Resolution-Like Theorem Proving for High-Level Conditions*

Karl-Heinz Pennemann

University of Oldenburg, Germany
pennemann@informatik.uni-oldenburg.de

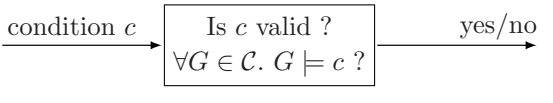
Abstract. The tautology problem is the problem to prove the validity of statements. In this paper, we present a calculus for this undecidable problem on graphical conditions, prove its soundness, investigate the necessity of each deduction rule, and discuss practical aspects concerning an implementation. As we use the framework of weak adhesive HLR categories, the calculus is applicable to a number of replacement capable structures, such as Petri-Nets, graphs or hypergraphs.

Keywords: first-order tautology problem, high-level conditions, theorem proving, resolution, weak adhesive HLR categories.

1 Introduction

(High-level) Conditions are a graphical formalism to specify valid objects as well as morphisms, i.e., they can be used to describe system or program states as well as specify matches for transformation rules. They provide an intuitive formalism for structural properties and are well suited for reasoning about the behavior of transformation systems.

Our goal is to decide the correctness of graphical specifications consisting of a precondition, a program [HP01, HPR06] and a postcondition. A classical approach [DS89] to this problem is the proof or refutation that the precondition implies the weakest precondition [HPR06] of the program and the postcondition. To decide such an implication is a special instance of the tautology problem: the decision whether or not a claimed statement is valid for all possible objects.



For the category of finite, directed, labeled graphs, conditions are expressively equivalent [Ren04, HP08] to first order logic on graphs [Cou90]. Consequently, the tautology problem for arbitrary conditions over arbitrary categories is not decidable, i.e., there does not exist an algorithm that decides the validity of

* This work is supported by the German Research Foundation (DFG), grants GRK 1076/1 (Graduate School on Trustworthy Software Systems) and HA 2936/2 (Development of Correct Graph Transformation Systems).

H. Ehrig et al. (Eds.): ICGT 2008, LNCS 5214, pp. 289–304, 2008.

arbitrary conditions over arbitrary categories. In the context of finite graphs, the tautology problem is not even semi-decidable: while it is possible to search for proofs, one is not guaranteed to find one, even if there is no (finite) graph that does not satisfy the given condition.

In the case of graphs, the translation of conditions into first order logic [HP08] enables to solve the tautology problem using existing first-order theorem provers such as Vampire [RV02], Darwin [BFT06] or Prover9 [McC08]. However, 7 out of 74 example tautologies generated from correct program specifications of the "access control" example in [HPR06] cannot be solved by any of the aforementioned tools, given 1 hour time per tautology (Intel T5600, 1.83GHz). One reason for this is that first-order theorem provers need to be restricted to graphs via axioms that become part of the problem to be solved. In contrast, a theorem prover based on conditions would be restricted to the considered category in a natural, constructive way. This property in combination with other advantages makes it worthwhile to investigate a theorem prover dedicated to conditions.

In this paper, we present a calculus for conditions over adhesive high-level replacement categories. Taking resolution [Rob65], the most successful approach to first-order theorem proving, as an ideal, we present six deduction rules able to refute conditions over graphs and graph-like structures in conjunctive normal form. We show that every rule application corresponds to a logical deduction, and investigate if omission of any rule leads to an incomplete calculus. We discuss practical aspects concerning an implementation such as filtering out structurally equivalent conditions, and briefly compare our results with related work, e.g. Koch et. al. [KMP05] and Orejas et. al. [OEP08].

The paper is organized as follows. In Section 2, the definition of conditions is reviewed and examples are given. In Section 3, the calculus is presented, and its soundness is shown. In Section 4, we discuss practical aspects concerning an implementation. We briefly relate our results to other work in Section 5. A conclusion including further work is given in Section 6.

2 Conditions

In this section, we recall the definition of conditions on graphs and graph-like structures. To abstract from a specific structure, we use the framework of weak adhesive HLR categories. A detailed introduction can be found in [EEPT06].

Assumption 1. *Assume that $\langle \mathcal{C}, \mathcal{M} \rangle$ is a weak adhesive HLR category consisting of a category $\mathcal{C}$ of objects and a class $\mathcal{M}$ of monomorphisms. Additionally, we require*

- *an* $\mathcal{M}$-initial object *I, i.e., an object $I \in \mathcal{C}$ such that, for every object $G \in \mathcal{C}$, there exists a unique morphism $i_G \colon I \to G$ and i_G is in $\mathcal{M}$,*
- epi-$\mathcal{M}$-factorization*, i.e., for every morphism there is an epi-mono-factorization with monomorphism in $\mathcal{M}$,*

- *a* finite number of $\mathcal{M}$-morphisms, *i.e. for every objects* G, H, *there exists only a finite number of morphisms* $G \hookrightarrow H$ *in* $\mathcal{M}$ *(up to isomorphism),*
- *a* finite number of epimorphisms *for any domain* G, *i.e. for every object* G, *there is only a finite number of epimorphisms* $e\colon G \to H$ *(up to isomorphism).*

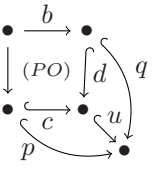
The last two requirements ensure the effectiveness of the constructions in this paper. From now on, morphisms in $\mathcal{C}$ are simply referred to as morphisms.

Example 1. The category **Graph** of finite, directed, labeled graphs together with the class $\mathcal{M}$ of all injective graph morphisms constitutes a weak adhesive HLR category [EEPT06] satisfying the assumptions. The empty graph $\emptyset$ is the $\mathcal{M}$-initial object.

Notation. A morphism m with domain A and codomain B is denoted by $m\colon A \to B$. We write "$\hookrightarrow$" instead of "$\to$" to indicate that the morphism is in $\mathcal{M}$. For graph morphisms, the mapping of nodes is depicted by indices, if necessary.

For a certain rule in our calculus, we require the notion of $\mathcal{M}$-pushout. An $\mathcal{M}$-pushout is a special pushout for which it can be guaranteed that the unique morphism u is in $\mathcal{M}$, if the commutative morphisms p, q are in $\mathcal{M}$.

Definition 1 ($\mathcal{M}$-pushout). *A pushout* $c \circ a = d \circ b$ *with* $c, d \in \mathcal{M}$ *is called* $\mathcal{M}$-pushout, *if for all morphisms* p, q *with* $p \circ a = q \circ b$, *the unique existing morphism* u *with* $p = c \circ u$ *and* $q = d \circ u$ *is in* $\mathcal{M}$.



We use the following characterization of $\mathcal{M}$-pushouts.

Fact 1 ($\mathcal{M}$-pushout). *A pushout* $c \circ a = d \circ b$ *with* $c, d \in \mathcal{M}$ *is an* $\mathcal{M}$-pushout, *if and only if for all epimorphisms* e *with* $\mathrm{dom}(e) = \mathrm{codom}(d)$ *we have* $e \notin \mathcal{M}$ *implies* $e \circ c \notin \mathcal{M}$ *or* $e \circ d \notin \mathcal{M}$.

