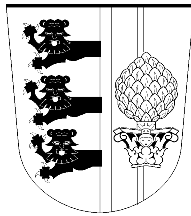
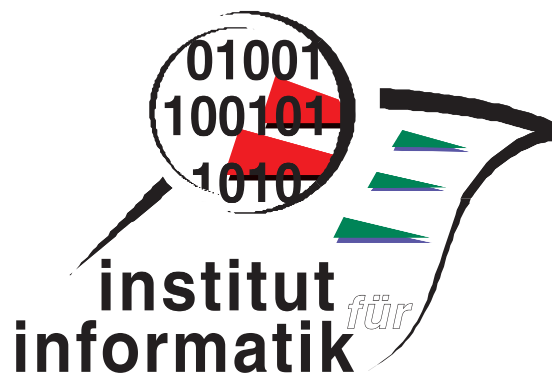
Universität Augsburg



A descriptive Mode Inference for Logic Programs

Ebénézer Ntienjem

Report 1997-05 Dezember 1997



Institut für Informatik
D-86135 Augsburg

Copyright © Ebénézer Ntienjem
Institut für Informatik
Universität Augsburg
D–86135 Augsburg, Germany
http://www.Informatik.Uni-Augsburg.DE
— all rights reserved —

A descriptive Mode Inference for Logic Programs

Ebénézer Ntienjem

December 17, 1997

Abstract

In general, an n-ary predicate (relation) describes the relationship among its arguments, and such that no argument has to be of a special mode. The unification and the resolution (SLDNF-resolution) do capture this state of affair. Hence, the aim of logic programming is in some point approximatively achieved if the system is able to automatically determine the descriptive mode of an n-ary predicate symbol with respect to a logic program. The descriptive mode of an n-ary predicate symbol with respect to a logic program indicates the instantiation of the arguments of that n-ary predicate symbol when it occurs in a goal. To get a sound descriptive mode for an n-ary predicate symbol we consider the abstraction of terms and hence of a set of clauses, the unification of abstract terms and define an NJSLDT-derivation. The descriptive mode inference determines which arguments of an n-ary predicate symbol in a goal become closed or partially instantiated or remain variable. Since the mode of an n-ary predicate symbol does influence the operational semantics of that predicate, this information may be used by a compiler for the purpose of efficiency. The mode inference will also help automatically determine the literal to be selected when constructing an SLDNF-derivation, weaken the condition of allowedness, automatically prove the termination of logic programs for a large class of programs, and detect some errors at compile time.

Keywords: Logic Programming, Unification, Abstract Interpretation, Resolution, Mode

Contents

1 Introduction **2**

2 Objective and related works **3**

3 Preliminaries and motivating examples **5**

4 **Unification by transformation on systems** **7**
4.1 Preliminaries and basic definitions 7
4.2 Transformation rules and soundness 9
4.3 Termination 10
4.4 Lattice on the quotient set of terms 11

5 **NJSLDT-Derivation for mode inference** **13**
5.1 NJSLDT-tree 16

6 **Descriptive mode inference** **16**
6.1 Examples 19

7 **Related works and conclusion** **23**
7.1 Related works 23
7.2 Conclusion 24

1 Introduction

We assume throughout this paper that the reader is acquainted with the basic notions of logic programming. If nothing else is noted, all notations used in the following are borrowed from Apt in [1] or Lloyd in [8] or Schwichtenberg in [18]. We say logic program as a short hand for normal logic program as defined in [8] respectively general logic program as defined in [1].

The mode is in general useful to both the compiler, for optimization, and the programmer, to help when verifying the correctness of the program. A mode of an n-ary predicate symbol defined in a program is a possible n-tuple of the instantiation of arguments of that n-ary predicate symbol in term of some domain. An element of such a domain says something about the degree of instantiation of an argument of an n-ary predicate symbol. Let us denote in the sequel such a domain by M.

In the context of imperative or functional languages, such a domain is the set $M = \{ \text{input}, \text{output} \}$. Note that in imperative or functional languages arguments are passed by pattern matching and a program is evaluated with respect to some fixed order of evaluation. Hence, a mode of an n-ary predicate symbol might be prescribed, that is declared. In this case a mode of an n-ary predicate symbol says how the arguments of this predicate symbol has to be according to the underlying domain M when this predicate symbol occurs and is selected in a goal. Let us call it a *prescriptive* mode.

In the context of logic programming languages, arguments are passed using the unification instead of pattern matching and a logic program is evaluated by the SLDNF-resolution which has no order of evaluation fixed in advance. Because of the unification it is reasonable to say *closed* term instead of input term. Hence,

it is not a good idea, if a mode of an n-ary predicate symbol is prescribed. To keep the spirit of unification and that of SLDNF-resolution, logic programming languages are not augmented with the notion of mode. But a mode of an n-ary predicate symbol may be inferred from a logic program if it is said that a mode of an n-ary predicate symbol says how the arguments of this predicate symbol are instantiated according to the underlying domain M when this predicate symbol occurs and is selected in a goal. Let us call it a *descriptive* mode.

Let us first of all find an adequate domain M for logic programming languages. In addition to closed or variable terms do logic languages allow partially instantiated terms as arguments. Hence, we classify arguments, that is terms, according to the degree of how they are instantiated. That is

$$M = \{ \text{ closed}, \text{partially instantiated}, \text{variable } \}.$$

Since the use of mode in logic programs has been discussed by many researchers with different objective, the domain M is not unique. Warren in [22] uses the set $\{+, -, ?\}$ where "+, −, ?" denotes respectively bound, unbound, and unknown argument; Stroetmann in [21] uses $\{+, -\}$; Stärk in [20] uses $\{ \text{in}, \text{out}, \text{normal} \}$ where "in, out, normal" stands respectively for input, output and normal(logical) argument; Debray in [5] and Debray and Warren in [6] use the set $\{c, d, e, f, nv\}$ where "c, d, e, f, nv" denotes respectively the set of closed terms, the set of don't know terms, the empty set of terms, the set of uninstantiated variables and the set of non variable terms.

This paper is organized as follows. In section 2 we briefly look at the objective and some related works on mode consideration in logic programming. In section 3 we give the syntax of our logic programming language, the notational conventions and motivate our method. In section 4 we revisit the unification as a transformation in a special domain of discourse. In section 5 we briefly describe the NJSLDT-derivation[1], which does recognize infinite derivations. We discuss in section 6 the descriptive mode inference. In section 7 we consider works related to mode for logic programs and give a short conclusion of our mode inference method.

2 Objective and related works

A better look at the objective of the descriptive mode inference is given when considering the intended meaning of a predicate and the formulation of the program for that predicate. The intended meaning of the well-known 3-ary predicate symbol `append` is the description of the fact that the third argument is the result of adding the second argument as a supplement to the first one. Hence, one gets the well known mode (closed, closed, variable) for `append`. The interpretation of this intended meaning of `append` is not unique since one may search for first and/or second argument such that the third argument holds. In this case the mode of `append` is for example (variable, variable, closed).

[1]NJSLDT means "nténe njem" SLD, T for terminating

It may happen that the body of a clause consists of two subgoals in which the same n-ary predicate symbol occurs and that these subgoals have an argument at an argument position instantiated as, say a closed term, in one subgoal and as, say a variable term, in the other. The intended meaning of that predicate remains unchanged, but the order of the evaluation. The evaluation of goals having one of these modes differs on the order of the evaluation of the subgoals derived from each one (see example 6.3 for an illustration of this claim) and therefore the interpretation. This has been seen as the expressive power of logic languages [16]. To define a new set of clauses because of this fact is any more evident if the definition of the n-ary ($n > 1$) predicate symbol is complex. Another reason why avoiding the definition of new sets of clauses can be stated as follows: suppose that the number of elements of the set M is $k \geq 1$ and that an n-ary predicate symbol with $n \geq 0$ is given. Then the number of possible modes of that n-ary predicate symbol is k^n.

Since languages for logic programming do allow partial instantiation of arguments (see for example the second argument of `append` in goal in example 3.1) it is not correct to speak of input argument and output argument as this is the case in imperative respectively functional languages. Hence, we will consider an argument, that is a term, as closed (ground) or partially instantiated or variable.

The informal definition of a descriptive mode of an n-ary predicate symbol with respect to a given logic program indicates which arguments of that n-ary predicate symbol become closed or partially instantiated or remain variable when a literal in which this n-ary predicate symbol occurred has to be selected in a goal. This definition of the mode of an n-ary predicate symbol is more general than the prescriptive, that is declarative, one given in [4, 6], which states that the mode of an n-ary predicate symbol in a logic program indicates how its arguments have to be instantiated when that n-ary predicate symbol appears in a goal; it is a functional definition of a mode of an n-ary predicate symbol. Prescriptive mode or well-known as syntactic mode declaration has been used to prove the completeness of SLDNF-resolution for a class of logic programs [20, 21].

