Two-level reasoning about graph transformation programs

Amani Makhlouf, Christian Percebois, Hanh Nhi Tran

Official URL
DOI : https://doi.org/10.1007/978-3-030-23611-3_7

Two-Level Reasoning About Graph Transformation Programs

Amani Makhlouf, Christian Percebois(✉), and Hanh Nhi Tran

IRIT, University of Toulouse, Toulouse, France
{Amani.Makhlouf,Christian.Percebois,Hanh-Nhi.Tran}@irit.fr

Abstract. This paper presents a method for verifying graph transformation programs written in Small-t$\mathcal{ALC}$, an imperative language which allows expressing graph properties and graph transformations in $\mathcal{ALCQI}$ description logic. We aim at reasoning not only about the local effect when applying a transformation rule on a matched subgraph but also about the global impact on the whole input graph when applying a set of rules. Using $\mathcal{ALCQI}$ assertional and terminological formulae to formalize directed labeled graphs, Small-t$\mathcal{ALC}$ allows specifying local properties on individual nodes and edges as well as global properties on sets of nodes and edges. Our previous work focuses on verifying local properties of the graph. In this paper, we propose a static analyzer at terminological level that intertwines with a static analyzer at assertional level to infer global properties of the transformed graph.

Keywords: Graph transformation · Description logics · Static analysis · Abstract interpretation · Program verification

1 Introduction

To allow verifying the correctness of graph transformations, many works, rooted in algebraic approach for formalizing graph transformations, have introduced logic systems that are specially tailored for expressing graph properties under study (see e.g. [1–5]).

The work presented in this paper uses another approach which directly encodes graphs in an existing logic [6,7] in order to benefit the inference mechanisms provided for the chosen logic. Adopting this approach, we proposed the graph transformation language Small-t$\mathcal{ALC}$ [8] which specifies graphs with $\mathcal{ALCQI}$ description logic formulae [9] and defines transformation statements to manipulate graphs in an imperative paradigm. Transformation specifications and code are based on the same logic thus we can take advantage of a Hoare-like calculus and also of proven program verification techniques to reason about the correctness of graph transformations.

Small-t$\mathcal{ALC}$ graphs are directed and labeled. A graph consists of nodes representing individuals and edges representing relations between individuals. A node can be labeled to express that it belongs to the concept denoted by the node's

https://doi.org/10.1007/978-3-030-23611-3_7

label; the label of an edge denotes the role of the relation represented by the edge. Graph properties can be specified by $\mathcal{ALCQI}$ assertional axioms ($ABox$) about nodes and edges and by terminological axioms ($TBox$) about set of nodes.

A Small-t$\mathcal{ALC}$ program consists of a set of transformation rules. Each rule comprises a precondition specifying the matching constraints of the rule on a host graph, a code consisting of transformation statements and a postcondition specifying the properties of the graph yielded from the rule's application. Both rule's specifications and code are formalized at $ABox$ level. In [7,10], we developed tools to formally verify the correctness of each transformation rule using Hoare logic. However, our previous works allow verifying only a plain set of rules, not the correctness of a whole transformation program. Moreover, using only assertional formulae to specify graph properties, we could analyze only local properties on individual nodes and edges.

We now extend the approach to reason not only about the local effect when applying a transformation rule on a matched subgraph but also about the global impact on the whole input graph when applying a set of rules. For this purpose, first we exploit both $\mathcal{ALCQI}$ assertional formulae ($ABox$) and terminological formulae ($TBox$) to formalize directed labeled graphs, and thus allow specifying respectively local properties as well as global properties. We then propose a static analyzer at terminological level that intertwines with a static analyzer at assertional level to infer global properties of the transformed graph. Rules verification at $ABox$ level was presented in [10]. The focus of this paper is the $TBox$ analyzer for transformation programs.

We introduces our graph transformation language Small-t$\mathcal{ALC}$ in Sect. 2 and present in Sect. 3 the main idea of two-levels reasoning about Small-t$\mathcal{ALC}$ programs by exploiting the $ABox$ and $TBox$ components of $\mathcal{ALCQI}$. In Sect. 4 we explain how to infer, by abstract interpretation, $TBox$ global properties from $ABox$ statements. The relation between $ABox$ and $TBox$ verifications is studied in Sect. 5. Section 6 shows that some monadic second-order properties can be expressed by Small-t$\mathcal{ALC}$ $TBox$ assertions too. We finally provide some discussions on related work in Sect. 7 and wrap up the paper with a conclusion including further work in Sect. 8.

2 The Small-t$\mathcal{ALC}$ Language

Small-t$\mathcal{ALC}$ [8] is an imperative graph transformation language based on the description logic $\mathcal{ALCQI}$ [9]. The distinctive characteristic of this graph transformation language is the tight integration of logical aspects with the intended execution mechanism, with the overall aim to obtain a decidable calculus for reasoning about program correctness in a pre-/post-condition style.

2.1 Logic Foundation

$\mathcal{ALCQI}$ represents knowledge at two levels: $TBox$ introduces the terminology, i.e., the vocabulary of an application domain, while $ABox$ contains assertions about named individuals in terms of this vocabulary. The vocabulary consists of concepts, which denote sets of individuals, and roles, which denote binary relationships between individuals. An interpretation $\mathcal{I}$ that is used to define the semantics of DLs comprises a non-empty set $\Delta^{\mathcal{I}}$ called the interpretation domain and an interpretation function $\cdot^{\mathcal{I}}$. The interpretation function assigns an element $i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to each individual i of the $ABox$, a subset of individuals $C^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to each concept C of the $TBox$, and a subset of ordered pairs of individuals $r^{\mathcal{I}} \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to every role r of the $TBox$.

Let C be a concept, x and y be individuals, and r be a role. If x belongs to the concept C, then x is called C-type. If x is r-related to y, then y is called a r-successor of x. $\mathcal{ALCQI}$ provides concept constructors to build more complex concepts as given in Table 1.