Proof. Via epi-$\mathcal{M}$-factorization using the converse statement.

$e \notin \mathcal{M}$	implies $(e \circ c \notin \mathcal{M}$ or $e \circ d \notin \mathcal{M})$	(characterization)
$\Leftrightarrow$ not $(e \circ c \notin \mathcal{M}$ or $e \circ d \notin \mathcal{M})$	implies not $e \notin \mathcal{M}$	(converse)
$\Leftrightarrow (e \circ c \in \mathcal{M}$ and $e \circ d \in \mathcal{M})$	implies $e \in \mathcal{M}$	(deMorgan)
$\Leftrightarrow (m \circ e \circ c \in \mathcal{M}$ and $m \circ e \circ d \in \mathcal{M})$	implies $m \circ e \in \mathcal{M}$	($m \in \mathcal{M}$, $\mathcal{M}$ closed under comp./decomp.)
$\Leftrightarrow (u \circ c \in \mathcal{M}$ and $u \circ d \in \mathcal{M})$	implies $u \in \mathcal{M}$	(epi-$\mathcal{M}$-factorization)
$\Leftrightarrow (p \in \mathcal{M}$ and $q \in \mathcal{M})$	implies $u \in \mathcal{M}$	(commutativity)

Example 2 (access control graphs). In the following, we present state graphs of a simple access control for computer systems, which abstracts authentication and models user and session management in a simple way. We use this example solely for illustrative purposes. A more elaborated, role-based access control model is considered in [KMP05]. The basic items of our model are users, sessions, logs, computer systems, and directed edges between those items. An edge between a user and a system represents that the user has the right to access the system, i.e. to establish a session with the system. Every user node is connected with one log node, while an edge from a log to the system represents a

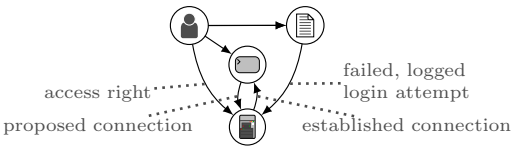


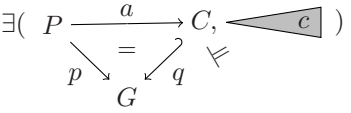
Fig. 1. The type graph of the access control system

failed (logged) login attempt. Every session is connected to a user and a system. The direction of the latter edge differentiates between sessions that have been proposed (an outgoing edge from a session node to a system) and sessions that have been established (an incoming edge to a session node from a system).

Conditions are nested constraints and application conditions generalizing the corresponding notions in [HW95, EEHP06] along the lines of [Ren04].

Definition 2 (conditions). *A* (nested) condition *over an object P is of the form* true *or $\exists(a, c)$, where $a\colon P \to C$ is a morphism and c is a condition over C. Moreover, Boolean formulas over conditions over P yield conditions over P, i.e., $\neg c$ and $\wedge_{j\in J} c_j$ are (Boolean) conditions over P, where J is a finite index set and c, $(c_j)_{j\in J}$ are conditions over P. Additionally, $\exists a$ abbreviates $\exists(a, \text{true})$, $\forall(a, c)$ abbreviates $\neg\exists(a, \neg c)$,* false *abbreviates* $\neg$true*, $\vee_{j\in J} c_j$ abbreviates $\neg\wedge_{j\in J} \neg c_j$ and $c \Rightarrow d$ abbreviates $\neg c \vee d$.*

Every object and morphism satisfies true*. A morphism p* satisfies *a condition $\exists(a, c)$, if there exists a morphism q in $\mathcal{M}$ such that $q \circ a = p$ and q satisfies c.*



An object G satisfies *a condition $\exists(a, c)$, if the condition is over the initial object I and the initial morphism $i_G\colon I \to G$ satisfies the condition. The satisfaction of conditions by objects and morphisms is extended onto Boolean conditions in the usual way. We write $G \models c$ resp. $p \models c$ to denote that the object G resp. the morphism p satisfies c. For two conditions c, d over C, d is a* consequence *or* logical deduction *of c, written $c \models d$, if for all morphisms p in $\mathcal{M}$ with domain C, $p \models c$ implies $p \models d$. Two conditions c and c' are* equivalent*, denoted by $c \equiv c'$, if for all morphisms p in $\mathcal{M}$, $p \models c$ iff $p \models c'$.*

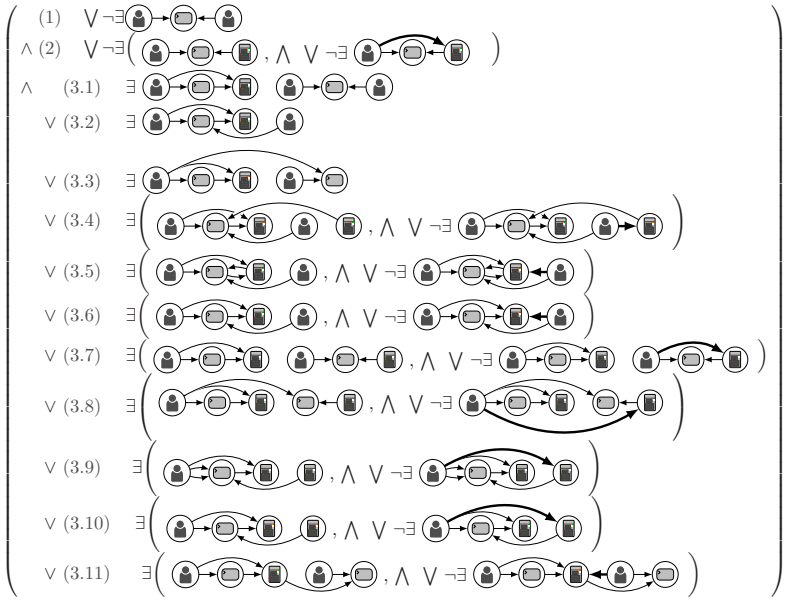
In the context of objects, conditions (over the initial object I) are also called *constraints*.

Notation. For every morphism $a\colon P \to C$ in a condition, we just depict the codomain C, if the domain P can be unambiguously inferred. This is the case for constraints, which are by definition conditions over I. For instance, the constraint $\forall(\emptyset \to \bigcirc_1,\ \exists(\bigcirc_1 \to \bigcirc_1 \to \bigcirc_2))$ with the meaning "Every node has an outgoing edge to another distinct node" can be represented by $\forall(\bigcirc_1,\ \exists(\bigcirc_1 \to \bigcirc_2))$.

Example 3 (access control conditions). Consider the access control graphs introduced in Example 2. Conditions allow to formulate statements on the graphs of the access control and can be combined to form more complex statements. The following conditions are over the empty graph:

$\exists($ $)$	A session is proposed
$\exists($ $)$	A session is established
$\forall($  $, \exists($  $) \vee \exists($  $))$	Every session is either proposed or established
$\neg\exists($  $)$	No session is shared between two users
$\forall($  $, \exists($ $))$	Every session is associated to a user
$\forall($  $, \exists($ $))$	Every user that is logged into a system, has an access right.