To automatically infer a descriptive mode of an n-ary predicate symbol with respect to a logic program, every closed term is abstracted to an abstract closed term; that means that the real value of a constant term is irrelevant. We will then speak of an abstract logic program respectively an abstract goal. We consider an abtract domain of data descriptions which is also called abstract interpretation. The abstract interpretation is then sound if the data descriptions computed for each program point approximate the set of concrete data that may occur during a program execution. To correctly infer a descriptive mode, an unification of abstract terms is considered, and next a terminating SLD-resolution of a logic program with a goal.

We consider a method which lets the system at compile time infer the mode of an n-ary predicate symbol, when given a logic program. Our method does not consider all possible goals of a given predicate to infer the mode of that n-ary predicate symbol, but just the clauses defining that n-ary predicate symbol and those predicate symbols on which the predicate symbol in consideration depends. When the n-ary predicate symbol does occur in a goal its mode is

then deduced. The method also consider functions as arguments in the definition of predicate symbols. Our method can also be used to verify the consistency of user supplied mode declarations. Such a consistency verification for user supplied mode declarations is necessary because of errors which may be made by the user. One can use this mode inference to improve the efficiency of logic programs [2, 3, 4, 6, 9, 11, 12, 13, 16, 22] or to control the evaluation of logic programs [14] or to prove the completeness of SLDNF-resolution for a large class of programs [20, 21].

3 Preliminaries and motivating examples

Let $\mathcal{V}$ be a countably infinite set of variables, for each $n \geq 0$ a countably infinite set $\mathcal{F}$ of function symbols be given, and for each $n \geq 0$ a countably infinite set $\mathcal{P}$ of predicate symbols be given. Let then the syntactic categories $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ of terms, FOR of formulae, $\mathcal{S}$ of substitutions, $\mathcal{R} \subseteq \mathcal{S}$ of variable renaming substitutions, MGU of most general unifiers of terms be defined as usual. The *falsehood* $\perp$ denotes a formula that is false at all or finitely failed. A *literal* is an atomic formula or a negated atomic formula. A *program clause* or *clause* for short is a formula of the form $\alpha \leftarrow \lambda_1, \cdots, \lambda_n$, where α is an atomic formula which is also called the *head*, $\lambda_1, \cdots, \lambda_n$ is a formula which is also called the *body*, and $n \geq 0$; we write $\alpha \leftarrow \epsilon$, if $n = 0$. Note that ',' in the body stands for '$\wedge$'. A *program goal* or *goal* for short is a clause of the form $\perp \leftarrow \lambda_1, \cdots, \lambda_n$, where $n \geq 0$; we write ϵ if $n = 0$. If no confusion is feared, we also write a goal in the form $\lambda_1, \cdots, \lambda_n$. A *logic program* or *program* for short is a (finite) set of clauses. Instead of considering a logic program to be a set of clauses we let it be the union of the definitions of predicate symbols, where the definition of an n-ary predicate symbol is the set of clauses such that this n-ary predicate symbol does occur in the head of each clause belonging to this set of clauses. Let $vars(t)$ be the set of variables occurring in the term t and $fvars(L)$ be the set of (free) variables occurring in the literal L.

We say that the definition of an n-ary predicate symbol containing exactly m clauses is *well-formed*, if for each $1 \leq i \leq n$ and for all $1 \leq j \leq m$ the arguments $t_{i,j}$ are linearly ordered with respect to the order $\leq$, where $E \leq F$ holds for two terms E and F if there exists some substitution θ such that $F = E\theta$ holds. A logic program is *well-formed*, if the definition of each predicate symbol occurring in this logic program is well-formed. We suppose in the following that each logic program unter consideration is well-formed.

Let us write for the sake of simplicity

$$\begin{array}{rll} [] & \text{for} & nil, \\ [x] & \text{for} & cons(x, nil), \\ [xs, ys] & \text{for} & cons(xs, ys), \\ [x|xs] & \text{for} & cons(x, xs), \\ \vec{x} & \text{for} & x_1, \ldots, x_n, \end{array}$$

where $n \geq 0$, xs and ys are list and x is an element of a list. Let in the sequel u

stands for a term t such that $\text{vars}(t) = \emptyset$, v stands for a term t which is a variable and s stands for a term of the form $f(t)$ such that $\text{vars}(f(t)) \neq \emptyset$.

Let us first of all motivate our method with two sample examples. The first example illustrates the problem that occurs when specifying the modes of an n-ary predicate symbol, and hence the modes of all predicates symbols occurring in a logic program.

Example 3.1 Let us consider the program APPEND and denote a closed term by u, u_1, u_2 etc.

$$\begin{array}{lll} \texttt{append}([], ys, ys) & \leftarrow & \\ \texttt{append}([x|xs], ys, [x|zs]) & \leftarrow & \texttt{append}(xs, ys, zs) \end{array}$$

The set of all modes of append when specifying an argument as closed, + for short, or variable, − for short, is $\{ (+,+,+), (+,+,-), (+,-,-), (-,-,-), (-,-,+), (+,-,+), (-,+,-), (-,+,+) \}$. Suppose now that one consider the goal $\Gamma = \leftarrow \texttt{append}(xs, [u_1, ys], [u_2, u_3])$. Then the specification of the second argument as closed or variable is any more correct. To fully consider this last fact we let an argument be specified as partially instantiated. The number of modes in the set of all modes of append is then $27 = 3^3$, that is very large to be listed. ♢

The next example illustrates information a compiler can obtain from a (prescriptive/descriptive) mode inference to make various improvements.

Example 3.2 Let us consider the following program PP and denote a closed term by u, u_1, u_2 etc.

$$\begin{array}{lll} C_1 : p(h(x), g(x,y)) & \leftarrow & q(x,z), r(f(z), z), s(z,y) \\ C_4 : s(x, u_2) & \leftarrow & q(x, u_2) \end{array}$$

With respect to this set of program clauses one remarks that the variable y occurring in the clause C_1 will always be instantiated to a closed term, if it is not. ♢

When one infers the mode, for example *closed* respectively *variable*, for an n-ary predicate symbol, that does not mean that the argument occurring in that n-ary predicate symbol at the argument position where a closed term respectively a variable term stands has to be a closed respectively variable term in a goal. A closed term means that the argument at this position will become a closed term if it is not; example 6.1 illustrates this claim. That is another reason why we use a closed term instead of "input" and a variable instead of "output".

Before formally discussing the descriptive mode inference for logic programs, let us first of all reconsider in section 4 the unification method for this special purpose. This is necessary because of the two new rules which are introduced when considering abstract terms; an abstract term is a term for which the real value of a constant is not relevant.

4 Unification by transformation on systems

It is important that the unification be fully reconsidered because of the two new rules needed for the purpose of descriptive mode inference. Since we are dealing with unification of abstract terms, i.e. terms in which any closed term is mapped to a symbolic closed term such that rule (3) in definition 4.5 is applicable, the assumption that a closed term unify with any term cannot be wrong. Note that the real value of a constant term, for example 5, is irrelevant in this context.

4.1 Preliminaries and basic definitions

We mainly deal with term in arguments and terms are inductively defined over $\mathcal{F}$ and $\mathcal{V}$. An important point by an automatic mode inference is the classification of terms occurring in a program with respect to how they are instantiated. For simplicity, a term t may be:

- *closed*, if $\mathrm{vars}(t) = \emptyset$,
- *partially instantiated*, if t is of the form $f(\vec{t'})$ with $\mathrm{vars}(\vec{t'}) \neq \emptyset$,
- *variable*, if $t = v$.

Definition 4.1 *A substitution is a function $\sigma : \mathcal{V} \longrightarrow \mathcal{T}_{\mathcal{F},\mathcal{V}}$ such that for finitely many $v \in \mathcal{V}$ the following holds: $v\sigma \neq v$. The domain of σ is the set* $\mathrm{dom}(\sigma) = \{v | v\sigma \neq v\}$. *The set of variables introduced by σ is* $\mathrm{ivars}(\sigma) = \bigcup_{v \in \mathrm{dom}(\sigma)} \mathrm{vars}(v\sigma)$.

We denote a substitution σ by $\{t_1/v_1, \ldots, t_n/v_n\}$ if $dom(\sigma) = \{v_1, \ldots, v_n\}$ and if $v_i\sigma = t_i$ for $1 \leq i \leq n$. The function σ may naturally be extended to terms by a recursive definition.

We suppose that the reader is acquainted with the basic notions on substitutions. A term t is more general than an other term r, denoted $t \leq r$, if and only if there exists a substitution θ such that $r = t\theta$ holds. A substitution σ is more general than an other substitution θ, denoted by $\theta \leq \sigma$, iff there exists a substitution ρ such that $\theta = \sigma \circ \rho$, where $\circ$ denotes the composition of substitutions, which we also denote by $\sigma\rho$.

Let us consider an equivalence relation on terms to formally achieve the abstraction of terms as noted in section 2. Our consideration of an equivalence relation on terms is a little different from the usual one in the sense that closed terms are identified. We now define the relation $\cong$ on $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ as follows:

- let the terms t and r be closed. Then $t \cong r$.
- let the terms t and r not be closed. Then $t \cong r$ if and only if there exists substitutions θ and σ such that $r \cong t\sigma$ and $t \cong r\theta$ hold.

