Using Closed Sets of Rules for the Entailment of Literals

Martin Caminada

Department of Information and Computing Sciences, Utrecht University

Technical Report UU-CS-2006-027

www.cs.uu.nl

ISSN: 0924-3275

Using Closed Sets of Rules for the Entailment of Literals

Martin W.A. Caminada

June 13, 2006

Abstract

Entailment that is based on the application of simple production rules, of the form $\mathtt{c} \leftarrow \mathtt{a}_1, \ldots, \mathtt{a}_\mathtt{n}$ ($n \geq 0$) is weaker than the propositional entailment that would be yielded by translating these rules into material implications. In this paper, we show that rule-based entailment coincides with the propositional entailment of literals, when the set of rules is closed under transposition, transitivity and antecedent cleanup.

1 Introduction

In the early days of logic programming, rules were often seen as some kind of computationally friendly material implications. It is a well-known fact that, without weak and strong negation, the set of atoms entailed by a set of rules coincides with the set of atoms entailed by the associated set of material implications.

Definition 1. *We say that $\mathcal{W}$ is an* implication based *set of formulas iff each formula in $\mathcal{W}$ is either a literal or a material implication of the form $\mathtt{c} \subset \mathtt{a}_1 \wedge \ldots \wedge \mathtt{a}_\mathtt{n}$ ($n \geq 1$) where $\mathtt{c}, \mathtt{a}_1, \ldots, \mathtt{a}_\mathtt{n}$ are literals. If $\mathcal{W}$ is implication based, then we define $\mathtt{Imps2Rules}(\mathcal{W})$ as $\{\mathtt{c} \leftarrow \mathtt{a}_1, \ldots, \mathtt{a}_\mathtt{n} \mid \mathtt{c} \subset \mathtt{a}_1 \wedge \ldots \wedge \mathtt{a}_\mathtt{n} \in \mathcal{W}\} \cup \{\mathtt{c} \leftarrow \mid \mathtt{c} \in \mathcal{W}\}$.*

Definition 2. *We say that $\mathcal{W}$ is a* negation-free *implication based set of formulas iff each formula in $\mathcal{W}$ is either an atomic proposition or a material implication of the form $\mathtt{c} \subset \mathtt{a}_1 \wedge \ldots \wedge \mathtt{a}_\mathtt{n}$ where $\mathtt{c}, \mathtt{a}_1, \ldots, \mathtt{a}_\mathtt{n}$ are atomic propositions.*

If r is a rule of the form $\mathtt{c} \leftarrow \mathtt{a}_1, \ldots, \mathtt{a}_\mathtt{n}$ ($n \geq 0$) then we define $head(r)$ as $\mathtt{c}$ and $body(r)$ as $\{\mathtt{a}_1, \ldots, \mathtt{a}_\mathtt{n}\}$. If S is a set of rules, we write $Cl(S)$ as the smallest set of literals that satisfies $\forall r \in S : (body(r) \subseteq Cl(S) \supset head(r) \in Cl(S)$.

Proposition 1. *Let $\mathcal{W}$ be a negation-free implication based set of formulas. Let $\mathtt{b}$ be an atomic proposition. It holds that $\mathcal{W} \vDash \mathtt{b}$ iff $\mathtt{b} \in Cl(\mathtt{Imps2Rules}(\mathcal{W}))$.*

When strong negation is added, this one-to-one correspondence no longer holds. The kind of entailment as specified by a logic program with strong negation differs greatly from the classical propositional entailment that would be yielded when the rules were interpreted as material implications.