Table 1. $\mathcal{ALCQI}$ concept constructors

Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
negation	$\neg C$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
conjunction	$C \cap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \cup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
existential restriction	$\exists\ r\ C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y, (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
universal restriction	$\forall\ r\ C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y, (x,y) \in r^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$
at-most restriction	$\leq n\ r\ C$	$\{x \in \Delta^{\mathcal{I}} \mid \lvert(x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\rvert \leq n\}$
at-least restriction	$\geq n\ r\ C$	$\{x \in \Delta^{\mathcal{I}} \mid \lvert(x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\rvert \geq n\}$
equality restriction	$= n\ r\ C$	$\{x \in \Delta^{\mathcal{I}} \mid \lvert(x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\rvert = n\}$
inverse role	r^{-1}	$\{(y,x) \mid (x,y) \in r^{\mathcal{I}}\}$

($\exists\ r\ C$) describes the set of individuals having at least a r-successor which is C-type. ($\forall\ r\ C$) presents the set of individuals whose all r-successors are C-type. ($\leq n\ r\ C$) and ($\geq n\ r\ C$) are qualified number restrictions expressing that an individual has at most (respectively at least) n r-successors which are C-type.

2.2 Small-t$\mathcal{ALC}$ Graphs

An interpretation $\mathcal{I}$ can be drawn as a directed labeled graph [11] where $TBox$ represents concepts and roles respectively as nodes labels and edges labels, and $ABox$ specifies individuals and binary relations between them respectively as graph nodes and graph edges.

In Small-t$\mathcal{ALC}$, concept assertions ($i : C$) express that an individual i is C-type, i.e. the node i is labeled with C (C-node). Role assertions in the form ($i\ r\ j$) express that an individual i is connected by the role r to the individual j i.e. the edge (i,j) is labeled with r (r-edge). By combining concept assertions and role assertions, $ABox$ formulae are made up and used to specify properties on named graph nodes and edges. Figure 1 depicts a graph having two A-nodes $a1$, $a2$ and two B-nodes $b1$, $b2$. $b1$ is a r-successor of $a2$ and $b2$ is a r-successor of $a1$. There are also two anonymous C-type nodes which are r-successors of $a1$, thus $a1$ belongs to the concept which has at least 2 C-nodes as r-successors.

In the rest of the paper, we call $AFact$ an $ABox$ assertion, and $AFormula$ an $ABox$ formula.

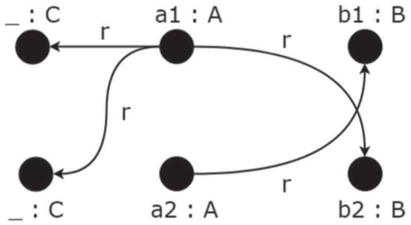

Fig. 1. A graph satisfying the $AFormula$ $(a1 : A) \wedge (a2 : A) \wedge (b1 : B) \wedge (b2 : B) \wedge (a1\ r\ b2) \wedge (a2\ r\ b1) \wedge (a1 : (\leq\ 2\ r\ C))$

$TBox$ axioms use general concept inclusions (GCI) [12] to express properties concerning concepts. $TBox$ axioms are so-called $TFacts$ in Small-t$\mathcal{ALC}$ and are of the form $C \subseteq D$ or $C = D$ where C and D are concepts. An interpretation $\mathcal{I}$ is a model of $C \subseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. When $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in every model of I, D subsumes C. Thus, $TFormulae$, which are Boolean combinations of $TFacts$, can be used now to express global graph properties on set of nodes. For example, the $TFact$ $(\forall\ r^{-1}\ A) \subseteq B$ expresses that r-edges outgoing from A-nodes all go towards B-nodes. The graph of Fig. 1 does not hold this property because it has two r-edges outgoing from A-nodes to C-nodes.

2.3 Small-t$\mathcal{ALC}$ Statements

Small-t$\mathcal{ALC}$ features atomic statements to add, delete or select graph nodes and edges. We have defined five atomic Small-t$\mathcal{ALC}$ statements according to the following grammar, where i and j are node variables which will be bound to the host graph's nodes during the transformation's execution, C is a concept name, r is a role name, F is an $\mathcal{ALCQI}$ $AFormula$ and v is a list of node variables:

$$\begin{array}{lll} stmt ::= & \texttt{add(}i :\ C\texttt{)} & (C^{\mathcal{I}} = C^{\mathcal{I}} + \{i\}) \\ \mid & \texttt{delete(}i :\ C\texttt{)} & (C^{\mathcal{I}} = C^{\mathcal{I}} - \{i\}) \\ \mid & \texttt{add(}i\ r\ i\texttt{)} & (r^{\mathcal{I}} = r^{\mathcal{I}} + \{i, j\}) \\ \mid & \texttt{delete(}i\ r\ i\texttt{)} & (r^{\mathcal{I}} = r^{\mathcal{I}} - \{i, j\}) \\ \mid & \texttt{select}\ \ v\ with\ F & \end{array}$$

Operationally, the first four Small-t$\mathcal{ALC}$ statements define new interpretations i.e. new graphs by adding and deleting individuals (nodes) and pair of individuals (edges) to and from the interpretations of concepts and roles. The interpretation function of $TBox$ concepts and roles are thus evolved. In this sense, a rule operates on $AFormulae$ but affects as well $TFormulae$. Note that the Small-t$\mathcal{ALC}$ statements do not add/delete individuals to/from the graph, but change their label to modify the interpretation represented by the graph.

Since concepts and roles are considered as sets of nodes and set of pairs of nodes respectively, $add(i : C)$ and $add(i\ r\ j)$ have no effects if $i \in C$ and $(i, j) \in r$ respectively. Therefore, no parallel edges with the same label are allowed. The statement $delete(i : C)$ does not remove it definitely from the graph, but excludes it from the interpretation function C^I of the indicated concept C, i.e. the node will no more be labeled with C.

An original construct is the *select* statement that non-deterministically binds node variables to nodes in the subgraph that satisfies an $AFormula$. This assignment is used to select specific nodes where the transformations are requested to occur. The remaining language constructs are conventional control structures: sequence, branching and iteration.