The relation $\cong$ is reflexive, symmetric and transitive. We denote by $\mathcal{T}_{\mathcal{F},\mathcal{V}}/\cong$ the quotient set of $\mathcal{T}_{\mathcal{F},\mathcal{V}}$. Let us for simplicity use the word term for formula as well and *var* for *vars* or *fvars*. We now define the unification of first order abstract terms as a set of non-deterministic rules of transformation. This elegant approach is due to [10, 19].

A pair of terms (for short a pair) is a multiset of two terms, denoted by $\langle s,t\rangle$; we call a substitution θ an abstract unifier of a pair $\langle s,t\rangle$ if $s\theta \cong t\theta$. A system of terms (for short a system) is a multiset of pairs; a substitution θ is an abstract unifier of a system if it unifies each pair. We denote by $U_a(S)$ the set of abstract unifiers of a system S.

Definition 4.2 *Let $\langle v,t\rangle$ be a pair in a system S. v is called a solved variable in a system S if v is a variable which does not occur anywhere else in S, and such that $v \notin var(t)$. $\langle v,t\rangle$ is in solved form if v is a solved variable. A system is in solved form if all its pairs are in solved form.*

Definition 4.3 *A substitution σ is a most general abstract unifier (for short a-mgu) of a system S iff*

(i) $dom(\sigma) \subseteq var(S)$,

(ii) $\sigma \in U_a(S)$ and

(iii) for each $\theta \in U_a(S)$ it holds $\theta \leq \sigma$.

Definition 4.4 *Let $U_a(S)$ be the set of all abstract unifiers of the system S, θ and σ be abstract substitutions.*

(i) $\theta \in U_a(S) \overset{\text{def}}{\Longleftrightarrow}$ for each $\langle s,t\rangle \in S \quad s\theta \cong t\theta$.

(ii) $\theta \approx \sigma \overset{\text{def}}{\Longleftrightarrow}$ for each $x \in \mathcal{V} \quad x\theta \cong x\sigma$.

(iii) $\theta \leq \sigma \overset{\text{def}}{\Longleftrightarrow}$ there exists ρ such that $\quad \theta \cong \sigma\rho$.

The following lemma shows the importance of a system in solved form. Let then S be a finite system.

Lemma 4.1 *Let $S = \{\langle v_1,t_1\rangle,\ldots,\langle v_n,t_n\rangle\}$ be in solved form.*
If $\sigma = \{t_1/v_1,\ldots,t_n/v_n\}$, then σ is an idempotent a-mgu *of S. Furthermore, for a substitution $\theta \in U_a(S)$, we have $\theta \cong \sigma\theta$.*

Proof Let θ a substitution which solves the system S be given. Then $v_i\theta \cong t_i\theta$ for each $\langle v_i,t_i\rangle \in S$. That means $v_i\theta \cong v_i\sigma\theta$ for $1 \leq i \leq n$ and $x\theta \cong x\theta$ otherwise. Since $dom(\theta) \cap ivars(\theta) = \emptyset$ by the definition of a system in solved form, θ is an a-mgu and is idempotent. $\square$

From this definition we investigate on a new special unification algorithm on terms.

4.2 Transformation rules and soundness

Let t and r be any terms, u and u' be closed terms in the sequel. We assume that the representation of any closed term in the following definition is a good candidate in the sense that it unifies with any not necessary closed term as expressed by rule (7) below. Hence, in this abstract interpretation closed terms are identified even if their representation is not the same.

Definition 4.5 (transformation rules) *Let S be a finite system, $\sigma = \{t/v\} = \langle v, t\rangle$ be in solved form, $v \in var(S)$ and f respectively g be n-ary respectively m-ary function symbols. Let furthermore assume that $f(\vec{t})$ and $g(\vec{r})$ are not closed in rules (3) and (9). The following defines the set of transformation rules.*

$$
\begin{aligned}
\{\langle t,t\rangle\}\cup S &\Rightarrow S && (1)\\
\{\langle u,u'\rangle\}\cup S &\Rightarrow S && (2)\\
\{\langle f(\vec{t}),g(\vec{r})\rangle\}\cup S &\Rightarrow \{\langle t_1,r_1\rangle,\ldots,\langle t_n,r_n\rangle\}\cup S \textit{ if } f=g \textit{ and } m=n && (3)\\
\{\langle v,t\rangle\}\cup S &\Rightarrow \{\langle v,t\rangle\}\cup S\sigma \quad \textit{if } v\notin var(t) && (4)\\
\{\langle t,v\rangle\}\cup S &\Rightarrow \{\langle v,t\rangle\}\cup S && (5)\\
\{\langle f(\vec{s}),u\rangle\}\cup S &\Rightarrow \{\langle u,f(\vec{s})\rangle\}\cup S \quad \textit{if } var(\vec{s})\neq\emptyset && (6)\\
\{\langle u,f(\vec{s})\rangle\}\cup S &\Rightarrow \{\langle f(\vec{u'}),f(\vec{s})\rangle\}\cup S \quad \textit{if } var(\vec{s})\neq\emptyset && (7)\\
\{\langle v,t\rangle\}\cup S &\Rightarrow \textit{fail} \quad \textit{if } t=f(r) \textit{ and } v\in var(r) && (8)\\
\{\langle f(\vec{t}),g(\vec{r})\rangle\}\cup S &\Rightarrow \textit{fail} \quad \textit{if } f\neq g,\ m\neq n,\ \vec{t}\neq\vec{u} \textit{ and } \vec{r}\neq\vec{u'} && (9)
\end{aligned}
$$

Note that an empty system is allowed in this definition, that rule (1) is a special case of rule (3) and that rule (9) is the negation of rules (3), (1) and (2). It will also be noted that transformation rule (3) is usually called *term decomposition*. Note that this unification algorithm may find an abstract unifier for a system which fails under the well-known unification algorithm as the example below shows. To deal with formulae is straightforward.

$\Rightarrow$ defines on $\mathcal{T}_{\mathcal{F},\mathcal{V}}/\cong$ a relation which is reflexive and transitive. Hence, for a system S there exists some sequence of transformations $S \Rightarrow \cdots \Rightarrow S'$, which we denote $S \Rightarrow^* S'$. If S' is in solved form, then S' is an abstract unifier $\theta = \sigma_{S'}$ of the system S.
Let us, before considering soundness result, fix this procedure in mind by a simple example.

Example 4.1 In this example we misuse the ordinary set meaning by considering the set as an ordered list, and hence always choose the first element of the list and insert a pair in solved form at the end of the list. $\stackrel{(i)}{\Rightarrow}$, $(1) \leq i \leq (9)$, means that the rule (i) is used to get the next system.

$$
\begin{aligned}
&\{\langle f(h(x,z),g(u_7,y),u_8), f(u_9,g(y,x),k(z,u_6))\rangle\}\\
&\stackrel{(3)}{\Rightarrow} \{\langle h(x,z),u_9\rangle, \langle g(u_1,y),g(y,x)\rangle, \langle u_8,k(z,u_6)\rangle\}\\
&\stackrel{(6)}{\Rightarrow} \{\langle u_9,h(x,z)\rangle, \langle g(u_1,y),g(y,x)\rangle, \langle u_8,k(z,u_6)\rangle\}
\end{aligned}
$$

$$\stackrel{(7)}{\Rightarrow} \{\langle h(u_4,u_5),h(x,z)\rangle,\langle g(u_1,y),g(y,x)\rangle,\langle u_8,k(z,u_6)\rangle\}$$
$$\stackrel{(3)}{\Rightarrow} \{\langle x,u_4\rangle,\langle z,u_5\rangle,\langle g(u_1,y),g(y,x)\rangle,\langle u_8,k(z,u_6)\rangle\}$$
$$\stackrel{(4)}{\Rightarrow} \{\langle z,u_5\rangle,\langle g(u_1,y),g(y,u_4)\rangle,\langle u_8,k(z,u_6)\rangle,\langle x,u_4\rangle\}$$
$$\stackrel{(4)}{\Rightarrow} \{\langle g(u_1,y),g(y,u_4)\rangle,\langle u_8,k(u_5,u_6)\rangle,\langle x,u_4\rangle,\langle z,u_5\rangle\}$$
$$\stackrel{(3)}{\Rightarrow} \{\langle y,u_1\rangle,\langle u_4,y\rangle,\langle u_8,k(u_5,u_6)\rangle,\langle x,u_4\rangle,\langle z,u_5\rangle\}$$
$$\stackrel{(4)}{\Rightarrow} \{\langle u_4,u_1\rangle,\langle u_8,k(u_4,u_6)\rangle,\langle x,u_4\rangle,\langle z,u_5\rangle,\langle y,u_1\rangle\}$$
$$\stackrel{(2)}{\Rightarrow} \{\langle u_8,k(u_5,u_6)\rangle,\langle x,u_4\rangle,\langle z,u_5\rangle,\langle y,u_1\rangle\}$$
$$\stackrel{(7)}{\Rightarrow} \{\langle k(u_2,u_3),k(u_5,u_6)\rangle,\langle x,u_4\rangle,\langle z,u_5\rangle,\langle y,u_1\rangle\}$$
$$\stackrel{(2)}{\Rightarrow} \{\langle x,u_4\rangle,\langle z,u_5\rangle,\langle y,u_1\rangle\}.$$