Example 1 (transposition needed). *Let $\mathcal{W} = \{\mathtt{a};\ \neg\mathtt{a} \subset \neg\mathtt{b}\}$. Here $\mathcal{W} \vDash \mathtt{b}$ but $\mathtt{b} \notin Cl(\mathtt{Imps2Rules}(\mathcal{W}))$.*

Example 2 (antecedent cleaning on transitivity needed). *Let $\mathcal{W} = \{\mathtt{b} \subset \mathtt{a};\ \neg\mathtt{a} \subset \mathtt{b}\}$. Here $\mathcal{W} \vDash \neg\mathtt{a}$ but $\neg\mathtt{a} \notin Cl(\mathtt{Imps2Rules}(\mathcal{W}))$.*

Example 3 (transposition on transitivity needed). *Let* $\mathcal{W} = \{\mathtt{c} \subset \mathtt{a} \wedge \mathtt{b};\ \mathtt{d} \subset \mathtt{c} \wedge \mathtt{a};\ \mathtt{b};\ \neg\mathtt{d}\}$. *Here,* $\mathcal{W} \models \neg\mathtt{a}$ *but* $\neg\mathtt{a} \notin Cl(\mathtt{Imps2Rules}(\mathcal{W}))$.

As rules with strong negation cannot be regarded as simply modelling the propositional entailment of the associated material implications, some alternative view is needed. One possibility would be to regard the rules as domain dependent derivation rules. Thus, a rule is no longer seen as something at the object level (like a material implication) but as a meta-level principle of entailment (like for instance modus ponens). With every rule corresponding to a domain dependent derivation rule, a logic program in fact boils down to a particular, domain dependent, form of logical entailment.

In the rest of this report, we specify three closure operators on a set of rules that collectively restore the one to one correspondence between rule based derivation and the propositional entailment of literals.

2 Rule Based Derivation as Propositional Entailment

Definition 3 (transposition, transitivity, antecendent cleaning).
Let s_1 *and* s_2 *be rules. We say that* s_2 *is a* transpositive *version of* s_1 *iff:*

$s_1 = \mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}$ *and*

$s_2 = \neg\mathtt{a_i} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_{i-1}}, \neg\mathtt{c}, \mathtt{a_{i+1}}, \ldots, \mathtt{a_n}$ *for some* $1 \leq i \leq n$.

Let s_1, s_2 *and* s_3 *be strict rules. We say that* s_3 *is a* transitive *version of* s_1 *and* s_2 *iff:*

$s_1 = \mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}$,

$s_2 = \mathtt{a_i} \leftarrow \mathtt{b_1}, \ldots, \mathtt{b_m}$ *for some* $1 \leq i \leq n$, *and*

$s_3 = \mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_{i-1}}, \mathtt{b_1}, \ldots, \mathtt{b_m}, \mathtt{a_{i+1}}, \ldots, \mathtt{a_n}$.

Let s_1 *and* s_2 *be strict rules. We say that* s_2 *is an* antecedent cleaned *version of* s_1 *iff:*

$s_1 = \neg\mathtt{a_i} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_i}, \ldots, \mathtt{a_n}$ *and*

$s_2 = \neg\mathtt{a_i} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_{i-1}}, \mathtt{a_{i+1}}, \ldots, \mathtt{a_n}$.