2.4 Small-t$\mathcal{ALC}$ Programs

A Small-t$\mathcal{ALC}$ program consists of a set of transformation rules and a *main* entry point of the program. A rule is structured into three parts: a precondition, the transformation code (a sequence of statements) and a postcondition. The pre- and postconditions of a rule are two $AFormulae$ which specify respectively a source graph which can be transformed by the rule and the target graph supposed to be produced by the rule.

```
rule $rename$ {
   pre: $(a : A) \wedge (b : B) \wedge (a\ r\ b)$
     $delete(a\ r\ b)$;
     $add(a\ s\ b)$;
   post: $(a : A) \wedge (b : B) \wedge (a\ s\ b) \wedge (a\ \neg r\ b)$
}
rule $reverse$ {
   pre: $(a : A) \wedge (b : B) \wedge (a\ s\ b)$
     $delete(a\ s\ b)$;
     $add(b\ r\ a)$;
   post: $(a : A) \wedge (b : B) \wedge (a\ \neg s\ b)$
}
main {
   assert: $(\exists\ r^{-1}\ A) \subseteq B\ \wedge\ (\exists\ s^{-1}\ A) = \bot$
     $rename$ !;
     $reverse$ !;
   assert: $(\exists\ r^{-1}\ A) = \bot\ \wedge\ (\exists\ r^{-1}\ B) \subseteq A\ \wedge\ (\exists\ s^{-1}\ A) = \bot$
}
```

Fig. 2. Small-t$\mathcal{ALC}$ program $Edges - Reversing$

Rules that are defined separately in a Small-t$\mathcal{ALC}$ program are called sequentially in the *main*. Two types of rule calls are proposed: a simple call (CALL) and an iterative call (CALL!). The first executes the code of the rule if a subgraph in the source graph matches the *ABox* precondition formula. The second executes the code of the rule as long as a subgraph matches the precondition. We can inject $TFormulae$ into *main* to specify the properties of the transformed graph before (*pre-*$TFormula$) and after (*post-*$TFormula$) applications of one or many rules.

We illustrate in Fig. 2 a Small-t$\mathcal{ALC}$ transformation program which reverses all r-edges from A-nodes to B-nodes. This transformation is done in two steps: first the r-edges from A-nodes to B-nodes are transformed into s-edges from A-nodes to B-nodes; then each s-edge from an A-node to a B-node is replaced by a r-edge in the opposite direction from the B-node to the A-node. The program thus is made up of two rules: (1) *rename* which locally renames a r-edge between an A-node a and a B-node b, so that $a\ r\ b$ turns into $a\ s\ b$; (2) $reverse$ which locally replaces a s-edge between an A-node a and a B-node b by a r-edge between b and a so that $a\ s\ b$ turns into $b\ r\ a$. The main of the program calls, in an iterative way, first the rule *rename* then $reverse$.

The question is how to prove that the given program produces the expected states of the graph specified by $TFormulae$. This verification problem will be discussed in the next section.

3 Small-t$\mathcal{ALC}$ Program Verification

We are interested in verifying the correctness of transformation programs, i.e. checking whether a transformation behaves the way it is expected to and produces what it should. Therefore, besides verifying the correctness of each rule, we need to verify that the sequence of rules in the main program is also correct.

3.1 Motivating Example

For instance, consider the program in Fig. 2. To prove that the transformation is correct, the following points must be verified:

1. The correctness of the rules *rename* and $reverse$ with respect to their $ABox$ pre- and postconditions,
2. The correctness of applying iteratively the two rules *rename* and $reverse$ with respect to the $TBox$ assert clauses.

The second point necessitates examining global modifications in the host graph. The properties to be verified in (2) are global because they concern a set of nodes of type A or of type B thus they cannot be expressed with $AFormulae$ but by *TFormulae*. We can specify the transformation program of Fig. 2 as follows: if in the source graph there are r-edges connecting A-nodes to B-nodes and there is not s-edges outgoing from A-nodes, then after applying iteratively the rules *rename* and *reverse*, there are r-edges connecting B-nodes to A-nodes and there is not r-edges nor s-edges outgoing from A-nodes.

More precisely, according to DL definitions [9], the *TFormula* $(\exists\ r^{-1}\ A) \subseteq B\ \wedge\ (\exists\ s^{-1}\ A) = \bot$ asserts that before transformation B-nodes subsume the target nodes of the r-edges outgoing from A-nodes and that the set of target nodes of the s-edges outgoing from A-nodes is empty. From this assumption, we verify after transformation the *TFormula* $(\exists\ r^{-1}\ A) = \bot\ \wedge\ (\exists\ r^{-1}\ B) \subseteq A$ which expresses now that the set of target nodes of the r-edges outgoing from A-nodes is empty and that A-nodes subsume the set of target nodes of the r-edges outgoing from B-nodes. After the transformation, $(\exists\ s^{-1}\ A) = \bot$ stays as an invariant to express the temporary use of s-edges which are created in the rule *rename* are deleted in the rule *reverse*.

This paper focuses on reasoning about global properties on concepts and roles, i.e. properties of the graph as a whole as in (2), that are impacted by application of a set of rules, one or many times. For this purpose, we provide reasoning capabilities not only at rule-level using $AFormulae$ but also at program-level using $TFormulae$. Proving the correctness of a program entails verifying that both the source graph (an interpretation) and the target graph (another interpretation) are models of the $TBox$ and $ABox$. The next sections presents our solution to verify a rule at $ABox$ level and to verify a program at $TBox$ level.

3.2 Rule Verification Using *ABox* Layer

Within a rule, Small-t$\mathcal{ALC}$ uses *AFormulae* to specify graph elements manipulated by the rule's code in the pre- and postconditions. Therefore, only named graph nodes and edges in the current matched graph are concerned. In other words, a rule-level verification allows reasoning only about the local effect when applying once a rule on a matched graph.

Adopting Hoare-like calculus, a prover was developed [7,10] to prove that a Small-t$\mathcal{ALC}$ rule $\{P\}S\{Q\}$ is correct. This verification process is based on an *ABox* static analysis performed in a backward mode in order to compute the weakest precondition (wp) [13]. Each rule statement s of S is assigned to a predicate transformer yielding an $\mathcal{ALCQI}$ formula $wp(s, Q)$ assuming the postcondition Q. The correctness of the code S of a rule with respect to Q is established by proving that the given precondition P implies the weakest precondition.