◇

Lemma 4.2 *If $S \Rightarrow S'$ using a transformation rule from definition 4.5 and $\theta \in U_a(S')$, then $\theta \in U_a(S)$.*

Proof The point to be proved lies in rule (4). Let $\sigma = \{t/v\} = \langle v,t\rangle$ be in solved form and θ be any abstract substitution. Suppose $\{\langle v,t\rangle\} \cup S \Rightarrow \{\langle v,t\rangle\} \cup S\sigma$. If $v\theta \cong t\theta$, then $v\theta \cong t\theta \cong (v\sigma)\theta \cong v\sigma\theta$. Hence,
$\theta \in U_a(\{\langle v,t\rangle\} \cup S) \iff v\theta \cong t\theta$ and $\theta \in U_a(S)$. That means
$v\theta \cong t\theta$ and $\theta \in U_a(S) \iff v\theta \cong t\theta$ and $\sigma\theta \in U_a(S)$. That means
$v\theta \cong t\theta$ and $\sigma\theta \in U_a(S) \Leftarrow v\theta \cong t\theta$ and $\theta \in U_a(S\sigma)$. That means
$v\theta \cong t\theta$ and $\theta \in U_a(S\sigma) \Leftarrow \theta \in U_a(\{\langle v,t\rangle\} \cup S\sigma)$. □

The soundness is then a straightforward induction on the length of a transformation sequence.

Theorem 4.1 (Soundness) *If $S \Rightarrow^* S'$ and S' is in solved form, then $\sigma_{S'}$ is an a-mgu of S.*

Proof By induction on the length of the transformation sequence and use of lemma 4.2, which trivially means $\theta \in U_a(S')$ imply $\theta \in U_a(S)$ for some abstract substitution θ. Hence, $\sigma_{S'}$ is an a-mgu of S. □

4.3 Termination

In this subsection we look at the termination of a sequence generated by the transformation rules.

Lemma 4.3 (Termination) *The relation $\Rightarrow$ does not produce infinite transformation sequences.*

Proof First of all let us recursively define the (pseudo-)length, $|t|$, of a term t as follows:

$$|t| = \begin{cases} 0 & \text{if } t \text{ is a closed term} \\ 1 & \text{if } t \text{ is a variable} \\ 1 + \sum_{i=1}^{n} |t_i| & \text{if } t = f(t_1, \ldots, t_n) \text{ and } \exists i,\ 1 \leq i \leq n,\ t_i \text{ is not closed.} \end{cases}$$

Let us now define, for any finite system S, a complexity measure

$$\mu\colon \mathrm{SYS} \longrightarrow \mathbb{N} \times \mathbb{N}, \quad S \longmapsto \langle m, n \rangle,$$

where SYS is the set of all systems S, m is the number of unsolved variables v occurring in the system, and n is the sum of the (pseudo-)length of the terms occurring in the system. It is well known that $\mathbb{N} \times \mathbb{N}$ with the lexicographic ordering on $\langle m, n \rangle$ is well-founded. One easily observes the following:

- rule (1) lets n stationary, if t is a closed term. Since S is finite and rule (1) does not increase m, S may become empty or another rule applies after some finite step. If t is not a closed term, then rule (1) may decrease n.
- rule (3) decreases n and does not increase m.
- rule (2) lets n stationary and does not increase m.
- rule (4) decreases m and does not increase n.
- rules (5)–(7) each lets m and n stationary and the relation $\Rightarrow$ does not cycle on these rules.

Hence, the relation $\Rightarrow$ is well-founded, and each transformation sequence terminates in a finite system to which no transformation rule applies, that is a system which is either in solved form or is failed. □

4.4 Lattice on the quotient set of terms

The ordering $\leq$ on $\mathcal{T}_{\mathcal{F},\mathcal{V}}$ induces an ordering $\leq$ on $\mathcal{T}_{\mathcal{F},\mathcal{V}}/\cong$. Let us in addition extend this ordering on $\mathcal{T}_{\mathcal{F},\mathcal{V}}/\cong$ as follows:

- a variable term is more general than a partially instantiated term respectively a closed term.
- a partially instantiated term is more general than a closed term.

We now show that $(\mathcal{T}_{\mathcal{F},\mathcal{V}}/\cong, \succeq)$ is a (complete) lattice, where the relation $\succeq$ stand for $\geq$. Similar to the greatest lower bound (*glb*) respectively the least upper bound (*lub*) of the usual lattice we define the greatest lower instance (*gli* for short) respectively the least upper instance (*lui* for short) as follows. Let t and r be terms.

1. The term t' is a lower instance of the terms t and r if and only if $t \succeq t'$ and $r \succeq t'$. The term t' is an gli(t, r) if and only if t' is a lower instance of t and r and if r' is another lower instance of t and r, then $t' \succeq r'$.
2. The term t' is a upper instance of the terms t and r if and only if $t' \succeq t$ and $t' \succeq r$. The term t' is an lui(t, r) if and only if t' is a upper instance of t and r and if r' is another upper instance of t and r, then $r' \succeq t'$.

The extension of gli and lui to a set of terms is straightforward. Since we consider terms modulo variable renaming an gli respectively an lui is unique. Following Plotkin in [15] and Reynolds in [17] we have

Lemma 4.4 *Let $\perp$ be a term such that for all $t \in \mathcal{T}_{\mathcal{F},\mathcal{V}}/\cong$ it holds $t \succeq \perp$. Then $(\mathcal{T}_{\mathcal{F},\mathcal{V}}/\cong \cup \{\perp\}, \succeq)$ is a complete lattice.*

We now state and prove an useful lemma when determining the gli respectively the lui of a set of terms.

Lemma 4.5 *Let S_1 and S_2 be sets of terms. Then*

(i) $lui(S_1 \cup S_2) \cong lui(lui(S_1), lui(S_2))$ *and*

(ii) $gli(S_1 \cup S_2) \cong gli(gli(S_1), gli(S_2))$.

Proof is well-known. □

The following corollary shows that gli and lui are associative.

Corollary 4.1 *Let S_1, S_2 and S_3 be sets of terms. Then*

(i) $lui(S_1 \cup S_2 \cup S_3) \cong lui(lui(S_1), lui(S_2 \cup S_3)) \cong lui(lui(S_1 \cup S_2), lui(S_3))$.

(ii) $gli(S_1 \cup S_2 \cup S_3) \cong gli(gli(S_1 \cup S_2), gli(S_3)) \cong gli(gli(S_1), gli(S_2 \cup S_3))$.

gli and lui are naturally extended to substitutions. Lemma 4.5 also holds if a set of substitutions is considered then. With all these results and methods in mind we are able to derive or give a method to derive a mode of an n-ary predicate symbol with respect to a given program.

5 NJSLDT-Derivation for mode inference

In this section we introduce a variant of the SLD-derivation for abstract logic programs, the NJSLDT-derivation. This variant is needed for the purpose of mode inference since a mode of an n-ary predicate is a (logical) consequence of the mode of those predicate symbls on which it depends. The derivation has to be finite since a mode of an n-ary predicate symbol has to be determined at compile time. We assume familiarity with the SLD-resolution.

We assume that a given logic program P is constructed using the terms in $\mathcal{T}_{\mathcal{F},\mathcal{V}}/\cong$. We call it the abstraction of P and denote it by P_a. P and P_a differ only on the abstraction of closed terms.

The unary operator $\neg$ will in the following be ignored, because the negation does not have an effect on the behavior of the inference of a mode of an n-ary predicate symbol. That means that the way the arguments of an n-ary predicate symbol are instantiated when this appears in a goal does not depend on the fact that this goal is a negative literal.

Let P_a be given and $G = \leftarrow \lambda_1, \cdots, \lambda_n$ be an abstract goal. The intended meaning of $P_a \cup \{G\}$ is: given an abstract logic program P_a and an abstract goal G determine a mode of the goal G with respect to P_a.

A mode for $P_a \cup \{G\}$ is an idempotent abstract substitution θ in the sense of section 4 such that $\mathrm{dom}(\theta) \subseteq \bigcup_{i=1}^{n} \mathrm{var}(\lambda_i)$.

Such a mode determination is better obtain by derivation. The following definitions are inspired from the SLD-definition since the data descriptions computed for each program point have to approximate the set of concrete data which may occur during a program execution. In the following we suppose a computation rule be given.

We consider an NJSLDT-resolution, that is an SLD-resolution, where a literal in a goal is removed if it is already selected and used with the same clause in a former resolution step.