The intuition behind transposition can be illustrated by translating a rule $\mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_n}$ to a material implication $\mathtt{c} \subset \mathtt{a_1} \wedge \cdots \wedge \mathtt{a_n}$. This implication is rewritten as a disjunction $\mathtt{c} \vee \neg(\mathtt{a_1} \wedge \ldots \wedge \mathtt{a_n})$, which in its turn can be written as a disjunction $\mathtt{c} \vee \neg\mathtt{a_1} \vee \cdots \vee \neg\mathtt{a_n}$. In this disjunction, different disjuncts can be put in front. Putting for instance $\mathtt{a_i}$ in front yields $\neg\mathtt{a_i} \vee \neg\mathtt{a_1} \vee \cdots \vee \neg\mathtt{a_{i-1}} \vee \mathtt{c} \vee \neg\mathtt{a_{i+1}} \vee \cdots \vee \neg\mathtt{a_n}$, which is again equivalent to $\neg\mathtt{a_i} \vee \neg(\mathtt{a_1} \wedge \ldots \wedge \mathtt{a_{i-1}} \wedge \neg\mathtt{c} \wedge \mathtt{a_{i+1}} \wedge \ldots \wedge \mathtt{a_n})$, and $\neg\mathtt{a_i} \subset \mathtt{a_1} \wedge \ldots \wedge \mathtt{a_{i-1}} \wedge \neg\mathtt{c} \wedge \mathtt{a_{i+1}} \wedge \ldots \wedge \mathtt{a_n}$. This can then be translated to the rule $\neg\mathtt{a_i} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_{i-1}}, \neg\mathtt{c}, \mathtt{a_{i+1}}, \ldots, \mathtt{a_n}$. Notice that, when $n = 1$, transposition coincides with classical contraposition. Transitivity, as used in Definition 3, basically boils down to the substitution of a literal in the body of a rule with the body of another rule that has this literal as its head. The meaning of antecedent cleaning is also straightforward. Translate a rule $\neg\mathtt{a_i} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_i} \ldots, \mathtt{a_n}$to a material implication $\neg\mathtt{a_i} \subset \mathtt{a_1} \wedge \cdots \wedge \mathtt{a_i} \wedge \cdots \wedge \mathtt{a_n}$, which is then equivalent to the disjunction $\neg\mathtt{a_i} \vee \neg\mathtt{a_1} \vee \cdots \vee \neg\mathtt{a_i} \vee \cdots \vee \neg\mathtt{a_n}$. In this formula, the double occurrence of $\neg\mathtt{a_i}$ can be eliminated, yielding $\neg\mathtt{a_i} \vee \neg\mathtt{a_1} \vee \cdots \vee \neg\mathtt{a_{i-1}} \vee \neg\mathtt{a_{i+1}} \vee \cdots \vee \mathtt{a_n}$, which is equivalent to $\neg\mathtt{a_i} \subset \mathtt{a_1} \wedge \cdots \wedge \mathtt{a_{i-1}} \wedge \mathtt{a_{i+1}} \wedge \cdots \wedge \mathtt{a_n}$. This is then translated to the rule $\neg\mathtt{a_1} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_{i-1}}, \mathtt{a_{i+1}}, \ldots, \mathtt{a_n}$.

Definition 4 (closed). *Let* S *be a set of rules. Then,* *(i) S is* closed under transposition *iff for each rule s_1 in S, a rule s_2 is in S if s_2 is a transpositive version of s_1.*

(ii) S is closed under transitivity *iff for each rule s_1 and s_2 in S, a rule s_3 is in S if r_3 is a transitive version of s_1 and s_2.*

(iii) S is closed under antecedent cleaning *iff for each rule s_1 in S, a rule s_2 is in S if s_2 is an antecedent cleaned version of s_1.*

Definition 5 ($\mathcal{P}_{\mathcal{W}}$). *Let $\mathcal{W}$ be an implication-based set of rules. We define $\mathcal{P}_{\mathcal{W}}$ as the smallest set that includes* `Imps2Rules`$(\mathcal{W})$ *and is closed under transposition, transitivity and antecedent cleaning.*

In the rest of this section it will be proved that a literal follows from $\mathcal{W}$ iff it is in $Cl(\mathcal{P}_{\mathcal{W}})$. As the proof of our main theorem is based on resolution theory, we first state a few preliminaries. Recall that any arbitrary set of propositional formulas can be converted to disjunctive normal form (notation: $DNF(\phi)$), which in its turn can be represented as a set of clauses.

Definition 6.