Example 4. Consider the access control graphs introduced in Example 2. The dynamic part of the access control system is the reflexive, transitive closure of a non-deterministic choice of programs such as the addition and removal of users, the grant and revocation of access rights and a login and logout procedure. See [HPR06] for a complete overview. We exemplarily consider the transformation rule `Access` $= \langle$  $\Rightarrow$  $\rangle$ which is a part of the login procedure. Such a rule consists of a left-hand side expressing the prerequisites "If a user proposes a session to a system for which he has the appropriate access right" and the local effect of the rule's application "Then this proposed session is accepted and becomes established". Our goal is to show that `Access` preserves the satisfiability of the condition $\forall($  $, \exists($  $)) \wedge \neg\exists($  $)$. By construction of a weakest precondition [HPR06], the problem reduces to prove that



is a tautology.

For rules in our calculus such as (Lift), we assume conditions to be in $\mathcal{M}$*-normal form* ($\mathcal{M}$NF), i.e. if for all subconditions $\exists(a, c)$ the morphism a is in $\mathcal{M}$.

Fact 2 ($\mathcal{M}$-normal form). *Every condition c over P can be transformed into a condition c' in $\mathcal{M}NF$ such that $c \equiv c'$.*

Proof. $\exists(a, c) \equiv \exists a \equiv \text{false}$, if $a \notin \mathcal{M}$ (see [HP08]).

We now define the notion of a non-negated subcondition, which we use later to restrict the applicability of deduction rules.

Definition 3 (non-negated subcondition). *A condition c is a* non-negated subcondition *of a condition d, if $c = d$ or if d is of the form $\exists(a, e)$ or $(e \wedge e')$ or $(e \vee e')$ and c is a non-negated subcondition of e or e'.*

3 Proving High-Level Conditions

The tautology or validity problem is the fundamental problem of deciding whether or not a claimed statement is true for all possible objects.

Definition 4 (tautology problem). *Given a category $\mathcal{C}$, the* tautology problem *is the problem to decide for any condition c, whether or not forall $G \in \mathcal{C}$, $G \models c$.*

We write "$\models c$" if c is a tautology and "$\not\models c$" if c is not a tautology. A straightforward approach to answer the tautology problem for a condition c is to prove "true $\models c$". This can be done by constructing a proof chain "true $\models \ldots \models c$", starting without any assumptions (true), yielding in logical deductions the given condition c. Instead of constructing such a proof top-down, resolution follows a more target-oriented view and considers the complementary problem of refuting the negated condition "$\neg c$". In this case, the goal is to find a refutation "$\neg c \models \ldots \models$ false".

After negation of an input F, a resolution-based algorithm on formulas would transform the negated statement $\neg F$ into prenex normal form and skolemize to yield clauses. However neither does there exists a comparable normal form for conditions, nor is skolemization possible for a given category such as **Graph**: Skolemization requires the introduction of fresh function symbols of unbounded arity, for which there seems no equivalent operation for a fixed structure. Nevertheless, it is possible to transform the condition $\neg c$ into conjunctive normal form.

Definition 5 (conjunctive normal form). *Condition* true *is in* conjunctive normal form (CNF). *Every condition $\wedge_{j \in J} \vee_{k \in K_j} c_k$ is in CNF, if for every $j \in J$ and every $k \in K_j$, $c_k = \exists(a_k, d_k)$ or $c_k = \neg\exists(a_k, d_k)$ for some morphism a and some condition d_k in CNF.*

Given a condition in CNF, the actual resolution process begins and adds derived facts (disjunctions) to the conjunction. The goal is the addition of false as conjunct. If a true resolution calculus were possible for conditions, each refutation step "$\models$" would be of the form:

- *Select* two disjunctions $(\neg\exists(a,c)\vee c_1)$ and $(\exists(b,d)\vee c_2)$ from the conjunction such that $\exists(b,d)\models\exists(a,c)$.
- *Add* the *resolvent* $(c_1\vee c_2)$ to the conjunction.

Special case: if c_1 and c_2 do not exist, or equivalently, are false, the resolvent is false and the negated condition $\neg c$ is refuted (the goal).

However, to decide $\exists(b,d)\models\exists(a,c)$ is as hard as the original problem $\neg c\models$ true, as we can not dissolve nested subconditions. Therefore we need additional deduction rules that cope with this situation and create, manage and (hopefully) solve subproblems of the form $\exists(b,d)\models\exists(a,c)$. Formally, these deduction rules are defined as follows.

Definition 6 (deduction rules). *Let $c_1,\ldots,c_n,e$ be conditions. A* (deduction) rule *R has the form*

$$\frac{\begin{array}{c}c_1\\ \vdots\\ c_n\end{array}}{e}\quad \textit{if } \alpha$$

and is shortly denoted by $R=[c_1,\ldots,c_n/e]\alpha$. The conditions $c_1,\ldots,c_n$ are called premises*, e is the* resolvent *and α is an (informal)* side condition*. A rule may be applied to a condition c in CNF, if there is exists a non-negated subcondition c' in c such that $c'=\wedge_{j\in J}\,d_j$ is a conjunction of disjunctions $(d_j)_{j\in J}$ that contains all premises of R, i.e. for all $1\le k\le n$, there is a $j\in J$ with $c_k=d_j$, and the side condition α is satisfied. Application of R yields a new condition d that is derived from c by adding the resolvent e to the conjunction c'. We write $c\vdash_R d$ to denote such a derivation step, whereas we write $c\vdash_{\mathcal{K}} d$ to denote a derivation sequence $c\vdash_R\ldots\vdash_Q d$ with rules $R,\ldots,Q$ in $\mathcal{K}$.*

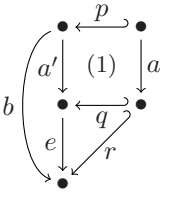
The deduction rules of our calculus contain variables for morphisms and conditions. Prior to a rule application, these variables must be matched in an unification process, as usual, to yield an applicable instance of the rule. For the formal definition of our calculus, we require the following theorem stating the possibility of combining two conditions $\exists p$ and c conjunctively:

Theorem 1 ([HP08, HP05]). *There is a transformation* A_2*, such that for every morphism p in $\mathcal{M}$ and every condition c over* $\mathrm{dom}(p)$ *the following holds: For all $p''\in\mathcal{M}$ with* $\mathrm{dom}(p'')=\mathrm{codom}(p)$*, $p''\models\mathrm{A}_2(p,c)\ \Leftrightarrow\ p''\circ p\models c$.*

The transformation A_2 is described in [HP08] as follows:

Construction 1. *For morphisms p in $\mathcal{M}$ and conditions over* $\mathrm{dom}(p)$*, let*

$$\begin{aligned}\mathrm{A}_2(p,\,\mathrm{true}) &= \mathrm{true}\\ \mathrm{A}_2(p,\,\exists(a,c)) &= \vee_{e\in\mathcal{E}}\,\exists(b,\,\mathrm{A}_2(r,c)).\end{aligned}$$



Construct the pushout (1) of p and a leading to morphisms a' and q. The disjunction $\vee_{e\in\mathcal{E}}$ ranges over all epimorphisms e with domain $\mathrm{dom}(a')$ *such that both $b=e\circ a'$ and $r=e\circ q$ are in $\mathcal{M}$.*

Furthermore, $\mathrm{A}_2(p,\,\neg c)=\neg\mathrm{A}_2(p,\,c)$ *and* $\mathrm{A}_2(p,\,\wedge_{j\in J}\,c_j)=\wedge_{j\in J}\,\mathrm{A}_2(p,\,c_j)$.