3.3 Program Verification Using *TBox* Layer

As stated in Sect. 2.3, *TFormulae* are implicitly updated by rules statements that explicitly add and delete individuals and pairs of individuals respectively into and from concepts interpretations. Reasoning about graph global properties when executing a sequence of $ABox$ rules turns into studying the effects of $ABox$ statements on the $TBox$ properties. This results in verifying *TFormulae* of the transformation program in order to check if the graph is correctly transformed as expected.

Using *TFormulae*, we consider an abstract graph that is a superset of the concrete Small-t$\mathcal{ALC}$ graph: properties on nodes are ignored and only properties about the sets of nodes and the sets of source and target nodes of roles are taken into account. Considering such global properties results in losing certain information regarding *AFormulae*. For example, we can not know concretely each pair of connected nodes given the property "all r-edges outgoing from A-nodes go towards B-nodes" i.e. $(\forall\ r^{-1}\ A) \subseteq B$. This abstraction idea and its formalization is called the theory of abstract interpretation [14].

The main question in this paper is how to infer the $TBox$ properties on abstract graphs thus allow verifying a program consisting of a sequence of $ABox$ rules, not only at rule level as done in our previous work. In the next section, we present in detail our solution for this question.

4 Static Analysis by Abstract Interpretation

In order to verify the global state of a graph before and after rule applications, we study the impact of $ABox$ Small-t$\mathcal{ALC}$ statements on a given *TFormula* representing $TBox$ properties. To do so, we analyze the effect of adding (deleting) an element to (from) a set on the set equality and inclusion relationships.

4.1 Interpretation of Small-t$\mathcal{ALC}$ Statements

The aim of our proposed static analysis is to infer a *post-*$TFormula$ on the basis of a given *pre-TFormula* considered as a rule's assumption by interpreting the rules statements in a forward chaining. The inference of a such $TFormula$ is done by studying the effect of *add* and *delete* statements on each $TFact$ in the *pre-*$TFormula$ considering the statement's precondition as hypothesis. For instance, given the $TFact$ $C = D$ in a *pre-TFormula*, adding an individual i to C through the instruction $add(i : C)$ may affect the validity of $C = D$. If i is already an element of C, according to set theory, $add(i : C)$ has no effect on the set C. Consequently, $C = D$ remains valid. However, if i does not belong to C, $add(i : C)$ will add one additional element to C, thus C becomes $C \cup \{i\}$. Consequently, $C = D$ turns into $C \supseteq D$. The $AFact$ $i : C$ can be checked in the precondition of the statement $add(i : C)$.

To clarify the static analysis process, consider the inference of a *post-TFormula* after the call of the rule *rename* with respect to the *pre-*$TFormula$ $(\exists\ r^{-1}\ A) \subseteq B\ \wedge\ (\exists\ s^{-1}\ A) = \bot$ given in the main of Fig. 2. As illustrated in Fig. 3, the pre- and postconditions of the statements are specified by computing the strongest postcondition (*sp*) of the statement from its precondition. The *sp* of a statement expresses most accurately the evolution of the graph being transformed at the $ABox$ level. Taking into account these $ABox$ effects on individual nodes and edges, we want to determine the most precise evolution, at $TBox$ level, of the concepts and edges containing these individuals. The inference of a *post-*$TFormula$ after each statement is done by studying the effect of the statement on the $TFormula$ while taking into account the properties of the manipulated nodes identified in the $ABox$ precondition of the statement. In this example, the statement $delete(a\ r\ b)$ that removes the r-edge between the nodes a and b does not affect any $TFact$ of the *pre-*$TFormula$. In fact, the deletion of the pair (b, a) from the set r^{-1}, knowing that $a : A$ and $b : B$, holds the validity of the inclusion $(\exists\ r^{-1}\ A) \subseteq B$ and does not concern the $TFact$ $(\exists\ s^{-1}\ A) = \bot$ which remains valid. However, adding an s-edge between the nodes a and b, knowing that $a : A$ and $b : B$ from the statement's precondition, transforms the $TFact$ $(\exists\ s^{-1}\ A) = \bot$ into $(\exists\ s^{-1}\ A) \subseteq B$ in the *post-*$TFormula$.

$$\boxed{\begin{array}{ll}
\multicolumn{2}{l}{(\exists\ r^{-1}\ A) \subseteq B \ \wedge\ (\exists\ s^{-1}\ A) = \bot} \\
\quad\quad\quad\ \big| & \{a : A \wedge b : B \wedge a\ r\ b\} \\
\quad\quad\quad\ \big| & \text{delete(a r b);} \\
\quad\quad\quad\ \downarrow & \{a : A \wedge b : B \wedge a\ \neg r\ b\} \\
\multicolumn{2}{l}{(\exists\ r^{-1}\ A) \subseteq B \ \wedge\ (\exists\ s^{-1}\ A) = \bot} \\
\quad\quad\quad\ \big| & \{a : A \wedge b : B \wedge a\ \neg r\ b\} \\
\quad\quad\quad\ \big| & \text{add(a s b);} \\
\quad\quad\quad\ \downarrow & \{a : A \wedge b : B \wedge a\ \neg r\ b \wedge a\ s\ b\} \\
\multicolumn{2}{l}{(\exists\ r^{-1}\ A) \subseteq B \ \wedge\ (\exists\ s^{-1}\ A) \subseteq B}
\end{array}}$$

Fig. 3. Inference of a $TFormula$ after each statement of the rule *rename*

Table 2 summarizes the effect of the statement $add(i : C)$ on both equality and inclusion relationships between concepts. The second column presents the *pre-*$TFact$; the $AFacts$ considered as hypothesis for the interpretation are shown in the third column. The fourth column provides the inferred $TFact$ obtained by the interpretation, so called *post-*$TFact$. In cases where the *pre-*$TFact$ is confirmed as being not valid yet the effect of the statement on the *pre-*$TFact$ can not be deduced, that $TFact$ will be deleted from the $TFormula$. This case is marked in the table by X. For instance, given the *pre-*$TFact$ $C \subseteq D$, adding an instance i to the concept C with i not declared of concept C nor D may make the inclusion not valid. No more informations can be deduced to infer a *post-*$TFact$ so it is deleted from the final $TFormula$ so-called *post-*$TFormula$. Due to the limited number of pages allowed, the tables referring to the others atomic statements as well as the supporting tools are not presented here but available for download[1].