- *A* clause *is a set of literals. The empty clause is denoted as $\Box$.*
- *Let C_1 and C_2 be clauses, such that for some literal* $\mathtt{l}$*:* $\mathtt{l} \in C_1$ *and* $\neg\mathtt{l} \in C_2$*. Then* $(C_1 - \{\mathtt{l}\}) \cup (C_2 - \{\neg\mathtt{l}\})$ *is called the* resolvent *of C_1 and C_2 on* $\mathtt{l}$*. The fact that C_3 is a resolvent of C_1 and C_2 on l is denoted as $C_1, C_2 \rightsquigarrow_{\mathtt{l}} C_3$.*
- *A* resolution-tree *RT from a set of clauses $\{C_1, \ldots, C_n\}$ to a clause C is a binary tree of clauses such that:*
  1. *the root of RT is C*
  2. *each leaf of RT is a clause from $\{C_1, \ldots, C_n\}$*
  3. *each non-leaf node is a resolvent of its children.*

In the following theorem and beyond, we write `Lits2Clauses`(L) as an abbreviation for $\{\{\mathtt{l}\} \mid \mathtt{l} \in L\}$. We also use $\neg L$ as an abbreviation for $\{\neg\mathtt{l} \mid \mathtt{l} \in L\}$.

Theorem 1 ([3]). *Let $\{C_1, \ldots, C_n\}$ be a set of clauses and ϕ be a formula. It holds that $C_1, \ldots, C_n \vDash \phi$ iff there exists a resolution-tree from $\{C_1, \ldots, C_n\} \cup DNF(\neg\phi)$ to $\Box$.*

Lemma 1. *Let $\{C_1, \ldots, C_n\}$ be a set of clauses and let L be a set of literals. Let RT be a resolution-tree from $\{C_1, \ldots, C_n\} \cup$*`Lits2Clauses`$(L)$ *to $\Box$. There also exists a resolution-tree RT' from $\{C_1, \ldots, C_n\} \cup$*`Lits2Clauses`$(L)$ *to $\Box$ in which for every resolution-step of C' and C'' on* $\mathtt{l} \in L$ *it holds that either* $C' = \mathtt{l}$ *or* $C'' = \mathtt{l}$*.*

Sketch of Proof. The idea is that by substituting one of the inputs of a resolution-step by the literal that is actually used by this resolution step, one obtains a resolvent which is a subset of the original resolvent (this means that we can prune a part of the remaining resolution-tree but still get the empty clause as root). The idea is to keep doing this until one obtains a resolution-tree that satisfies the lemma. $\Box$

An example of the application of Lemma 1 would be the following. Let $C = \{\{\texttt{a}\}, \{\neg\texttt{c}\}, \{\neg\texttt{a}, \texttt{c}\}\}$ and $L = \{\texttt{c}\}$. There exists a resolution-tree RT in which $\{\neg\texttt{a}, \texttt{c}\}$ is resolved with $\{\neg\texttt{c}\}$ to $\{\neg\texttt{a}\}$ and $\{\neg\texttt{a}\}$ is resolved with $\{\texttt{a}\}$ to $\Box$. Lemma 1 then tells that there also exists a resolution-tree RT' in which for every resolution step on $\texttt{c}$ it holds that $C' = \texttt{c}$ or $C'' = \texttt{c}$. In this case RT' simply resolves $\{\neg\texttt{c}\}$ and $\{\texttt{c}\}$ to $\Box$.

Theorem 2. *Let $\{C_1, \ldots, C_n\}$ be a consistent set of clauses and L be a minimal and consistent set of literals such that $\{C_1, \ldots, C_n\} \cup \texttt{Lits2Clauses}(L) \models \bot$. There exists a resolution-tree from $\{C_1, \ldots, C_n\}$ to $\neg L$.*

Proof. The fact that $\{C_1, \ldots, C_n\} \cup \texttt{Lits2Clauses}(L) \models \bot$ means that there exists a resolution-tree RT from $\{C_1, \ldots, C_n\} \cup \texttt{Lits2Clauses}(L)$ to $\Box$. Then, according to Lemma 1, there exists a resolution-tree RT' from $\{C_1, \ldots, C_n\} \cup \texttt{Lits2Clauses}(L)$ to $\Box$ in which every resolution-step on $\texttt{l} \in L$ involves at least one element of L. The fact that L is a *minimal* set such that $\{C_1, \ldots, C_n\} \cup \texttt{Lits2Clauses}(L) \models \bot$ means that each $\texttt{l}_\texttt{i} \in L$ must occur as a leaf in RT'.