3.1 A Calculus for High-Level Conditions

In the following, we introduce a calculus $\mathcal{K}$ for high-level conditions representing the possible actions a theorem prover based on $\mathcal{K}$ may perform.

Definition 7 (calculus $\mathcal{K}$). *The calculus $\mathcal{K}$ for high-level conditions consists of the following six rules: (Descent), (Resolve), (Partial resolve), (Partial lift), (Lift) and (Supporting lift). Let a, b, m be morphisms and let c, d, c_1, c_2 be conditions.*

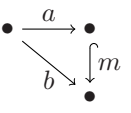
(Descent)

$$\frac{\exists(a,\ \mathrm{false} \wedge c) \vee c_1}{c_1}$$

(Resolve)

$$\frac{\neg\exists(a,\ \mathrm{true}) \vee c_1 \qquad \exists(b,\ d) \vee c_2}{c_1 \vee c_2}$$

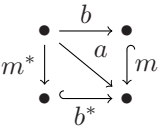
if $\exists m \in \mathcal{M}.\ m \circ a = b$ and $d \neq \mathrm{false}$



(Partial resolve)

$$\frac{\neg\exists(a,\ \mathrm{true}) \vee c_1 \qquad \exists(b,\ d) \vee c_2}{\neg\exists(m^*, \mathrm{true}) \vee c_1 \vee c_2}$$

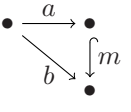
if $\exists m \in \mathcal{M}.\ m \circ b = a$ and $\langle m^*, b^* \rangle$ is the $\mathcal{M}$-pushout complement of $\langle b, m \rangle$ and $d \neq \mathrm{false}$



(Partial lift)

$$\frac{\neg\exists(a, c) \vee c_1 \qquad \exists(b, d) \vee c_2}{\exists(b,\ d \wedge \mathrm{A}_2(m, \neg c)) \vee c_1 \vee c_2}$$

if $c \neq \mathrm{true}$ and $\exists m \in \mathcal{M}.\ m \circ a = b$ and $d \neq \mathrm{false}$



(Lift)

$$\frac{\neg\exists(a, c) \vee c_1 \qquad \exists(b, d) \vee c_2}{\exists(b,\ d \wedge \mathrm{A}_2(b, \neg\exists(a, c))) \vee c_1 \vee c_2}$$

if $c \neq \mathrm{true}$ and $b \in \mathcal{M}$ and $d \neq \mathrm{false}$

(Supporting lift)

$$\frac{\exists(a, c) \vee c_1 \qquad \exists(b, d) \vee c_2}{\exists(b,\ d \wedge \mathrm{A}_2(b, \exists(a, c))) \vee c_1 \vee c_2}$$

if $b \in \mathcal{M}$ and $c \neq \mathrm{false}$ and $d \neq \mathrm{false}$

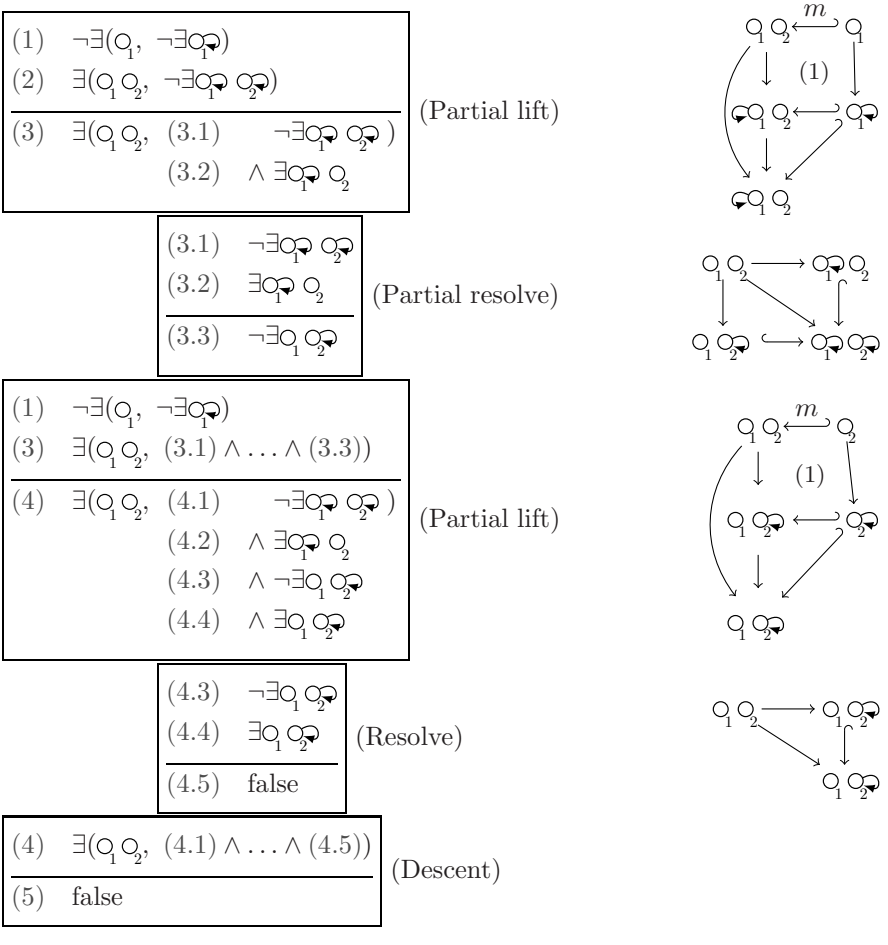
The rule (Descent) is used to carry over a successful nested refutation into an outer refutation. The rule (Resolve) is the core of our calculus and represents a straightforward case for which the problem $\exists(b, d) \models \exists(a, c)$ is decidable. The rules (Descent) and (Resolve) are the only ones that (may) reduce the number of elements in a disjunction. The rule (Partial resolve) is necessary for proving the validity of conditions outside the decidable $\forall$-free fragment of conditions. The rules (Partial lift), (Lift) and (Supporting lift) are similar in the sense that they create additional facts to find nested refutations by combining information. Outstandig is the rule (Partial lift) which moves the negation from a condition

towards the nested subcondition. Note that (Supporting lift) is the only rule for which repeated applications on its own resolvent may be necessary. For graphs, unbounded applications of (Supporting lift) are the only reason a theorem prover based on $\mathcal{K}$ does not terminate, assuming that the deduction of structural equivalent conditions is surpressed, as discussed in Section 4.