The *select* statement has no effect on a $TFormula$ as it is an assignment of nodes variables. Whereas *if condition then* $s1$ *else* $s2$ is interpreted by transforming the *pre-*$TFormula$ regarding the sequence $s1$ on the one hand, and $s2$ on the other. The result is the disjunction of both of the resulting *post-*$TFormulae$.

The body of the *while* loop is interpreted once, as well as the body of a rule that is called in an iterative way in the main. In fact, whether the interpretation

[1] https://www.irit.fr/~Martin.Strecker/CLIMT/Software/smalltalc.html.

Table 2. Interpretation of the statement $add(i : C)$

Statement	pre-TFact	AFact	post-TFact
$add(i : C)$	$C = \bot$	-	$\neg(C = \bot)$
	$\neg(C = \top)$	$i : C$	$\neg(C = \top)$
		else	X
	$C = D$	$i : C$	$C = D$
		else	$D \subseteq C$
	$C \subseteq D$	$i : C \vee i : D$	$C \subseteq D$
		else	X
	$C \cup D = \bot$	-	$\neg(C \cup D = \bot)$
	$\neg(C \cup D = \top)$	$i : C \vee i : D$	$\neg(C \cup D = \top)$
		else	X
	$C \cup D = E$	$i : C \vee i : D \vee i : E$	$C \cup E = E$
		else	$E \subseteq C \cup D$
	$C \cup D \subseteq E$	$i : C \vee i : D \vee i : E$	$C \cup D \subseteq E$
		else	X
	$C \cap D = \bot$	$i : C \vee i : \neg D$	$C \cap D = \bot$
		else	$\neg(C \cap D = \bot)$
	$\neg(C \cap D = \top)$	$i : C \vee i : \neg D$	$\neg(C \cap D = \top)$
		else	X
	$C \cap D = E$	$i : C \vee i : E \vee i : \neg D$	$C \cap D = E$
		else	$E \subseteq C \cap D$
	$C \cap D \subseteq E$	$i : C \vee i : E \vee i : \neg D$	$C \cap D \subseteq E$
		else	X
	$(\exists\ r\ C) = D$	-	$D \subseteq (\exists\ r\ C)$
	$(\exists\ r\ C) \subseteq D$	-	X
	$(\exists\ r^{-1}\ C) = D$	$i : (= 0\ r\ \neg D)$	$(\exists\ r^{-1}\ C) = D$
		else	$D \subseteq (\exists\ r^{-1}\ C)$
	$(\exists\ r^{-1}\ C) \subseteq D$	$i : (= 0\ r\ \neg D)$	$(\exists\ r^{-1}\ C) \subseteq D$
		else	X

of the same sequence of statements is done one or several times, the resulting $TFormula$ remains the same as Small-t$\mathcal{ALC}$ statements are limited to adding and deleting elements to and from sets as already mentioned. For instance, consider a $TFact$ $C = D$ and a statement that adds repeatedly a selected instance d to the concept D. By interpreting the statement for the first time, $C = D$ turns into $C \subseteq D$. Adding other elements d to the concept D maintains the validity of the $TFact$ $C \subseteq D$. Consequently, the traditional widening operator of the abstract interpretation, which guarantees termination when applied to increasing sequences [14], is simpler in our context than in programs employing non symbolic operations.

4.2 Soundness of the Static Analysis

Deriving a $TFormula$ for Small-t$\mathcal{ALC}$ programs does not guarantee that our static verification calculus is sound. Given a correctness formula $\vdash \{P\}S\{Q\}$, we need to show that the proposition $\models \{P\}S\{Q\}$ about the semantics of the correctness formula holds. This entails to consider $\models \{P\}S\{Q\}$ as a new judgment based on state updates meaning that the program S when invoked in the state σ will terminate in the state τ. We denote $S(\sigma, \tau)$ this relation and define $\models \{P\}S\{Q\}$ as $\forall\sigma.P(\sigma) \Rightarrow (\exists\tau.S(\sigma, \tau) \wedge Q(\tau))$. Proof is done on the derivation of Hoare correctness formulae considering Small-t$\mathcal{ALC}$ operational semantics.

Let us consider $S = add(i : C)$, inspired from the assignment statement $V := E$ in imperative languages for which $sp(V := E, P) = \exists V'.P[V'\backslash V] \wedge (V = E[V'\backslash V])$, we compute $sp(add(i : C), P)$ as a substitution: $sp(add(i : C), P) = \exists C'.P[C'\backslash C] \wedge (C' + i\backslash C)$. If the formula $sp(add(i : C), P) \Rightarrow Q$ is valid, then for all source graphs G verifying $P(\sigma)$ we conclude $Q(\sigma')$ for target graphs G' where σ' denotes the state σ updated by the action add.

In the following, we prove two inferences about adding a node i to a concept C. The first one is basic and corresponds to the first line of Table 2. Suppose a state $\sigma = (C = \bot)$. Then $sp(add(i : C), C = \bot) = \exists C'.(C = \bot)[C'\backslash C] \wedge (C' + i\backslash C)$, that is $C' = \bot \wedge C = C' + i$ which implies $\sigma' = \neg(C = \bot)$. As $P(\sigma)$ is true for G, we have $Q(\sigma')$ for G'. Thus, $P(\sigma) \Rightarrow S(\sigma, \sigma') \wedge Q(\sigma')$. This case is quite straightforward because it does not presuppose any $AFact$ for P.