Now, for an arbitrary $\texttt{l} \in L$ do the following. Convert the resolution-tree RT' to a resolution-tree RT'' by cutting out all occurrences of resolution on $\texttt{l}$. In figure 1, we it is shown how this is done in the case where resolution on $\texttt{l}$ is the last step as well as in the case where it is not the last step.

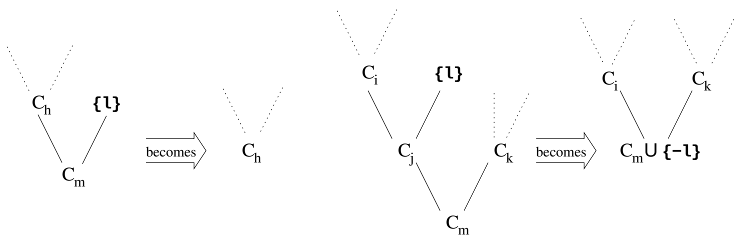

Figure 1: Removing L from a resolution tree.

Furthermore, $\neg\texttt{l}$ is added to every clause on the path from C_m to the root of the resolution-tree. The idea is that one keeps carrying out this procedure until all resolution-steps on $\texttt{l}$ are dealt with. The resulting resolution-tree RT'' goes from $\{C_1, \ldots, C_n\} \cup \texttt{Lits2Clauses}(L - \{\texttt{l}\})$ to $\{\texttt{l}\}$. Repeat this procedure for *every* $\texttt{l} \in L$. The final result will be a resolution tree RT''' from $\{C_1, \ldots, C_n\}$ to $\neg L$. $\Box$

We now introduce two operators to typecast a rule or set of rules into a clause or set of clauses.

Definition 7. *Let r be a rule of the form $\texttt{c} \leftarrow \texttt{a}_1, \ldots, \texttt{a}_\texttt{n}$. We define $\texttt{Rule2Clause}(r)$ as $\{\texttt{c}, \neg\texttt{a}_1, \ldots, \neg\texttt{a}_\texttt{n}\}$. Let S be a set of rules. We define $\texttt{Rules2Clauses}(S)$ as $\{\texttt{Rule2Clause}(r) \mid r \in S\}$.*

Theorem 3. *Let RT be a resolution-tree from $C_1, \ldots, C_n$ to C_0 (with $C_0 \neq \Box$) and let S be a set of rules, closed under transposition, transitivity and antecedent cleaning with $\{C_1, \ldots, C_n\} \subseteq \texttt{Rules2Clauses}(S)$. Then, for every clause C in RT there exists a rule $r \in S$ such that $\texttt{Rule2Clause}(r) = C$.*

Proof. We prove this by induction on the depth of subtree RT' of RT.

basis Let the depth of RT' be one. Then the only clause in RT' is an element of $\{C_1, \ldots, C_n\}$. As $\{C_1, \ldots, C_n\} \subseteq \texttt{Rules2Clauses}(S)$, it follows that for every clause C in RT there exists a rule $r \in S$ such that $\texttt{Rule2Clause}(r) = C$.