Example 5. Consider the following tautology $\forall(\bigcirc_1,\ \exists\bigcirc_1) \Rightarrow \forall(\bigcirc_1\ \bigcirc_2,\ \exists\bigcirc_1\ \bigcirc_2)$ expressing "Every node has a loop implies every two nodes each have a loop". A transformation into CNF yields

$$\begin{aligned} &(1) \quad \neg\exists(\bigcirc_1,\ \neg\exists\bigcirc_1) \\ &(2) \quad \wedge\ \exists(\bigcirc_1\ \bigcirc_2,\ \neg\exists\bigcirc_1\ \bigcirc_2) \end{aligned}$$

and a proof of the statement's validity is as follows:



Example 6. Consider the condition stated in Example 4. Given the rules of $\mathcal{K}$, our goal is to refute the disjunction (3) with the help of the facts (1) and (2). The rule (Resolve) can be applied with argument (1) to resolve (3.1)-(3.6), e.g.,

$$\frac{(1)\quad (3.1) \vee ((3.2) \vee \ldots \vee (3.11))}{(3.2) \vee \ldots \vee (3.11)}\ \text{(Resolve)}$$

Subconditions (3.7)-(3.11) are resolved by applying rule (Partial lift) with argument (2) and subsequent application of (Resolve) on the nested subconditions, and (Descent), e.g.,

$$\frac{(2)\quad (3.7) \vee ((3.8) \vee \ldots \vee (3.11))}{(3.7') \vee (3.8) \vee \ldots \vee (3.11)}\ \text{(Partial lift)}$$

with (3.7') 

Eventually, we yield an empty disjunction, or equivalently, false as an element of the outer conjunction, thus the input condition is refuted and the condition proved.

3.2 Soundness

In this section, we prove the soundness of the calculus $\mathcal{K}$. We show that every application of a rule R in $\mathcal{K}$ corresponds to a logical deduction.

Theorem 2 (soundness of $\mathcal{K}$). *The calculus $\mathcal{K}$ for high-level conditions is sound, i.e., for every conditions c, d over C in CNF the following holds:*

$$c \vdash_{\mathcal{K}} d \quad \textit{implies} \quad c \models d.$$

The proof is done in three steps: first, we establish that we can investigate the soundness of deduction rules independently of disjunctive context. In the following, let r, p_j, q_j be conditions for $1 \leq j \leq n$.

Fact 3. *For every rule $R = [(p_1 \vee q_1), \ldots, (p_n \vee q_n)/(r \vee q_1 \vee \ldots \vee q_n)]\alpha$ we have $(p_1 \wedge \ldots \wedge p_n) \models r$ implies $((p_1 \vee q_1) \wedge \ldots \wedge (p_n \vee q_n)) \models (r \vee q_1 \vee \ldots \vee q_n)$.*

Second, we can prove the soundness of each individual rule R in $\mathcal{K}$.

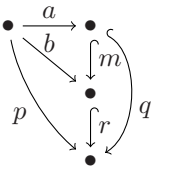
Lemma 1. *For every rule $R = [c_1, \ldots, c_n/d]\alpha$ in $\mathcal{K}$, if α holds then $(c_1 \wedge \ldots \wedge c_n) \models d$.*

Proof. For the rule (Descent), we have $\exists(a,\ \text{false} \wedge c) \equiv \exists(a,\ \text{false}) \equiv \text{false}$. For every rule of the form $R = [(p_1 \vee q_1), \ldots, (p_n \vee q_n)/(r \vee q_1 \vee \ldots \vee q_n)]\alpha$ with $c_j = (p_j \vee q_j)$ for $1 \leq j \leq n$, we first show $(p_1 \wedge \ldots \wedge p_n) \models r$:

(Resolve). First, we transform the proof obligation:

$$\begin{aligned}
&(\neg\exists(a, \text{true}) \wedge \exists(b, d)) \Rightarrow \text{false} \\
&\equiv \neg(\neg\exists(a, \text{true}) \wedge \exists(b, d)) \vee \text{false} && \text{(Def. } \Rightarrow\text{)} \\
&\equiv \exists(a, \text{true}) \vee \neg\exists(b, d) \vee \text{false} && \text{(De Morgan)} \\
&\equiv \exists(a, \text{true}) \vee \neg\exists(b, d) && ((c \vee \text{false}) \equiv c) \\
&\equiv \exists(a, \text{true}) \Leftarrow \exists(b, d) && \text{(Def. } \Rightarrow\text{)}
\end{aligned}$$

We show $\exists(b,\, d) \models \exists(a,\, \text{true})$:

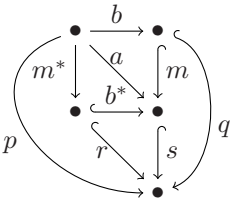


$$
\begin{aligned}
&\exists(b,\, d)\\
\models\ &\exists(b,\, \text{true}) && (d \models \text{true})\\
\models\ &\exists(a,\, \text{true}) && (\exists m \in \mathcal{M}.\ m \circ a = b, \text{ Def. 2})
\end{aligned}
$$

(Partial resolve). First, we transform the proof obligation:

$$
\begin{aligned}
&(\neg\exists(a,\, \text{true}) \wedge \exists(b,\, d)) \Rightarrow \neg\exists(m^*,\, \text{true})\\
\equiv\ &\neg(\neg\exists(a,\, \text{true}) \wedge \exists(b,\, d)) \vee \neg\exists(m^*,\, \text{true}) && (\text{Def. } \Rightarrow)\\
\equiv\ &\exists(a,\, \text{true}) \vee \neg\exists(b,\, d) \vee \neg\exists(m^*,\, \text{true}) && (\text{De Morgan})\\
\equiv\ &\exists(a,\, \text{true}) \Leftarrow \neg(\neg\exists(b,\, d) \vee \neg\exists(m^*,\, \text{true})) && (\text{Def. } \Rightarrow)\\
\equiv\ &\exists(a,\, \text{true}) \Leftarrow (\exists(b,\, d) \wedge \exists(m^*,\, \text{true})) && (\text{De Morgan})
\end{aligned}
$$

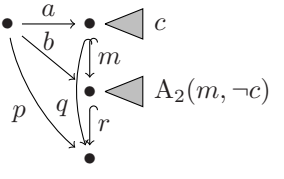
We show $(\exists(b,\, d) \wedge \exists(m^*,\, \text{true})) \models \exists(a,\, \text{true})$:



$$
\begin{aligned}
&p \models (\exists(b,\, d) \wedge \exists(m^*,\, \text{true}))\\
\Leftrightarrow\ &\exists q \in \mathcal{M}.\ q \circ b = p \text{ and } q \models d\\
&\text{and } \exists r \in \mathcal{M}.\ r \circ m^* = p \text{ and } r \models \text{true} && (\text{Def. 2})\\
\Rightarrow\ &\exists s \in \mathcal{M}.\ r \circ m^* = s \circ b^* \circ m^* = s \circ a = p\\
&\text{and } s \models \text{true} && (\mathcal{M}\text{-Pushout})
\end{aligned}
$$