The second one assumes the precondition $C = D$, as indicated by the third line of Table 2. We aim at knowing when this relation of subsumption $TBox$ between the concepts C and D is also a postcondition of the substitution $[C' + i\backslash C]$ related to $add(i : C)$. The outcome of this question depends on whether the individual i belongs to concept C. If $\sigma = (C = D \wedge i \in C)$, we can conclude that $\sigma' = (C = D)$, otherwise, $\sigma = (C = D \wedge i \notin C)$ is transformed into $\sigma' = (C \supseteq D)$. In the first case, we have $sp(add(i : C), C = D \wedge i \in C) = \exists C'.(C = D \wedge i \in C)[C'\backslash C] \wedge (C' + i\backslash C)$, i.e. $C' = D \wedge i \in C' \wedge C = C' + i$ which implies $C = D$, because $i \in C' \wedge C = C' + i \Rightarrow C = C'$. On the other hand, when $i \notin C$, $sp(add(i : C), C = D \ \wedge \ i \notin C) = C' = D \ \wedge \ i \notin C' \wedge C = C' + i$ which implies $C \supseteq D$, because $i \notin C' \wedge C = C' + i \Rightarrow C \supseteq C'$. As previously, and in both cases, as $P(\sigma)$ is true for G, we have $Q(\sigma')$ for G'. Thus, $P(\sigma) \Rightarrow S(\sigma, \sigma') \wedge Q(\sigma')$.

We can prove the other lines of Table 2 similarly, considering the $ABox$ substitutions of the language and the $TFormulae$ involved.

5 Relation Between the ABox/TBox Verifications

The purposes of $ABox$ and $TBox$ verifications differ. $TBox$ verification aims to verify concepts inclusion relationships (universal assertions), whereas $ABox$ verification is more about fact-checking and instance-checking (membership assertions). In terms of program verification, they are complementary. However, these two components are undoubtedly dependent.

5.1 Dependence Between the ABox/TBox Verifications

Inferring $TFormulae$ does not consider only rules statements, but takes into account rule specifications on instances properties too. Therefore, weakening $AFormulae$ has a direct effect on the process of inferring $TFormulae$. In case where instances properties are not revealed in the precondition, some properties on sets may not be proven to be valid and so are discarded from the *post*-$TFormula$.

For instance, consider the program of Fig. 4 consisting of the rule *replace* which replaces a r-edge between two nodes a and b with s. The precondition of the rule asserts that b is a B-node, however, it does not inform about the concept of a. Note that this rule is proven to be correct by the Small-t$\mathcal{ALC}$ prover.

```
rule replace {
  pre: $(a\ r\ b) \wedge (b : B)$
  // strengthened pre: $(a\ r\ b) \wedge (b : B) \wedge (\mathbf{a : A})$
    delete$(a\ r\ b)$;
    add$(a\ s\ b)$;
  post: $(a\ s\ b) \wedge (b : B)$
}
main {
  assert: $(\exists\ s^{-1}\ A) = \bot$
    replace;
  assert: $(\exists\ s^{-1}\ A) \subseteq B$
}
```

Fig. 4. Example of inconsistent $TFormulae$

Consider the $TFact$ $(\exists\ s^{-1}\ A) = \bot$ before the rule call expressing that there is no s-edges outgoing from A-nodes. Aiming for verifying after the rule call that edges outgoing from A-nodes are going towards B-nodes i.e. $(\exists\ s^{-1}\ A) \subseteq B$, the static analyzer studies the effect of the rule *replace*. So it interprets firstly the statement $delete(a\ r\ b)$ which does not affect the validity of $(\exists\ s^{-1}\ A) = \bot$, and secondly the statement $add(a\ s\ b)$ which certainly does because the given $TFact$ concerns the added s-edge. In this case, the static analyzer shows that the $TFact$ $(\exists\ s^{-1}\ A) = \bot$ is unsatisfiable, but does not infer any other fact since the concept of a is unknown (corresponding to a case X in the interpretation table of the statement $add(i\ r\ j)$). Hence, the given $TFact$ $(\exists\ s^{-1}\ A) \subseteq B$ is supposed inconsistent with $ABox$ assertions of the rule *replace*.

Now suppose that the developer asserts as well in the precondition of the rule that a is an A-node as shown in bold on Fig. 4. In this case, the static analyzer would deduce that $(\exists\ s^{-1}\ A) \subseteq B$. We can conclude that the more strengthened $AFormulae$ are, the more the diagnostic of $TFormulae$ gets refined.

5.2 Complementarity Between the ABox/TBox Verifications

Verification of a rule's triple using Hoare logic guarantees a correct transformation of the manipulated nodes. At a more abstract level, verification of the $TBox$ checks the effect of the rules on the graph as a whole. These two verification levels are complementary: each level verifies properties that can not be expressed by the other one.

Let us reconsider the program in Fig. 2 consisting in reversing r-edges outgoing from an A-node a towards B-nodes. Suppose now that the developer makes an error in the rule *reverse* by writing the statement $add(a\ r\ b)$ instead of $add(b\ r\ a)$ as shown in bold in Fig. 5. In this case, the rule renames each s-edge to r-edge without reversing it. Consequently, applying the sequence $\{rename!, reverse!\}$ on a graph will produce a target graph identical to the source graph.

```
rule $reverse$ {
   pre: $(a : A) \wedge (b : B) \wedge (a\ s\ b)$
     $delete(a\ s\ b)$;
     $add(a\ r\ b)$;
    // correct statement: add(b r a);
   post: $(a : A) \wedge (b : B) \wedge (a\ \neg s\ b)$
}
```

Fig. 5. Incorrect rule *reverse*

The rule *reverse* is proven as a correct Hoare-triple by the Small-t$\mathcal{ALC}$ prover, i.e. the rule's code ensures the postcondition with the given precondition. This happens because the rule's postcondition is weak: it checks only the concept of a and the nonexistence of s-edges outgoing from a. However, exploiting the Small-t$\mathcal{ALC}$ $TBox$ static analyzer to verify the *post-TFormula* given in the *main* program, we notice that the $TFact$ $(\exists\ r^{-1}\ A) = \bot$, which expresses that there is not a r-edges outgoing from A-nodes, is unsatisfiable.