Definition 5.1 *Let $G = \leftarrow \lambda_1, \cdots, \lambda_n$ be an abstract goal, $\kappa = \alpha \leftarrow L_1, \cdots, L_k$ be a variant of a clause in P_a such that $var(\kappa) \cap var(G) = \emptyset$, S be the set of pairs (selected literal, used clause), and θ be an abstract substitution. Then G' and S' are derived from G and S using θ and κ if there exists an $j \in \{1, \ldots, n\}$ such that*

(i) λ_j is the selected literal, $\theta = a\text{-}mgu(\alpha, \lambda_j)$ and

(ii) if there exists a variable renaming substitution ρ and $(L, \kappa) \in S$ such that $\lambda_i \leq L$ or $L \leq \lambda_i$ and $var(L\rho) = var(\lambda_i)$, then
$G' = \lambda_1, \cdots, \lambda_{j-1}, \lambda_{j+1}, \cdots, \lambda_n$ and $S' = S$,
else $G' = (\lambda_1, \cdots, \lambda_{j-1}, L_1, \cdots, L_k, \lambda_{j+1}, \cdots, \lambda_n)\theta$ and $S' = S \cup \{(\lambda_j, \kappa)\}$.

An NJSLDT-derivation of $P_a \cup \{G\}$ consists of four sequences

$$G = G_0, G_1, G_2, \ldots \qquad \text{of goals,}$$

$C_1, C_2, C_3, \ldots$ of clause variantes,
$\theta_1, \theta_2, \theta_3, \ldots$ of substitutions and
$S^0 = S, S^1, S^2, \ldots$ of sets of pairs (literal, clause)

such that G_{i+1} and S^{i+1} are derived from G_i and S^i using θ_{i+1} and C_{i+1}.

An NJSLDT-derivation of $P_a \cup \{G\}$ is finite or infinite. An NJSLDT-derivation of $P_a \cup \{G\}$ is *successful* if it is finite and has the empty goal ϵ as the last goal in the derivation. An NJSLDT-derivation of $P_a \cup \{G\}$ is *failed* if it is finite and has the goal fail as the last goal in the derivation. A successful NJSLDT-derivation has length n if $G_n = \epsilon$.

Definition 5.2 *A substitution θ is a* computed mode *for $P_a \cup \{G\}$, if there exists a successful NJSLDT-derivation $G = G_0, G_1, G_2, \ldots, G_n$, $C_1, C_2, C_3, \ldots, C_n$, $\theta_1, \theta_2, \theta_3, \ldots, \theta_n$ and $S^0 = S, S^1, S^2, \ldots, S^n$ of $P_a \cup \{G\}$ such that θ is obtained by restricting the composition $\theta_1\theta_2\theta_3 \cdots \theta_n$ to the variables of G.*

A mode θ for $P_a \cup \{\leftarrow \lambda_1, \cdots, \lambda_n\}$ is *acceptable* if and only if for each $i \in \{1, \ldots, n\}$ $P_a \cup \{\leftarrow \lambda_i\theta\}$ has a successful NJSLDT-derivation.

We first of all prove that the deletion of a literal that occurs in a goal according to definition 5.1 does not worsen a computed mode. For this sake, let us consider a definition wherein the set S does contain the pairs literal and used clause even if such a pair were already used in a former step of the derivation. The set S is maintained in this definition for the sake of compatibility.

Definition 5.3 *Let $G = \leftarrow \lambda_1, \cdots, \lambda_n$ be a goal, $\kappa = \alpha \leftarrow L_1, \cdots, L_k$ be a variant of a clause in P such that $var(\kappa) \cap var(G) = \emptyset$, S be the set of pairs (selected literal, used clause), and θ be an abstract substitution. Then G' and S' are derived from G and S using θ and κ if there exists an $j \in \{1, \ldots, n\}$ such that*

(i) λ_j is the selected literal, $\theta = a\text{-}mgu(\alpha, \lambda_j)$ and

(ii) $G' = (\lambda_1, \cdots, \lambda_{j-1}, L_1, \cdots, L_k, \lambda_{j+1}, \cdots, \lambda_n)\theta$ and $S' = S \cup \{(\lambda_j, \kappa)\}$.

We are now able to formulate the lemma stating the correctness of the deletion of some particular literals occurring in the goal and used in a former derivation step with the same clause.

Lemma 5.1 *Let P_a and Γ_a be given. If θ is a computed mode for $P_a \cup \{\Gamma_a\}$ using definition 5.3, then there exists a computed mode ϑ for $P_a \cup \{\Gamma_a\}$ using definition 5.1 such that $\theta \cong \vartheta\sigma$ and $ivars(\theta) = ivars(\vartheta\rho)$ hold for some substitution σ and variable renaming substitution ρ.*

Proof By induction on the length n of an NJSLDT-derivation. The length n is represented as a multiset of the indices of the literals in an NJSLDT-derivation. It is well-known that the lexicographical ordering on multiset over $\mathbb{N}$ is well-founded. A multiset $\{1, \ldots, n\}$ is *minimal* with respect to $P_a \cup \{\Gamma_a\}$ if for all

$j \in \{1, \ldots, n\}$ it holds that $\theta_1 \cdots \theta_{j-1}\theta_{j+1} \cdots \theta_n$ is not a computed mode for $P_a \cup \{\Gamma_a\}$.

If the length n of the derivation of $P_a \cup \{\Gamma_a\}$ using definition 5.3 is minimal, then we are done. In this case it holds $m = n$ and $\vartheta \cong \theta$. Let us therefore suppose that the length n is not minimal at all. To construct a finite NJSLDT-derivation of $P_a \cup \{\Gamma_a\}$ of length m from that of $P_a \cup \{\Gamma_a\}$ of length n we show for the multiset $n = \{l_1, \ldots, l_\nu\}$ that $\theta \cong \vartheta\sigma$ and $\text{ivars}(\theta) = \text{ivars}(\vartheta\rho)$ hold for some j in n and for some substitution σ and variable renaming substitution ρ. Since the derivation is finite, the multiset n is finite too. Hence, it suffices to show that $\theta \cong \vartheta\sigma$ and $\text{ivars}(\theta) = \text{ivars}(\vartheta\rho)$ for some j in n, substitution σ and variable renaming substitution ρ. Suppose the derivation uses a literal λ_j and a clause κ_{j+1} which were already used in a former step, say k. Suppose in addition that in step k the selected literal is λ, and that it hold that $\lambda \leq \lambda_j$ or $\lambda_j \leq \lambda$ and $\text{vars}(\lambda) = \text{vars}(\lambda_j\rho)$ for some variable substitution ρ. Hence, using definition 4.4 (iii), it holds that $\vartheta \cong \theta\sigma$ and $\text{ivars}(\theta) = \text{ivars}(\vartheta\rho)$. Now let the new derivation be obtained by deleting the literal, say λ_j, in the step G_j to G_{j+1}, that is by strictly applying the definition 5.1 above. Then let m be $\{l_1, \ldots, l_{j-1}, l_{j+1} \ldots, l_\nu\}$. It holds then $\{l_1, \ldots, l_{j-1}, l_{j+1} \ldots, l_\nu\} < \{l_1, \ldots, l_{j-1}, l_j, l_{j+1} \ldots, l_\nu\}$; that means $m \leq n$. Applying this method repeatedly we get the result, since the derivation is finite. □

Let us now establish the relation between an acceptable mode and a computed mode.

Theorem 5.1 *Let P_a be given and G be an abstract goal. Then every computed mode for $P_a \cup \{G\}$ is an acceptable mode.*

Proof By induction on the length of the NJSLDT-derivation. Let an abstract goal be $G = \leftarrow \lambda_1, \cdots, \lambda_n$ and $\theta_1, \ldots, \theta_m$ be a sequence of a-mgu's used in a mode of $P_a \cup \{G\}$. We have to show that $\theta_1 \cdots \theta_m$ is acceptable for $P_a \cup \{\leftarrow \lambda_1, \cdots, \lambda_n\}$. The case $G = \epsilon$ is so obvious that we always neglect it.

Suppose $m = 1$. Then $G = \leftarrow \lambda_1$ and $S^0 = \emptyset$. The program P_a has then an unit clause, say $\alpha \leftarrow$, such that $\lambda_1\theta_1 = \alpha\theta_1$. Since $\alpha\theta_1 \leftarrow$ is an instance of a unit clause of P_a, it is clear that θ_1 is a mode for $P_a \cup \{\leftarrow \lambda_1\}$.

Suppose now $m > 1$, $S^{m-1} \neq \emptyset$ and $\theta_1, \ldots, \theta_m$ is a sequence of a-mgu's used in an NJSLDT-derivation of $P_a \cup \{G\}$ of length m. Let $C = \alpha \leftarrow L_1, \cdots, L_k$ be a variant of a clause and λ_i be the selected literal of G. Two cases arise.