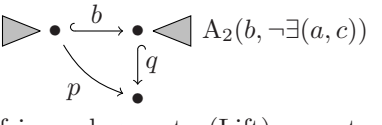
(Partial lift). We show $\exists(b,d) \wedge \neg\exists(a,c) \models \exists(b,\ d \wedge \mathrm{A}_2(m, \neg c))$:

$$
\begin{aligned}
&p \models \exists(b,d) \wedge \neg\exists(a,c)\\
\Leftrightarrow\ &p \models \exists(b,d) \text{ and } p \models \neg\exists(a,c) && (\text{Def. 2})\\
\Leftrightarrow\ &\exists r \in \mathcal{M}.\ r \circ b = p \text{ and } r \models d \text{ and } r \circ b \models \neg\exists(a,c) && (\text{Def. 2})\\
\Rightarrow\ &\exists r \in \mathcal{M}.\ r \circ b = p \text{ and } r \models d \text{ and}\\
&\exists r \circ m \in \mathcal{M}.\ r \circ m \circ a = p \text{ and } r \circ m \models \neg c && (m \circ a = b, \text{ Def. 2})\\
\Leftrightarrow\ &\exists r \in \mathcal{M}.\ r \circ b = p \text{ and } r \models d \text{ and}\\
&\exists r \circ m \in \mathcal{M}.\ r \circ m \circ a = p \text{ and } r \models \mathrm{A}_2(m, \neg c) && (\text{Thm. 1})\\
\Leftrightarrow\ &\exists r \in \mathcal{M}.\ r \circ b = p \text{ and } r \models d \text{ and } r \models \mathrm{A}_2(m, \neg c) && (m \circ a = b, \text{ Def. 2})\\
\Leftrightarrow\ &p \models \exists(b,\ d \wedge \mathrm{A}_2(m, \neg c)) && (\text{Def. 2})
\end{aligned}
$$



(Lift). We show $\exists(b,d) \wedge \neg\exists(a,c) \models \exists(b,\ d \wedge \mathrm{A}_2(b, \neg\exists(a,c)))$:

$$
\begin{aligned}
&p \models \exists(b,d) \wedge \neg\exists(a,c)\\
\Leftrightarrow\ &\exists q \in \mathcal{M}.\ q \circ b = p \text{ and } q \models d \text{ and } q \circ b \models \neg\exists(a,c) && (\text{Def. 2})\\
\Leftrightarrow\ &\exists q \in \mathcal{M}.\ q \circ b = p \text{ and } q \models d \text{ and } q \models \mathrm{A}_2(b, \neg\exists(a,c)) && (\text{Thm. 1})\\
\Leftrightarrow\ &p \models \exists(b,\ d \wedge \mathrm{A}_2(b, \neg\exists(a,c))) && (\text{Def. 2})
\end{aligned}
$$



(Supporting lift). The proof is analogous to (Lift) except $\neg\exists(a,c)$ is replaced with $\exists(a,c)$.

By Fact 3, we can lift any statement $(p_1 \wedge \ldots \wedge p_n \models r)$ for any disjunctive context $q_1, \ldots, q_n$ and yield $(p_1 \vee q_1) \wedge \ldots \wedge (p_n \vee q_n) \models (r \vee q_1 \vee \ldots \vee q_n)$. This concludes the soundness proof for the deduction rules in $\mathcal{K}$.

Third, we show that deductions concerning non-negated subconditions within a condition in CNF can be lifted to the whole condition.

Fact 4. *For any non-negated condition c' within a condition c over C in CNF, with d derived from c by replacing c' with d', we have $c' \models d'$ implies $c \models d$.*

Proof. By induction over the structure of conditions.
Basis: $c = c' \models d' = d$.
Step: We show, for all morphisms m in $\mathcal{M}$ with domain C:
Case $\exists(a, c)$: $m \models \exists(a, c)$ iff $(\exists q \in \mathcal{M}.\ m = q \circ a$ and $q \models c)$ implies $(\exists q \in \mathcal{M}.\ m = q \circ a$ and $q \models d)$ iff $m \models \exists(a, d)$.
Case $(c \wedge e)$: $m \models (c \wedge e)$ iff $(m \models c$ and $m \models e)$ implies $(m \models d$ and $m \models e)$ iff $m \models (d \wedge e)$.
Case $(c \vee e)$: analogous to $(c \wedge e)$.
The case $\neg c$ is excluded by the assumption that c' is a non-negated subcondition.

Finally, we can prove the soundness of $\mathcal{K}$.

Proof of Theorem 2. Let c, d be arbitrary conditions over C in CNF. A deduction $c \vdash_{\mathcal{K}} d$ is a sequence of deductions $c \vdash_R \ldots \vdash_Q d$ for rules $R, \ldots, Q$ in $\mathcal{K}$. Using induction over the length of the deduction, we can reduce the proof obligation to "$c \vdash_R d$ implies $c \models d$", where c, d are arbitrary conditions over C in CNF and $R = [c_1, \ldots, c_n/e]\alpha$ is an arbitrary deduction rule in $\mathcal{K}$. Assume, $c \vdash_R d$. By Definition 6, there is a non-negated subcondition c' which is a conjunction $(c_1 \wedge \ldots \wedge c_n \wedge q)$ and d is derived from c by adding e to the conjunction, i.e. $(c_1 \wedge \ldots \wedge c_n \wedge q) \vdash (e \wedge c_1 \wedge \ldots \wedge c_n \wedge q)$. By Lemma 1, we have $(c_1 \wedge \ldots \wedge c_n) \models e$. Consequently, $(c_1 \wedge \ldots \wedge c_n \wedge q) \models (e \wedge q)$. By Fact 4, we conclude $c \models d$.

3.3 Necessity

In the following, we investigate whether or not a rule is necessary, for every rule of the calculus $\mathcal{K}$. A rule R is necessary, if there exists a tautology which cannot be proven anymore if R is omitted. We show that (Resolve) could be omitted, if the artificial restriction $c \neq \text{true}$ is omitted from (Partial lift), and show that all other rules are necessary. However, our considerations do not exclude the existence of a smaller calculus with similar or different rules.

Fact 5. *For every deduction $c' \vdash_{(Resolve)} f'$, there is a sequence of deductions*

$$c' \vdash_{(Partial\ lift')} d' \vdash_{(\neg\text{true} \equiv \text{false})} e' \vdash_{(Descent)} f'$$

where (Partial lift')=$[\exists(b, d) \wedge \neg\exists(a, c)/\exists(b,\ d \wedge \mathrm{A}_2(m, \neg c))]\alpha$ and $\alpha = (\exists m \in \mathcal{M}.\ m \circ a = b$ and $d \neq \text{false})$.

Proof. Let $c' = (\neg\exists(a, \text{true}) \wedge \exists(b, d))$. Then $c' \vdash_{(\text{Partial lift'})} d' \vdash_{(\neg\text{true} \equiv \text{false})} e' \vdash_{(\text{Descent})} f'$ where $d' = \exists(b,\ d \wedge \mathrm{A}_2(m, \neg\text{true})) = \exists(b,\ d \wedge \neg\text{true})$ and

$e' = \exists(b,\ d \wedge \text{false})$ and $f' = \text{false}$. Using Fact 3 and Lemma 1, our considerations can be lifted to arbitrary c', f' with $c' \vdash_{(\text{Resolve})} f'$.

We propose to let shortcut (Resolve) remain in $\mathcal{K}$, as the side conditions of (Resolve) and (Partial lift) prevent both rules from being applicable simultaneously.