Warned by the result of the $TBox$ verification, the developer strengthens the postcondition of the rule *reverse* with the $AFact$ $a : (= 0\ r\ B)$ to check that no r-edge is outgoing from a. Now the prover fails to verify the rule with the modified postcondition. The developer then realizes that b must be connected by r to a.

Since one is allowed to write weaken specifications of a code while maintaining the validity of a rule's triple at the $ABox$ level, a given postcondition may not reveal all the properties of the transformed instances to be verified yet yield to a correct triple. In those cases, using $TBox$ verification with *TFormulae* can identify an abnormal effect on the graph.

On the other hand, verifying exclusively that the given *TFormulae* are consistent with the global graph does not attest actually that rules triples are written correct since the $TBox$ verification infer $TFormulae$ from rules supposed correct. Hence, it is necessary as well to prove rules triples using Hoare logic by writing complete specifications to get tangible results.

Ultimately, each of the $ABox$ and $TBox$ verifications has different level of verification and so are complement. $ABox$ checks whether instances that are manipulated in a rule are locally transformed. $TBox$ checks the effect of instances transformation on the abstract graph. Hence, errors that are not identified by one level, can be identified by the other.

6 Verifying Monadic Second-Order Properties

Verifying rules using the Hoare logic with $ABox$ assertions on individuals is limited for checking local properties of the graph. With quantification over sets, $TBox$ assertions can express global properties of graphs and can be exploited to verify some monadic second-order (MSO) properties [6].

For instance, consider the problem of verifying that a graph is bipartite i.e. a graph that is colored in two colors e.g. A and B, and in which every edge connects a node of A to one of B. Figure 6 shows the Small-t$\mathcal{ALC}$ rule *grow* that allows connecting, with a r-edge, two nodes belonging to two different concepts. The bipartiteness property can be expressed in the Small-t$\mathcal{ALC}$ *TFormulae* by two $TFacts$: $(\exists\ r\ A) \cap A = \bot$ to verify that the set of source nodes of the r-edges going towards A and A are disjoint, and $(\exists\ r\ B) \cap B = \bot$ to verify that the set of source nodes of the r-edges going towards B and B are disjoint. To close off the possibility to add a r-edge outgoing from nodes belonging to other concepts than A or B, closure axioms are necessary: $A \cup B = \top \wedge A \cap B = \bot$ i.e. all the graph's nodes are exclusively of concept A or of concept B. This *TBox* invariant expressing a global property of the graph can be checked before and after calling iteratively the rule *grow*.

```
rule $grow$ {
  pre: $x : (= 0\ r\ A \cup B)$
    $if(x : A)\ then$
      $select\ y\ with\ y : B;$
    $else$
      $select\ y\ with\ y : A;$
    $add(x\ r\ y);$
  post: $x : (= 1\ r\ A \cup B)$
}
main {
  assert: $(\exists\ r\ A) \cap A = \bot\ \wedge\ (\exists\ r\ B) \cap B = \bot \wedge\ A \cup B = \top \wedge A \cap B = \bot$
    $grow!;$
  assert: $(\exists\ r\ A) \cap A = \bot\ \wedge\ (\exists\ r\ B) \cap B = \bot \wedge\ A \cup B = \top \wedge A \cap B = \bot$
}
```

Fig. 6. Small-t$\mathcal{ALC}$ program making up a bipartite graph

Our $TBox$ abstraction level neglects the source and target nodes of an edge. Hence, our current work is not able to express directly MSO properties related to connectivity of a graph. We envisage increasing the expressiveness of $TBox$ formulae by choosing a richer description logic, notably which offer role constructors and role connectors such as inclusion and transitivity.

7 Related Work

In the theory of algebraic graph transformations, Habel and Pennemann [1] defined nested application conditions to describe graph properties. However these first-order tailored logic formulae need to be derived into specific inference rules in order to provide a specific theorem-proving that suits them. This approach has been adopted by the graph transformation language GP [15] which provides a Hoare-like calculus. Nested conditions of GP have been recently extended to MSO properties on graphs by introducing new quantifiers for set variables of nodes and edges and having morphisms with constraints about set membership [5].

The algebraic approach has also given rise to the dedicated logic for graph properties, called Graph Pattern Logic [16] and Navigational Logic [17], which consider that a graph pattern P is just an object in the category of graphs. Thereby, a global property for a graph G can be reduced to identifying a morphism from P to G. The authors have invested patterns dedicated to graph paths between nodes. We share with them the idea that reasoning mechanisms are supported by the underlying logic.

The static satisfiability of a DL knowledge base updated by a finite sequence of insertions and deletions performed on concepts and roles has been studied by Calvanese et al. [18]. The authors introduce a simple imperative language with the basic actions $A \oplus C$ and $A \ominus C$ on an interpretation $\mathcal{I}$ for concepts A and C. $A \oplus C$ stands for the addition of the content of $C^{\mathcal{I}}$ to $A^{\mathcal{I}}$ and $A \ominus C$ represents the removal of $C^{\mathcal{I}}$ from $A^{\mathcal{I}}$.

In order to capture the action effects on a DL knowledge base $\mathcal{K}$, a transformation $TR(\mathcal{K})$ associated to each action has been defined. This transformation on a finite interpretation domain enables to reduce static verification to finite satisfiability of $\mathcal{K}$: $TR(\mathcal{K})$ is $\mathcal{K}$-preserving if there exists a model when applying $TR(\mathcal{K})$ on interpretations. Transformations allow to modify labels of sets of nodes instead of individuals. Constraints on interpretations coding graph-structured data are expressed by specific $\mathcal{ALCHOIQ}$br DL formulae, including nominals ($\mathcal{O}$) which enables modifying single node labeling.

Dynamic logics [19] are well suited for dealing about properties of evolving data. J. H. Brenas et al. [20] investigate such logics for graph transformations and define $\mathcal{C2PDLS}$, a combination of both combinatory and converse propositional dynamic logics, augmented by substitutions. The main idea is to split the nodes of the considered graphs into two sets: one contains the nodes before substitutions take place; the other stores nodes that will be created by future transformations and those that have been deleted in the past. This separation allows some reasoning on reachability properties considering named nodes.

