0), \\
& C_{(x:=x-z);p_2} \sqsubseteq \exists(\mathsf{valueOf}_y)(\mathsf{nextState}\ \mathsf{valueOf}_y).=, \\
& C_{(x:=x-z);p_2} \sqsubseteq \exists(\mathsf{valueOf}_z)(\mathsf{nextState}\ \mathsf{valueOf}_z).= \qquad \}
\end{aligned}
$$

Note that $\mathcal{T}^{p_0}$ is not acyclic, since there are inclusion axioms (a) with C_{p_2} as lhs and $C_{p_3;p_2}$ in the rhs, (b) with $C_{p_3;p_2}$ as lhs and $C_{p_4;p_2}$ in the rhs, and (c) with $C_{p_4;p_2}$ as lhs and C_{p_2} in the rhs. These axioms are marked with the symbol †.

It is straightforward to construct models of the TBox $\mathcal{T}^{p_0}$. One can simply take an execution of the program p_0, and transform this execution into a model of $\mathcal{T}^{p_0}$ according to the intuition behind the encoding described above.

5 Reasoning Problems

Theorems 1, 2 and 3 allow us to use the encoding of *While* programs into $\mathcal{ALC}(\mathcal{D})$ to reduce several reasoning problems over *While* programs to reasoning problems over $\mathcal{ALC}(\mathcal{D})$. For instance, termination of a program p reduces to unsatisfiability of the ABox $\mathcal{A}^p = \{o_1 : C_p\}$ with respect to the TBox $\mathcal{T}^p \cup \{C_{skip} \sqsubseteq \bot\}$.

Also, we are able to encode abduction problems over *While* problems in the description logic $\mathcal{ALCO}(\mathcal{D})$, which is $\mathcal{ALC}(\mathcal{D})$ extended with nominals. The question what input states for a program p could have led to the (partial) output state s, for $dom(s) \subseteq Var(p)$, reduces to finding models for $\mathcal{A}^p = \{i : C_p, o : C_{skip}\} \cup \{(o, s(x)) : \mathsf{valueOf}_x \mid x \in dom(s)\}$ and $\mathcal{T}^p$, where C_{skip} is required to be a nominal concept.

Another example is checking whether two (terminating) programs p_1 and p_2 are equivalent. Without loss of generality, we can assume $Var(p_1) = Var(p_2)$. This equivalence check can be reduced to the problem of $\mathcal{ALCO}(\mathcal{D})$ unsatisfiability of the ABox $\mathcal{A}^{p_1,p_2} = \{o : C_{p_1}, o : C_{p_2}, s : C_{test}\}$ with respect to the TBox $\mathcal{S}^{p_1} \cup \mathcal{S}^{p_2} \cup \mathcal{T}^{eq}$, where $\mathcal{S}^{p_i}$ is

$\mathcal{T}^{p_i}$ with C_{skip} replaced by C^i_{skip}, and $\mathcal{T}^{eq} = \{C_{test} \equiv (\exists.(\mathsf{res}_1\ \mathsf{valueOf}_{x_1})(\mathsf{res}_2\ \mathsf{valueOf}_{x_1}).\neq \sqcup \cdots \sqcup \exists.(\mathsf{res}_1\ \mathsf{valueOf}_{x_n})\ (\mathsf{res}_2\ \mathsf{valueOf}_{x_n}).\neq) \sqcap \exists\mathsf{res}_1.C^1_{skip} \sqcap \exists\mathsf{res}_2.C^2_{skip}\}$, for res_1, res_2 abstract features, and C^1_{skip}, C^2_{skip} and C_{test} nominal concepts.

Naturally, this approach allows us to encode more intricate reasoning problems over *While* programs into description logic reasoning problems. Description logic offers us a very flexible formalism to express a variety of reasoning problems over *While* programs.

5.1 Decidability

A bit of care has to be taken with this powerful and general approach. The problems we consider generally balance on the bounds of decidability.

A concrete domain $\mathcal{D}$ is called arithmetic if its values contain the natural numbers, and it contains predicates for equality, equality with zero and incrementation. Unfortunately, satisfiability of $\mathcal{ALC}(D)$ concepts for arithmetic concrete domains D with respect to general TBoxes is undecidable [4]. So, for many cases, our approach doesn't directly result in a decision procedure.

This is no surprise, however. We know that we cannot decide equivalence of *While* programs in general. For instance for the concrete domain $\mathbb{Z}$ (with addition and equality) we can easily encode the undecidable problem of whether a given Diophantine equation $\Phi(x_1, \ldots, x_n) = 0$ has an integer solution (the subject of Hilbert's Tenth Problem) as a reasoning problem over *While* programs. Similarly, it can be proven that equivalence of *While* programs with concrete domain $\mathbb{N}$ is undecidable.

Nevertheless, several decidable fragments of the *While* language can be obtained by either restricting the concrete domains or by forbidding statements of the form (**while** b **do** p). By forbidding these **while**-statements, we end up with acyclic TBoxes. We know that reasoning with respect to acyclic TBoxes is decidable for $\mathcal{ALC}(\mathcal{D})$. Restricting the concrete domain (and keeping **while**-statements), does not change the fact that we are dealing with general TBoxes. In this case, the concrete domain needs to be restricted quite severely, to get decidability.