Fact 6. *For every rule $R \in \mathcal{K} \setminus \{$(Resolve)$\}$, there is a condition c such that "$\neg c \vdash_{\mathcal{K}}$ false" and not "$\neg c \vdash_{\mathcal{K} \setminus \{R\}}$ false".*

Proof. **(Descent)** Negation of the tautology $\neg\exists(\bigcirc,\ \text{false})$ with the meaning "No node satisfies false" yields $\exists(\bigcirc,\ \text{false})$. No other rule is applicable.

(Partial resolve) Negation of the tautology $\forall(\bigcirc_1\ \bigcirc_2,\ \exists\circlearrowright_1\ \circlearrowright_2 \vee \neg\exists\bigcirc_1\ \circlearrowright_2 \vee \neg\exists\circlearrowright_1\ \bigcirc_2)$ with the meaning "For every pair of nodes, either both have loops or the first node has no loop or the second node has no loop" yields $\exists(\bigcirc_1\ \bigcirc_2,\ \neg\exists\circlearrowright_1\ \circlearrowright_2 \wedge \exists\bigcirc_1\ \circlearrowright_2 \wedge \exists\circlearrowright_1\ \bigcirc_2)$. Only the rule (Partial resolve) can derive the intermediate fact $\exists\bigcirc_1\ \circlearrowright_2$, or alternatively $\exists\circlearrowright_1\ \bigcirc_2$, required for a refutation.

(Partial lift) Negation of the tautology $\exists(\bigcirc_1,\ \neg\exists\circlearrowright_1) \Rightarrow \neg\forall(\bigcirc_1,\ \exists\circlearrowright_1)$ with the meaning "There is a node without a loop implies not every node has a loop" yields $\exists(\bigcirc_1,\ \neg\exists\circlearrowright_1) \wedge \neg\exists(\bigcirc_1,\ \neg\exists\circlearrowright_1)$ and only the use of (Partial lift) leads to a successful refutation as it negates the subcondition $\neg\exists\circlearrowright_1$:

$$\frac{\begin{array}{l}\neg\exists(\bigcirc_1,\ \neg\exists\circlearrowright_1)\\ \exists(\bigcirc_1,\ \neg\exists\circlearrowright_1)\end{array}}{\exists(\bigcirc_1,\ \neg\exists\circlearrowright_1 \wedge \exists\circlearrowright_1)} \quad \text{(Partial lift)}$$

(Lift) Negation of the tautology $\neg\exists\circlearrowright \Rightarrow \forall(\bigcirc_1,\ \neg\exists\circlearrowright_1 \vee \exists\bigcirc_1\ \bigcirc_2)$ with the meaning "There is no node with a loop implies for all nodes, there is no loop or there is a second node" yields $\neg\exists\circlearrowright \wedge \exists(\bigcirc_1,\ \exists\circlearrowright_1 \wedge \neg\exists\bigcirc_1\ \bigcirc_2)$ and no rule other than (Lift) is applicable. Note, partial resolve is not applicable because the ($\mathcal{M}$-)pushout complement is non-existent.

$$\begin{array}{ccc}\emptyset & \xrightarrow{b} & \bigcirc \\ \downarrow & \searrow^{a} & \downarrow m \\ \circlearrowright & \hookrightarrow & \circlearrowright\end{array}$$

(Supporting lift) Negation of the tautology $\exists\circlearrowright_1 \Rightarrow \forall(\bigcirc_1,\ \exists\circlearrowright_1 \vee \exists\ \bigcirc_1\ \circlearrowright_2)$ with the meaning "There is a node with a loop implies for all nodes, the node has a loop or there is a second node with a loop" yields $\exists\circlearrowright_1 \wedge \exists(\bigcirc_1,\ \neg\exists\circlearrowright_1 \wedge \neg\exists\bigcirc_1\ \circlearrowright_2)$ and no rule other than (Supporting lift) is applicable.

4 Implementation

In this section, we discuss practical aspects of a theorem prover based on $\mathcal{K}$. The deduction rules represent the main computation steps a theorem prover based on $\mathcal{K}$ will perform. Besides an implementation of those deduction rules, one requires a method that transforms any condition into conjunctive $\mathcal{M}$-normal form.

The equivalences depicted in Figure 2, strictly read from left to right, can be applied as long as possible to transform any condition into an (optimized) condition in conjunctive $\mathcal{M}$-normal form. In [Pen04], the equivalences are proven and it is shown that an as long as possible application yields the desired normal form. Another implementational aspect is the prevention of redundancy of rule applications with the intent to contain non-termination as far as possible. For example, any of the rules (Partial lift), (Lift) or (Supporting lift) may add subconditions to a conjunction that are already present anyway. Repeated application of such a rule on its own resolvent would lead into an infinite redundant branch of the search space. In theses cases, a notion of structural equivalence can help to filter out double subconditions and to prevent unnecessary deductions: Two conditions c, d are said to be *structurally equivalent*, denoted by $c \mathrel{\widehat{=}} d$, if $c = \text{true} = d$, or if $c = \neg c'$, $d = \neg d'$ and c', d' are structurally equivalent, or if $c = (c_1 \wedge c_2)$, $d = (d_1 \wedge d_2)$ and at least (c_1, d_1 and c_2, d_2) or (c_1, d_2 and c_2, d_1) are structurally equivalent (case $\vee$ analogous), or if $c = \exists(a, c')$, $d = \exists(a, d')$ and c', d' are structurally equivalent. The applicability of deduction rules may then be restricted to those cases for which the resolvent is not structurally equivalent to already existing conditions. Except for the rule (Supporting lift), this effectively prevents recursive application of rules to derived conditions.

$$
\begin{aligned}
\exists a &\equiv \exists(a, \text{true}) & \qquad \exists(\text{id},\ c) &\equiv c \\
\forall(a, c) &\equiv \neg\exists(a, \neg c) & \exists(a,\ \text{false}) &\equiv \text{false} \\
\exists(a, c) &\equiv \text{false} \ \text{ if } a \notin \mathcal{M} & \exists(a,\ \exists(b, c)) &\equiv \exists(b \circ a,\ c) \\
\neg\neg c &\equiv c & \exists(a,\ \vee_{j\in J}\ c_j) &\equiv \vee_{j\in J}\ \exists(a,\ c_j) \\
\neg\text{true} &\equiv \text{false} & \vee_{j\in J}\ c_j &\equiv \text{true} \ \text{ if } \exists k \in J.\ c_k = \text{true} \\
\neg\text{false} &\equiv \text{true} & \vee_{j\in J}\ c_j &\equiv \vee_{j\in J\setminus\{k\}}\ c_j \ \text{ if } \exists k \in J.\ c_k = \text{false} \\
\neg(\vee_{j\in J}\ c_j) &\equiv (\wedge_{j\in J}\ \neg c_j) & \wedge_{j\in J}\ c_j &\equiv \wedge_{j\in J\setminus\{k\}}\ c_j \ \text{ if } \exists k \in J.\ c_k = \text{true} \\
\neg(\wedge_{j\in J}\ c_j) &\equiv (\vee_{j\in J}\ \neg c_j) & \wedge_{j\in J}\ c_j &\equiv \text{false} \ \text{ if } \exists k \in J.\ c_k = \text{false} \\
((\wedge_{j\in J}\ c_j) \vee c) &\equiv (\wedge_{j\in J}\ (c_j \vee c)) & \vee_{j\in\emptyset}\ c_j &\equiv \text{false} \\
& & \wedge_{j\in\emptyset}\ c_j &\equiv \text{true}
\end{aligned}
$$

Fig. 2. Equivalences for conjunctive $\mathcal{M}$-normal form

5 Related Work

In this section, we briefly relate our results to other work. Earliest attempts to find deduction rules for graphical conditions are made by Koch et. al. [KMP05], but remain incomplete. They investigate the notion of conflicting conditions of the form $\forall(I \to X,\ \exists(X \to C))$ and state prerequisites under which a conjunction of two graph conditions of this form is unsatisfiable.