`case 1:` there exists (λ_j, C) in S^{m-1} and variable renaming substitution ρ such that $\lambda_i \leq \lambda_j$ or $\lambda_j \leq \lambda_i$ and $\text{vars}(\lambda_i) = \text{vars}(\lambda_j\rho)$ hold. Then the NJSLDT-derivation of $P_a \cup \{G\}$ has already been considered in a former derivation step. Hence, from Lemma 5.1 removing the subgoal λ_i does not affect the resulting substitution. By the induction hypothesis $\theta_1 \cdots \theta_m$ is an acceptable mode for $P_a \cup \{\leftarrow \lambda_1, \cdots, \lambda_{i-1}, \lambda_{i+1}, \cdots, \lambda_n\}$. The set S^{m-1} remains unchanged.

`case 2:` the condition in case 1 does not hold. Then by the induction hypothesis $\theta_1 \cdots \theta_m$ is an acceptable mode for

$$P_a \cup \{\leftarrow \lambda_1, \cdots, \lambda_{i-1}, L_1, \cdots, L_k, \lambda_{i+1}, \cdots, \lambda_n\}.$$

Hence $\theta_1 \cdots \theta_m$ is an acceptable mode for $P_a \cup \{\leftarrow L_1, \cdots, L_k\}$. That means $\lambda_i\theta_1 \cdots \theta_m = \alpha\theta_1 \cdots \theta_m$. Hence, $\theta_1 \cdots \theta_m$ is an acceptable mode for $P_a \cup \{\leftarrow \lambda_i\}$. □

Since the NJSLDT-derivation is a special case of the SLD-derivation and since the SLD-derivation is sound, the abstract interpretation is sound as we claim in the introduction. In the next subsection we just, for the sake of illustration later, give the definition of NJSLDT-tree.

5.1 NJSLDT-tree

We inductively define an NJSLDT-tree as usual, top down.

Definition 5.4 *Let P be a program and Γ be a goal. Then the NJSLDT-tree for $P \cup \{\Gamma\}$ is defined as follows:*

(i) a node of the tree is a pair $\langle \Gamma, S\rangle$ consisting of a goal Γ and a set S.

(ii) a leaf of the tree is a node consisting of a goal $\Gamma \in \{\ \epsilon, fail\ \}$ and a set S.

(iii) let the pair consisting of a goal $\leftarrow \lambda_1, \cdots, \lambda_n$ and a set S be a node in the tree. Suppose that there exists i an element of the set $\{1, \ldots, n\}$ and $C_j = \alpha_j \leftarrow L_{j,1}, \cdots, L_{j,k(j)}$ in P such that $\theta = a\text{-}mgu(\lambda_i, \alpha_j)$. Then, if there exists (λ_j, C_j) in S and a variable renaming substitution ρ such that $\lambda_i \leq \lambda_j$ or $\lambda_j \leq \lambda_i$ and $vars(\lambda_j) = vars(\lambda_i\rho)$ hold, then the pair
$\langle \lambda_1 \wedge \cdots \wedge \lambda_{i-1} \wedge \lambda_{i+1} \wedge \cdots \wedge \lambda_n, S\rangle$
is a descendant node, else the pair consisting of the goal
$\leftarrow (\lambda_1, \cdots, \lambda_{i-1}, L_{j,1}, \cdots, L_{j,k(j)}, \lambda_{i+1}, \cdots, \lambda_n)\theta$
and the set $S \cup \{(\lambda_i, C_j)\}$ is a descendant node.

(iv) the root is the node consisting of the pair $\langle \Gamma, \emptyset\rangle$.

Three examples of NJSLDT-trees are given in subsection 6.1 in combination with the inference of the mode of some predicate symbols with respect to some program.

6 Descriptive mode inference

The mode of an n-ary predicate symbol with respect to a program represents statement about computations that are possible from it. From this claim mode information has been studied for the sake of making logic programs as efficient as functional or imperative ones [2, 3, 4, 6, 9, 11, 12, 13, 16, 22].

Our aim in this section is to automatically infer the mode of an n-ary predicate symbol from a given logic program while conserving the mathematical intention of a relation. This is a descriptive mode inference. Hence, our interest is not focused on a syntactic mode declaration of logic programs, because doing so we will lose the expressive power of logic programs. The intended meaning of an

n-ary predicate is preserved when the descriptive mode does consider the most general use of that n-ary predicate; that is its application not only with one meaning.

In subsection 4.3 we consider the *gli* and the *lui* of a set of abstract terms. Without loss of generality the extension of lui respectively gli to (idempotent) substitutions is as follows: let $\theta = \{t_1/x_1, \ldots, t_m/x_m\}$ and $\sigma = \{r_1/x_1, \ldots, r_m/x_m\}$ be substitutions. Then $\text{lui}(\theta, \sigma) = \{\text{lui}(t_i/x_i, r_i/x_i) \mid 1 \leq i \leq m\}$ and $\text{gli}(\theta, \sigma) = \{\text{gli}(t_i/x_i, r_i/x_i) \mid 1 \leq i \leq m\}$ wherein $\text{lui}(t_i/x_i, r_i/x_i) = \text{lui}(t_i, r_i)/x_i$ and
$\text{gli}(t_i/x_i, r_i/x_i) = \text{gli}(t_i, r_i)/x_i$. We now formally define the inferred mode of an n-ary predicate symbol with respect to a given program.

Definition 6.1 *Let P be a program and p be an n-ary predicate symbol occurring in P. The* inferred *mode of p with respect to P, denoted by $I_m^P(p)$, is the least acceptable mode for $P \cup \{\leftarrow p(\vec{v})\}$; that is $I_m^P(p) = lui(\{\theta \mid \theta$ is an acceptable mode for $P \cup \{\leftarrow p(\vec{v})\}\})$.*

Note that if $\{\theta \mid \theta$ is an acceptable mode for $P \cup \{\leftarrow p(\vec{v})\}\}$ is empty, then $I_m^P(p) = \varepsilon$, the identity substitution. Let us in the following use a literal λ for a predicate symbol q as well, since what is meant will be clear from the context.

Definition 6.2 *Let $\leftarrow q(t_1, \ldots, t_n)$ be a goal and $dom(I_m^P(q)) = \{v_1, \ldots, v_n\}$. Suppose that $var(q(\vec{t})) \cap dom(I_m^P(q)) = \emptyset$. Then $I_c^P(q(\vec{t})) = \{t_1/v_1, \ldots, t_n/v_n\}$ is a* called*mode of q with respect to P. That means*

$$q(t_1, \ldots, t_n) = q(v_1, \ldots, v_n) I_c^P(q(\vec{t})).$$

Combining these two previous definitions we next define the instantiated mode of a goal literal.

Definition 6.3 *Let λ_i be a literal, $I_m^P(\lambda_i)$ be the inferred mode of λ_i, and $I_c^P(\lambda_i)$ be a called mode of λ_i and such that $dom(I_c^P(\lambda_i)) = dom(I_m^P(\lambda_i))$. Then the instantiated mode or goal mode, $I_g^P(\lambda_i)$, for the literal λ_i is $I_g^P(\lambda_i) = gli(I_m^P(\lambda_i), I_c^P(\lambda_i))$.*

We assume that the mode of an n-ary predicate symbol p with respect to a given program is variable independent and closed term independent, that means $\theta \in I_m^P(p)$ implies $\forall \sigma \in \mathcal{R} : \theta\sigma \in I_m^P(p)$. Since a closed term is a gli of any set of terms, the instantiated mode for a literal does always exist. To illustrate this fact let us suppose $I_m^P(\lambda) = \{u/v\}$ and $I_c^P(\lambda) = \{g(y)/v\}$. Then $I_g^P(\lambda) = \{g(y)/v, u/y\}$ where u stands for a closed term. This is due to the fact that closed terms are identified.
The following lemma is an immediate consequence of the definitions.

Lemma 6.1 *Let $G = \leftarrow \lambda_1, \cdots, \lambda_n$ be a goal, λ_i, $1 \leq i \leq n$, be the selected literal. If $I_m^P(\lambda_i) = \{u_1/x_1, \ldots, u_n/x_n\}$ or $I_c^P(\lambda_i) = \{u_1/x_1, \ldots, u_n/x_n\}$, then $I_g^P(\lambda_i) = \{u_1/x_1, \ldots, u_n/x_n\}$.*

The inference of the mode of certains predicate symbols with respect to a given program may be done bottom up. For this sake let P be a program and $\mathcal{P}$ be the set of all predicate symbols occurring in P. Assume the relation "depend on", denoted by $\sqsupseteq$, be defined on the set $\mathcal{P}$ (see Kunen in [7]). The relation $\sqsupseteq$ defines an equivalence relation on $\mathcal{P}$. We denote by $\sqsupset$ the transitive and irreflexive relation $\sqsupseteq$. Hence, $\mathcal{P} = \cup_{j=1}^{k} \mathcal{P}_j$ where $\mathcal{P}_j$ is an equivalence class.
Let us now state and prove an useful theorem of our method.