step Suppose for every RT' that is a subtree of RT with a depth of at most n, it holds that every clause C in RT', there exists a rule $r \in S$ such that $\texttt{Rule2Clause}(r) = C$. We will now prove that also for every resolution tree RT'' with a depth of $n + 1$, it holds that for every clause C in RT'', there exists a rule $r \in S$ such that $\texttt{Rule2Clause}(r) = C$. Let C be a clause in RT''. If C is not the root of RT'' then we can immediately apply the induction hypothesis, and have that there indeed exists a rule $r \in S$ such that $\texttt{Rule2Clause}(r) = C$. In the remainder of this proof, we will treat the case that C is the root of RT''. As RT'' has a depth of at least 2, C is the resolvent of two other clauses (children), say C_1 and C_2, which are themselves the roots of resolution trees RT_1 and RT_2. Now, the trees RT_1 and RT_2 each have a depth of at most n so the induction hypothesis tells us that there exists a rule $r_1 \in S$ such that $\texttt{Rule2Clause}(r_1) = C_1$ and there exists a rule $r_2 \in S$ such that $\texttt{Rule2Clause}(r_2) = C_2$. As S is closed under antecedent-cleaning, there also exist two rules r_1' and r_2' of which the negation of their head heads is not in their respective bodies (they are antecedent cleaned). Let us assume that C_1 and C_2 are resolved to C by resolution on some literal $\texttt{q}$. Then, the fact that S is closed under transposition means that S contains two rules r_1'' and r_2'' of the form $\texttt{t}_1 \leftarrow \texttt{t}_2, \ldots, \texttt{t}_n, \texttt{q}$ and $\texttt{q} \leftarrow \texttt{s}_1, \ldots, \texttt{s}_m$. As S is closed under transitivity, this also means that S contains a rule $\texttt{t}_1 \leftarrow \texttt{t}_2, \ldots, \texttt{t}_n, \texttt{s}_1, \ldots, \texttt{s}_m$. It is this rule ($r'''$) for which $\texttt{Rule2Clause}(r''') = C$.

□

Theorem 4. *Let S be a set of rules that is closed under transposition, transitivity and antecedent-cleanup, and let L be a consistent set of literals such that* $\texttt{Rules2Clauses}(S) \cup \texttt{Lits2Clauses}(L) \not\models \bot$. *Let* $\texttt{p}$ *be a literal. It holds that* $\texttt{Rules2Clauses}(S) \cup \texttt{Lits2Clauses}(L) \models \texttt{p}$ *iff S contains a rule* $\texttt{p} \leftarrow \texttt{l}_1, \ldots, \texttt{l}_n$ *with* $\texttt{l}_1, \ldots, \texttt{l}_n \in L$.

Proof.
"$\Longrightarrow$": Suppose $\texttt{Rules2Clauses}(S) \cup \texttt{Lits2Clauses}(L) \models \texttt{p}$. Then, it also holds that $\texttt{Rules2Clauses}(S) \cup \texttt{Lits2Clauses}(L) \cup \{\{\neg\texttt{p}\}\} \models \bot$. Let $L' = \{\texttt{l}_1, \ldots, \texttt{l}_n\}$ be a minimal subset of L such that $\texttt{Rules2Clauses}(S) \cup \texttt{Lits2Clauses}(L') \cup \{\{\neg\texttt{p}\}\} \models \bot$. Then, $L' \cup \{\neg p\}$ is also a minimal set of literals such that $\texttt{Rules2Clauses}(S) \cup \texttt{Lits2Clauses}(L' \cup \{\neg\texttt{p}\}) \models \bot$ (this is because $\texttt{Rules2Clauses}(S) \cup \texttt{Lits2Clauses}(L) \not\models \bot$, so $\neg\texttt{p}$ is actually needed to entail $\bot$). Then, according to theorem 2, there exists a resolution-tree RT from $\texttt{Rules2Clauses}(S)$ to $\neg(L' \cup \{\neg\texttt{p}\}) = \{\neg\texttt{l}_1, \ldots, \neg\texttt{l}_n, \texttt{p}\}$. Theorem 3 then tells us that there exists a rule $r \in S$ such that $\texttt{Rule2Clause}(r) = \{\neg\texttt{l}_1, \ldots, \neg\texttt{l}_n, \texttt{p}\}$. As S is closed under antecedent cleaning and transposition, this also means that there exists a rule in S of the form $\texttt{p} \leftarrow \texttt{l}_1, \ldots, \texttt{l}_n$.
"$\Longleftarrow$": Suppose S contains a rule $\texttt{p} \leftarrow \texttt{l}_1, \ldots, \texttt{l}_n$ with $\texttt{l}_1, \ldots, \texttt{l}_n \in L$. Then, $\texttt{Rules2Clauses}(S) \cup \texttt{Lits2Clauses}(L) \models \texttt{p}$ (this follows from the correctness of resolution). □