Independently to our work, Orejas et. al. [OEP08] investigate sound and complete calculi for three fragments of graph conditions: the fragment of Boolean conditions over basic existential conditions $\exists(I \to C)$, the fragment of Boolean conditions over basic existential conditions $\exists(I \to C)$ and non-negated "atomic" conditions of the form $\forall(I \to X,\ \exists(X \to C))$, and the fragment of Boolean Conditions over "atomic" conditions of the form $\forall(I \to X,\ \exists(X \to C))$. Their deduction

rules relate to our own as follows: (R1) is a special case of (Resolve), (R2) is comparable to the rule (Supporting Lift), and (R3) is comparable to the rule (Partial lift), although (R2), (R3) do not lift (parts of) the resolvent (as this is neither necessary nor possible for the considered fragments of conditions). The operator $\oplus$ is a special instance of A_2 restricted to basic existential conditions. In [Ore08], a sound and complete calculus for the fragment of "basic" and "positive atomic" attributed graph constraints is presented. Attributed graph constraints are conditions over attributed graphs combined with a formula expressing conditions on the attributes such as "$(x > y)$".

An alternative approach to apply theorem proving to graph transformation is a translation into logical formulas [Cou90]. Following this idea, Strecker [Str08] models graph transformation in the proof assistant Isabelle. His approach supports the manual verification of formulas of "a fragment of first-order logic enriched by transitive closure".

The relation to our previous work is as follows: In [Pen08], a correct and complete satisfiability algorithm named `SeekSat` is described. For the fragment of Boolean conditions over basic existential conditions $\exists a$, `SeekSat` is shown to terminate, thus is able to decide. While `SeekSat` covers contradictions with finite counterexamples and tautologies in the decidable fragment of conditions, the presented calculus $\mathcal{K}$ is intended to cover all tautologies with finite proofs.

6 Conclusion

In this paper, we presented a calculus for conditions over adhesive high-level replacement categories. We took resolution [Rob65] as an ideal and postulated six deduction rules able to refute conditions in conjunctive normal form. We proved that every rule application corresponds to a logical deduction, and investigated whether or not omission of any rule leads to an incomplete calculus. We discussed practical aspects concerning an implementation such as filtering out structural equivalent conditions, and briefly compared our results with related work. An implementation of $\mathcal{K}$ is currently under development. Future topics include

- a proof of the completeness of the calculus,
- a systematic evaluation of the implementation. Currently, all 74 example tautologies generated from correct program specifications of the "access control" example in [HPR06] can be proved in average 9.1 seconds (median 0.1s) (Intel T5600, 1.83GHz).

Acknowledgment. Many thanks to the referees for thoroughly reviewing the paper and suggesting several improvements.

References

[BFT06] Baumgartner, P., Fuchs, A., Tinelli, C.: Implementing the model evolution calculus. Int. Journal on Artificial Intelligence Tools 15(1), 21–52 (2006)

[Cou90] Courcelle, B.: Graph rewriting: An algebraic and logical approach. In: Handbook of Theoretical Computer Science, vol. B, pp. 193–242. Elsevier, Amsterdam (1990)

[DS89] Dijkstra, E.W., Scholten, C.S.: Predicate Calculus and Program Semantics. Springer, Heidelberg (1989)

[EEHP06] Ehrig, H., Ehrig, K., Habel, A., Pennemann, K.-H.: Theory of constraints and application conditions: From graphs to high-level structures. Fundamenta Informaticae 74, 135–166 (2006)

[EEPT06] Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs of Theoretical Computer Science. Springer, Berlin (2006)

[HP01] Habel, A., Plump, D.: Computational completeness of programming languages based on graph transformation. In: Honsell, F., Miculan, M. (eds.) FOSSACS 2001. LNCS, vol. 2030, pp. 230–245. Springer, Heidelberg (2001)

[HP05] Habel, A., Pennemann, K.-H.: Nested constraints and application conditions for high-level structures. In: Kreowski, H.-J., Montanari, U., Orejas, F., Rozenberg, G., Taentzer, G. (eds.) Formal Methods in Software and Systems Modeling. LNCS, vol. 3393, pp. 293–308. Springer, Heidelberg (2005)

[HP08] Habel, A., Pennemann, K.-H.: Correctness of high-level transformation systems relative to nested conditions. In: MSCS 2008 (Accepted for publication, 2008)

[HPR06] Habel, A., Pennemann, K.-H., Rensink, A.: Weakest preconditions for high-level programs. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 445–460. Springer, Heidelberg (2006)

[HW95] Heckel, R., Wagner, A.: Ensuring consistency of conditional graph grammars. In: SEGRAGRA 1995. ENTCS, vol. 2, pp. 95–104 (1995)

[KMP05] Koch, M., Mancini, L.V., Parisi-Presicce, F.: Graph-based specification of access control policies. JCSS 71, 1–33 (2005)

[McC08] McCune, W.: Homepage of Prover9 (2008), `http://www.prover9.org/`

[OEP08] Orejas, F., Ehrig, H., Prange, U.: A logic of graph constraints. In: Fiadeiro, J.L., Inverardi, P. (eds.) FASE 2008. LNCS, vol. 4961, pp. 179–199. Springer, Heidelberg (2008)

[Ore08] Orejas, F.: Attributed graph constraints. In: Ehrig, H., et al. (eds.) Proc. ICGT 2008. LNCS, vol. 5214, Springer, Heidelberg (2008)

[Pen04] Pennemann, K.-H.: Generalized constraints and application conditions for graph transformation systems. Master's thesis, Department of Computing Science, University of Oldenburg, Oldenburg (2004)

[Pen08] Pennemann, K.-H.: An algorithm for approximating the satisfiability problem of high-level conditions. In: Proc. GT-VC 2007. ENTCS, vol. 213, pp. 75–94. Elsevier, Amsterdam (2008)

[Ren04] Rensink, A.: Representing first-order logic by graphs. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 319–335. Springer, Heidelberg (2004)

[Rob65] Robinson, J.A.: A machine-oriented logic based on the resolution principle. Journal of the ACM 12, 23–41 (1965)

[RV02] Riazanov, A., Voronkov, A.: The design and implementation of VAMPIRE. AI Communications 15(2-3), 91–110 (2002)

[Str08] Strecker, M.: Modeling and verifying graph transformations in proof assistants. In: Proc. Termgraph 2007. ENTCS, vol. 203, pp. 135–148. Elsevier, Amsterdam (2008)