Theorem 6.1 *Let P be a program, the definition of $p \in \mathcal{P}$ consists k clauses, $p(\vec{t}) \leftarrow L_{i,1}, \cdots, L_{i,m(i)}$ be the i-th clause and $\sigma_i = a\text{-}mgu(p(\vec{v}), p(\vec{t}))$. Then $I_m^P(p) = lui(\{\vartheta_i \mid 1 \leq i \leq k\})$, where for each $1 \leq i \leq k$ it holds either $\vartheta_i = I_g^P(L_{i,1}\sigma_i) \cdots I_g^P(L_{i,j-1}\sigma_i) I_g^P(L_{i,j+1}\sigma_i) \cdots I_g^P(L_{i,m(i)}\sigma_i)$ if $\exists \rho \in \mathcal{R}$ such that $L_{i,j} \leq p(\vec{t})$ or $p(\vec{t}) \leq L_{i,j}$ and $vars(p(\vec{t})) = vars(L_{i,j}\rho)$ or $\vartheta_i = I_g^P(L_{i,1}\sigma_i) \cdots I_g^P(L_{i,m(i)}\sigma_i)$ otherwise.*

Proof Let us suppose that the definition of p consists of k clauses. Then from the definition 6.1 we have

$$\begin{aligned} I_m^P(p) &= \text{lui}(\{\theta' \mid \theta' \text{ is an acceptable mode for } P \cup \{\leftarrow p(\vec{v})\}\}) \\ &= \text{lui}(\{\theta_i \mid \theta_i \text{ is an acceptable mode for} \\ &\quad P \cup \{\leftarrow (L_{i,1}, \cdots, L_{i,m(i)})\sigma_i\} \text{ and } 1 \leq i \leq k\}) \end{aligned}$$

where $\sigma_i = \text{a-mgu}(p(\vec{v}), p(\vec{t}))$; $p(\vec{v})$ unifies with the head of each clause C_i, $1 \leq i \leq k$. It suffices to determine θ_i for each $i \in \{1, \ldots, k\}$. That is ϑ_i. Hence, let us consider a clause C_i for a given $1 \leq i \leq k$. The proof is by induction using the definition of the NJSLDT-derivation. Two cases arise.

`case 1:` $p(\vec{v}) \not\leq L_{i,j}$ and $L_{i,j} \not\leq p(\vec{t})$ or $\text{vars}(p(\vec{t}))) \neq \text{vars}(L_{i,j}\rho)$ hold for all variable renaming substitutions ρ and literal $L_{i,j}$ occurring in the body of the used clause. Since θ_i is an acceptable mode for $P_a \cup \{\leftarrow L_{i,1}\sigma_i, \cdots, L_{i,m(i)}\sigma_i\}$, θ_i is also an acceptable mode for $P_a \cup \{\leftarrow L_{i,j}\sigma_i\}$ for each $1 \leq j \leq m(i)$. That means $P_a \cup \{\leftarrow L_{i,j}\sigma_i\}$ has a derivation which ends with ϵ. Let now $L_{i,j}\sigma_i = q(\vec{s})$. Then it holds that $q(\vec{s}) = q(\vec{x}) I_c^P(q(\vec{s}))$. Then from definition 6.3 it suffices to determine $I_m^P(q)$ which is done by the induction hypothesis. Hence, the goal mode for $L_{i,j}\sigma_i$ is $\text{lui}(I_m^P(q), I_c^P(q(\vec{s})))$. Hence, $\vartheta_i = I_g^P(L_{i,1}\sigma_i) \cdots I_g^P(L_{i,m(i)}\sigma_i)$.

`case 2:` $p(\vec{v}) \leq L_{i,j}$ or $L_{i,j} \leq p(\vec{t})$ and $\text{vars}(p(\vec{t}))) = \text{vars}(L_{i,j}\rho)$ hold for some variable renaming substitution ρ and literal $L_{i,j}$ occurring in the body of the used clause. Since θ_i is an acceptable mode for $P_a \cup \{\leftarrow L_{i,1}\sigma_i, \cdots, L_{i,m(i)}\sigma_i\}$, θ_i is also an acceptable mode for $P_a \cup \{\leftarrow L_{i,j}\sigma_i\}$ for each $1 \leq j \leq m(i)$. Using Lemma 5.1 we can remove the literal, say $L_{i,j}\sigma_i$, such that $L_{i,j}\sigma_i = p(\vec{v})\rho$ for some variable renaming substitution ρ. Then by case 1 above we obtain the result. □

Corollary 6.1 *Let P be a program and p be an n-ary predicate symbol such that at most p occurs in the body of each clause κ with head predicate symbol p, $head(\kappa) \leq body(\kappa)$ or $body(\kappa) \leq head(\kappa)$ and $vars(head(\kappa))) = vars(body(\kappa)\rho)$ hold for some variable substitution ρ. Then $I_m^P(p) = lui(\{\, a\text{-}mgu(p(\vec{v}), head(C_i)) \mid 1 \leq i \leq k\})$ where C_i, $1 \leq i \leq k$, are all clauses of P which define the predicate p.*

6.1 Examples

Note that repeated variables do occur in the following examples. Let us now formally infer the mode of some n-ary predicate symbols with respect to a given program and use $C_i: 1 \leq i \leq n$ to refer to a clause. The selected literal, say λ, is underlined. The edge is marked by the used clause, say C, and the substitution if there does not exist $(L, C) \in S$ such that for all variable renaming substitutions ρ it holds that $\text{var}(\lambda) \neq \text{var}(L\rho)$ or $\lambda \not\leq L$ and $L \not\leq \lambda$. For the sake of simplicity the set S has been omitted; but it may be constructed following a path in the tree. Let us write u, u_i with $i \geq 0$ as a short hand for a closed term in the NJSLDT-trees.

Example 6.1 Every closed term is mapped to a symbolic constant denoted by u.

$$
\begin{array}{lll}
C_1: \texttt{person}(u_1) & \leftarrow & \\
C_2: \texttt{parent}(u_2, u_3) & \leftarrow & \\
C_3: \texttt{same_generation}(x, x) & \leftarrow & \texttt{person}(x) \\
C_4: \texttt{same_generation}(x, y) & \leftarrow & \texttt{parent}(x, xp), \texttt{same_generation}(xp, yp), \\
 & & \texttt{parent}(y, yp)
\end{array}
$$

Let us write `sg` as a short hand for `same_generation`, `p` for `parent` and `q` for `person` in the NJSLDT-tree (figure 1). Let further θ_i with $1 \leq i \leq 3$ be the composition of the substitutions on a path from the root to the leaf.

$$
\begin{array}{lll}
\theta_1 & = & \{v/w\} \circ \{u/v\} \\
 & = & \{u/v, u/w\} \\
\theta_2 & = & \{v/x, w/y\} \circ \{u_1/vp\} \circ \{u_2/v\} \circ \{u_3/w\} \quad = \{u_1/v, u_3/w\} \\
\theta_3 & = & \{v/x, w/y\} \circ \{u_4/v, u_4/vp\} \circ \{u_6/w, u_7/wp\} \quad = \{u_4/v, u_6/w\}
\end{array}
$$

Then the inferred mode of sg is $I_m^P(\text{sg}) = \text{lui}\{\theta_1, \theta_2, \theta_3\} = \{u'/v, u''/w\}$. $\diamondsuit$

Since the NJSLDT-derivation returns a most general substitution, since the existence of a fair NJSLDT-tree is guaranteed and since infinite loops are recognise and make finite the evaluation strategy does not influent a computed mode.

Example 6.2 Let us consider the following program `PP` and denote a closed term by u, u_1, u_2 etc.

$$
\begin{array}{lll}
C_1: p(h(x), g(x, y)) & \leftarrow & q(x, z), r(f(z), z), s(z, y) \\
C_2: q(x, u_1) & \leftarrow & \\
C_3: q(z, h(z)) & \leftarrow & r(z, y), q(f(y), z) \\
C_4: s(x, u_2) & \leftarrow & q(x, u_2) \\
C_5: r(u_3, u_4) & \leftarrow &
\end{array}
$$