Theorem 5. *Let $\mathcal{W}$ be an implication based set of formulas and let* $\texttt{l}_1, \ldots, \texttt{l}_n, \texttt{k}$ *be literals such that* $\mathcal{W} \cup \{\texttt{l}_1, \ldots, \texttt{l}_n\}$ *is consistent. It holds that* $\mathcal{W} \cup \{\texttt{l}_1, \ldots, \texttt{l}_n\} \models \texttt{k}$ *iff* $\texttt{k} \in Cl(\mathcal{P}_\mathcal{W} \cup \{\texttt{l}_1 \leftarrow, \ldots, \texttt{l}_n \leftarrow\})$.

Proof.
"$\Longrightarrow$":
Suppose that $\mathcal{W} \cup \{\mathtt{l_1}, \ldots, \mathtt{l_n}\} \models \mathtt{k}$. From the fact that $\mathcal{W} \cup \{\mathtt{l_1}, \ldots, \mathtt{l_n}\}$ is consistent it follows that $\mathtt{Rules2Clauses}(\mathcal{P}_\mathcal{W}) \cup \mathtt{Lits2Clauses}(\{\mathtt{l_1}, \ldots, \mathtt{l_n}\}) \not\models \bot$. From the fact that $\mathcal{W} \cup \{\mathtt{l_1}, \ldots, \mathtt{l_n}\} \models \mathtt{k}$ it follows that $\mathtt{Rules2Clauses}(\mathcal{P}_\mathcal{W}) \cup \mathtt{Lits2Clauses}(\{\mathtt{l_1}, \ldots, \mathtt{l_n}\}) \models \mathtt{k}$. From Theorem 4 it then follows that $\mathcal{P}_\mathcal{W}$ contains a rule $\mathtt{k} \leftarrow \mathtt{l_1}, \ldots, \mathtt{l_i}$ $(0 \leq i \leq n)$. Therefore, it holds that $\mathtt{k} \in Cl(\mathcal{P}_\mathcal{W} \cup \{\mathtt{l_1} \leftarrow, \ldots, \mathtt{l_n} \leftarrow\})$.
"$\Longleftarrow$":
Suppose that $\mathtt{k} \in Cl(\mathcal{P}_\mathcal{W} \cup \{\mathtt{l_1} \leftarrow, \ldots, \mathtt{l_n} \leftarrow\})$. Then, as for every rule $\mathtt{c} \leftarrow \mathtt{a_1}, \ldots, \mathtt{a_m}$ in $\mathcal{P}_\mathcal{W}$ it holds that $\mathcal{W} \models \mathtt{c} \subset \mathtt{a_1}, \ldots, \mathtt{a_m}$, it also holds that $\mathcal{W} \cup \{\mathtt{l_1}, \ldots, \mathtt{l_n}\} \models \mathtt{k}$. □

3 Discussion

When ELP rules are seen as a domain dependent specification of entailment rules, the question then becomes under which conditions this entailment satisfies any reasonable conditions. The current approach seems to be to require no conditions at all on the specific entailment as specified by the ELP in question. However, this allows one to specify quite weird forms of entailment, and it should not come as a surprise that the results can then be quite unusual as well, especially when applied in a broader context. Examples of this are provided in [1] and [2]. The question of what restrictions have to be specified on an ELP in order to obtain a specific property of the outcome is a relevant topic that deserves further study.

References

[1] M. Caminada and L. Amgoud. An axiomatic account of formal argumentation. In *Proceedings of the AAAI-2005*, pages 608–613, 2005.

[2] M.W.A. Caminada. Well-founded semantics for semi-normal extended logic programs. In *Proceedings of the 11th International Workshop of Nonmonotonic Reasoning, special session on answer set programming*, 2006.

[3] C.-L. Chang and R.C.-T Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, Boston, 1973.