In our Small-t$\mathcal{ALC}$ context, $ABox$ updates do not represent changes or refinements in the conceptualization of $TBox$ axioms. We allow adding and deleting individuals and roles in an imperative style with extensional $ABox$ rules, while provide a mechanism to infer intentional $TBox$ knowledge which is consistent with $ABox$ changes. From the user point of view, we share the same desired effect called projection in action-oriented paradigm, i.e. knowing whether an assertion that one wants to make true really holds after executing a rule [21].

8 Conclusion and Future Work

Our logic-based graph transformation language Small-t$\mathcal{ALC}$ allows to reason on graph transformations and verify local and global properties of graphs by exploiting $ABox$ and $TBox$ levels of description logic respectively. The properties of nodes manipulated in each rule are expressed in $ABox$ pre- and postconditions so that a Hoare-like calculus can be realized to verify the correctness of a rule. Besides this $ABox$ verification, we presented an approach based on a static analysis aiming to deduce implicit $TBox$ assertions about concepts from explicit $ABox$ assertions and valid $TBox$ premises. Our $TBox$ verification process determines whether the given $ABox$ and $TBox$ assertions are consistent. A formal proof sketch of our static algorithm has been addressed.

We showed that using $TFormulae$, some monadic second-order properties can be verified. It would be interesting as future work to improve the expressiveness of our $TFormulae$ in such a way that more global properties can be verified e.g. considering the cardinality restrictions and roles constructors.

Other dialects and in particular DL $\mathcal{ALCQIO}$ with nominals $\mathcal{O}$ which allows the description of concepts by the enumeration of named individuals can be considered as well. The key is to work out how we can increase the expressivity of Small-t$\mathcal{ALC}$ programs in order to be able to prove more interesting specifications. We also investigate Small-t$\mathcal{ALC}$ functionalities to manage explicit and inalterable $TBox$ axioms now given by the end-user.

References

1. Habel, A., Pennemann, K.-H.: Correctness of high-level transformation systems relative to nested conditions. Math. Struct. Comput. Sci. **19**(2), 245–296 (2009). https://doi.org/10.1017/S0960129508007202
2. Rensink, A.: Representing first-order logic using graphs. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 319–335. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30203-2_23
3. Orejas, F., Ehrig, H., Prange, U.: A logic of graph constraints. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 179–198. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78743-3_14
4. Lambers, L., Orejas, F.: Tableau-based reasoning for graph properties. In: Giese, H., König, B. (eds.) Graph Transformation, pp. 17–32. Springer, Cham (2014)
5. Poskitt, C.M., Plump, D.: Verifying monadic second-order properties of graph programs. In: Giese, H., König, B. (eds.) Graph Transformation, pp. 33–48. Springer, Cham (2014)

6. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations, pp. 313–400 (1997)
7. Strecker, M.: Modeling and verifying graph transformations in proof assistants. Electron. Notes Theoret. Comput. Sci. **203**(1), 135–148 (2008)
8. Baklanova, N., et al.: Coding, executing and verifying graph transformations with small-t$\mathcal{ALCQ}$e. In: 7th International Workshop on Graph Computation Models(GCM) (2016). http://gcm2016.inf.uni-due.de/
9. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York (2003)
10. Makhlouf, A., Percebois, C., Tran, H.N.: An auto-active approach to develop correct logic-based graph transformations. Int. J. Adv. Softw. **11**(1,2), 147–158 (2018) http://oatao.univ-toulouse.fr/22689/
11. Sattler, U.: Reasoning in description logics: basics, extensions, and relatives. In: Antoniou, G., et al. (eds.) Reasoning Web 2007. LNCS, vol. 4636, pp. 154–182. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74615-7_2
12. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On instance-level update and erasure in description logic ontologies. J. Logic Comput. **19**(5), 745–770 (2009)
13. Dijkstra, E.W., Scholten, C.S.: Predicate Calculus and Program Semantics. Springer, New York (1990). https://doi.org/10.1007/978-1-4612-3228-5
14. Cousot, P.: Abstract interpretation based formal methods and future challenges. In: Wilhelm, R. (ed.) Informatics. LNCS, vol. 2000, pp. 138–156. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44577-3_10
15. Poskitt, C.M., Plump, D.: Hoare-style verification of graph programs. Fundam. Inform. **118**, 135–175 (2012)
16. Navarro, M., Pino, E., Orejas, F., Lambers, L.: A logic of graph conditions extended with paths. In: Pre-proceedings 7th International Workshop on Graph Computation Models (2016). http://gcm2016.inf.uni-due.de/pre-proceedings.html
17. Lambers, L., Navarro, M., Orejas, F., Pino, E.: Towards a navigational logic for graphical structures. In: Heckel, R., Taentzer, G. (eds.) Graph Transformation, Specifications, and Nets. LNCS, vol. 10800, pp. 124–141. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75396-6_7
18. Ahmetaj, S., Calvanese, D., Ortiz, M., Simkus, M.: Managing change in graph-structured data using description logics. ACM Trans. Comput. Logic **18**(4), 27:1–27:35 (2017). https://doi.org/10.1145/3143803
19. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. In: Gabbay, D.M., Guenthner, F. (eds.) Handbook of Philosophical Logic. Handbook of Philosophical Logic, vol. 4, pp. 99–217. Springer, Dordrecht (2002). https://doi.org/10.1007/978-94-017-0456-4_2
20. Brenas, J.H., Echahed, R., Strecker, M.: C2PDLS: a combination of combinatory and converse PDL with substitutions. In: Gammarth, T., Mosbah, M., Rusinowitch, M. (eds.) 2017 the 8th International Symposium on Symbolic Computation in Software Science, SCSS 2017, 6–9 April 2017, pp. 29–41 (2017). https://easychair.org/publications/paper/dx4z
21. Liu, H., Lutz, C., Miličić, M., Wolter, F.: Reasoning about actions using description logics with general TBoxes. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 266–279. Springer, Heidelberg (2006). https://doi.org/10.1007/11853886_23