5.2 Examples

We can use the example from Section 4.3 to illustrate how to encode several reasoning problems over *While* programs into $\mathcal{ALC}(\mathcal{D})$ reasoning problems. For instance, we can encode the problem of checking whether p_0 is terminating as the unsatisfiability problem of the ABox $\{o_1 : C_{p_0}\}$ with respect to the TBox $\mathcal{T}^p \cup \{C_{skip} \sqsubseteq \bot\}$.

Using the expressivity of the description logic $\mathcal{ALC}(\mathcal{D})$, we can in fact express a variety of different semantic properties to be checked automatically. For instance, in this fashion, we can express the problem whether p_0 does not terminate with an output value for y that is less than or equal to 20 for all input values for x that are greater than or equal to 4. This problem can be reduced to the unsatisfiability of the ABox $\{o_1 : C_{p_0} \sqcap \exists(\mathsf{hasValue}_x).\geq_4\}$ with respect to the TBox $\mathcal{T}^p \cup \{C_{skip} \sqsubseteq \exists(\mathsf{hasValue}_y).\leq_{20}\}$.

These examples of expressing reasoning problems in the $\mathcal{ALC}(\mathcal{D})$ language illustrate the flexibility we get in expressing different reasoning problems. We can reduce the decision of any semantic property of *While* programs that is expressible using the modelling of *While* programs in the $\mathcal{ALC}(\mathcal{D})$ language to reasoning on $\mathcal{ALC}(\mathcal{D})$.

6 Further Research

The results presented in this paper are only the beginning of a larger inquiry investigating the possibilities and bounds of approaching automated reasoning over (procedural) programming languages by means of assigning model-theoretic semantics to programs. We suggest a number of directions for further research needed to get a better understanding of the topic.

Similar encodings of programming languages into the description logic $\mathcal{ALC}(\mathcal{D})$ could also be devised also for other procedural programming languages, as well as for declarative programming languages. It needs to be investigated to what extent this method is extendable to such other languages. Suitable programming languages for which this could be investigated as a next step include extensions of the programming language *While* with language constructs for nondeterminism or parallellism, procedural programming languages that are based on **goto**-statements rather than **while**-statements, and simple, representative functional and logic programming languages that operate on similar domains (e.g. numerical values). Once the method presented in this paper has been applied to such languages, it will also be possible to investigate to what extent reasoning problems on multiple programs of different programming languages (e.g. the equivalence problem of a *While* program and a program of a **goto**-based language) can be encoded in this framework.

Another important direction for further research is identifying larger fragments of the programming language *While* for which reasoning problems on programs such as the ones we considered in this paper are decidable. By encoding programs into description logics, we get a conceptually simpler setting to investigate such questions. Programming languages can have a variety of different constructs behaving in various ways. Reasoning over such diverse structures can get rather messy and complex. On the other hand, practically all reasoning problems for description logics can be reduced to finding models for description logic knowledge bases (i.e. reduced to the satisfiability problem). The problem of finding such models is conceptually simple, and the semantic definition of the description logic language guides the search for models. Such a conceptually simpler setting might make it easier to identify decidable fragments. In addition to this conceptual simplification of the problem, we could use results and techniques from the field of description logic when investigating the decidability of such fragments. For those fragments of the programming languages that lead to decidable reasoning problems, we can investigate the computational complexity of these reasoning problems.

7 Conclusions

We assigned a model theoretic semantics to programs of the simple procedural language *While*, by encoding them into the description logic $\mathcal{ALC}(\mathcal{D})$. This allowed us to express a variety of reasoning problems over *While* programs using the expressivity of the description logic $\mathcal{ALC}(\mathcal{D})$. Furthermore, for a number of restricted fragments of the programming language *While*, this directly results in a method of solving such reasoning problems using existing (description logic) algorithms. Furthermore, this encoding leads to a new approach of exploring what fragments of the programming language allow for decidable reasoning over programs. Further research includes a further characterization of fragments of the language that allow decidable reasoning, and extending this approach to different procedural (and declarative) programming languages.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2003)
2. Baader, F., Hanschke, P.: A scheme for integrating concrete domains into concept languages. In: Proceedings of the 12th international joint conference on Artificial intelligence - Volume 1. pp. 452–457. IJCAI'91, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1991)
3. Lutz, C.: Description logics with concrete domains—a survey. In: Advances in Modal Logics Volume 4. King's College Publications (2003)
4. Lutz, C.: NExpTime-complete description logics with concrete domains. ACM Trans. Comput. Logic 5(4), 669–705 (Oct 2004)
5. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1999)
6. Nielson, H.R., Nielson, F.: Semantics with Applications: A Formal Introduction. John Wiley & Sons, Inc., New York, NY, USA (1992)
7. Schmidt-Schaußß, M., Smolka, G.: Attributive concept descriptions with complements. Artif. Intell. 48(1), 1–26 (Feb 1991)