The NJSLDT-tree (figure 2) illustrates the inference of the inferred mode of p with respect to the program PP. Let θ_i with $i \geq 0$ be the composition of the substitutions on a path from the root to the leaf. Then

$$I_m^P(p) = \text{lui}\{\theta_i | i \geq 0\} = \{h(x_1)/v, g(x_1, u)/w\}.$$

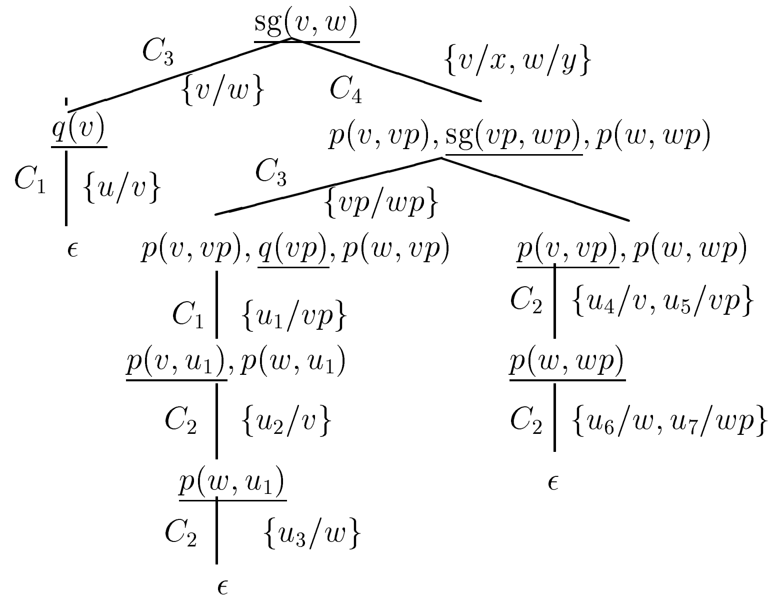


Figure 1: NJSLDT-tree for `same_generation`

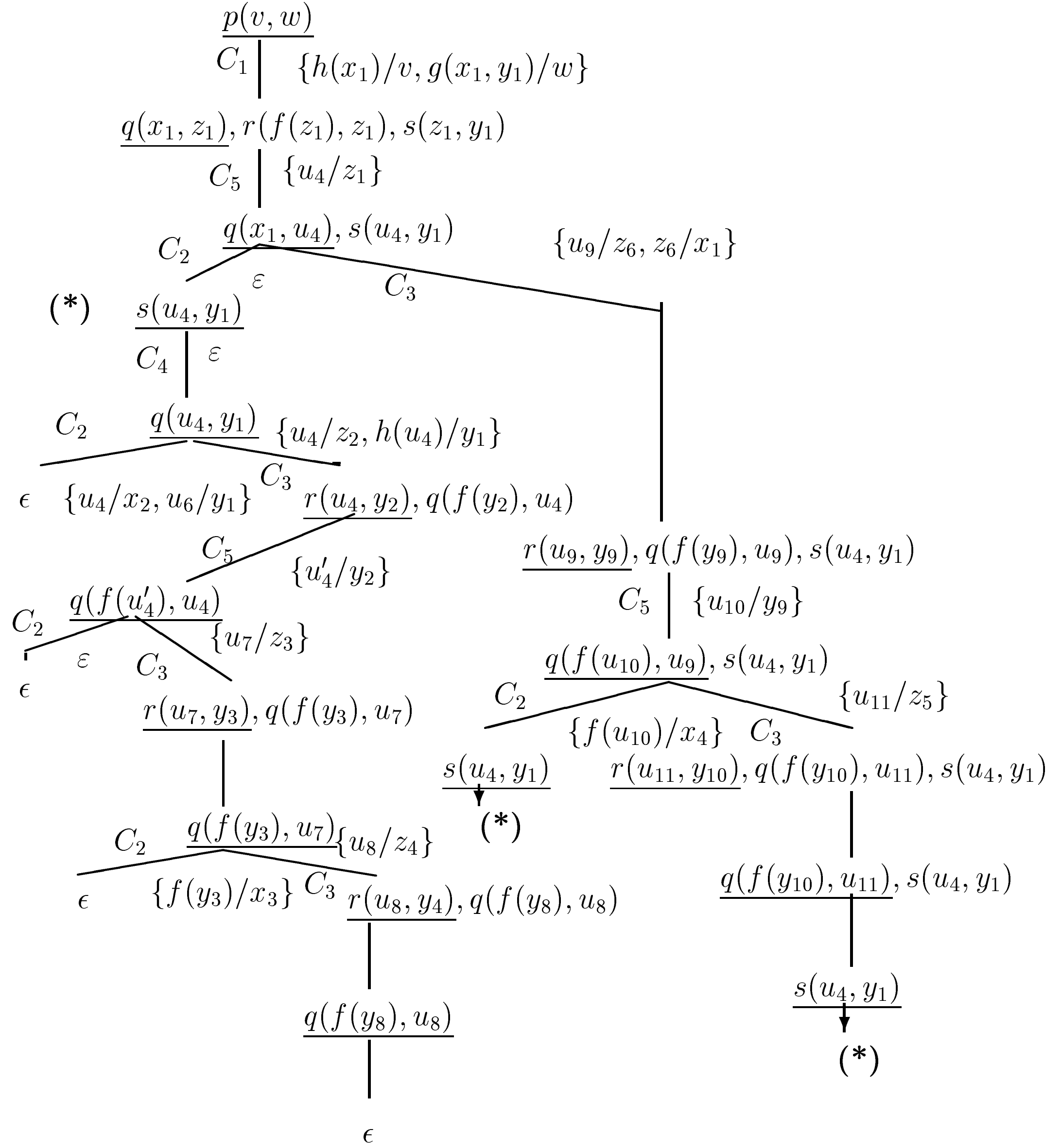


Figure 2: NJSLDT-tree for $\texttt{PP} \cup \{\leftarrow p(v, w)\}$

$\diamondsuit$

To illustrate theorem 6.1 and corollary 6.1 above let us consider the following example.

Example 6.3 Let `APPEND` be the program defining the predicate `append`. [] is mapped to u. The NJSLDT-tree (figure 3) illustrates the inference of the inferred mode of `append`.

$$
\begin{array}{lll}
C_1 : \texttt{reverse}(u_1, u_2) & \leftarrow & \\
C_2 : \texttt{reverse}([x|xs], [y|ys]) & \leftarrow & \texttt{reverse}(xs, zs), \texttt{append}(zs, [x], [y|ys]) \\
C_3 : \texttt{rotate}(u_1, u_2) & \leftarrow & \\
C_4 : \texttt{rotate}([y|ys], [x|xs]) & \leftarrow & \texttt{reverse}([y|ys], [x|zs]), \texttt{reverse}(xs, zs)
\end{array}
$$

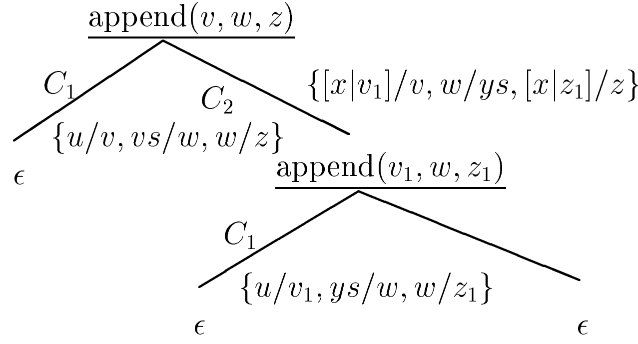


Figure 3: NJSLDT-tree for `append`

Let θ_i with $1 \leq i \leq 3$ be the composition of the substitutions on a path from the root to the leaf; that is

$$\begin{aligned}
\theta_1 &= \{u/v, vs/w, w/z\} \\
\theta_2 &= \{[x|v_1]/v, w/ys, [x|z_1]/z\} \circ \{u/v_1, ys/w, w/z_1\} \\
&= \{[x|u]/v, w/ys, [x|w]/z\} \\
\theta_3 &= \{[x|v_1]/v, ys/w, [x|z_1]/z\}
\end{aligned}$$

Then the inferred mode of sg is

$$I_m^P(\text{append}) = \text{lui}\{\theta_1, \theta_2, \theta_3\} = \{[x|v_1]/v, ys/w, [x|z_1]/z\}.$$

Applying theorem 6.1 we get

$$\begin{aligned}
I_m^P(\texttt{reverse}) &= \text{lui}(I(C_1), I(C_2)) \\
&= \text{lui}(\{u_1/v, u_2/w\}, (\{[x|xs]/v, [y|ys]/w\} I_g^P(\texttt{append}))\ ) \\
&= \{[x|xs]/v, [y|ys]/w\} I_g^P(\texttt{append}) \\
&= \{[x|xs]/v, [y|ys]/w\} \text{gli}(I_c^P(\texttt{append}), I_m^P(\texttt{append})) \\
&= \{[x|xs]/v, [y|ys]/w\}
\end{aligned}$$

The inferred mode of `rotate` will be inferred in a similar manner.

Suppose now a goal $\leftarrow$ `rotate`$([1,2,3], vs)$ be given. When using clause C_4 and the substitution $\{1/y_1, [2,3]/ys_1, [x_1|xs_1]/vs\}$ a part of the derivation structure is as follows:

$$\begin{array}{c}
\texttt{rotate}([1,2,3], vs) \\
\Big| \\
\texttt{reverse}([1,2,3], [x_1|zs_1]) \wedge \texttt{reverse}(xs_1, zs_1) \\
\vdots
\end{array}$$

The goal mode of the subgoal `reverse`$([1,2,3], [x_1|zs_1])$ is

$$I_g^P(\texttt{reverse}([1,2,3], [x_1|zs_1]) = \{u/v, [x|xs]/w\}.$$

After the resolution of this subgoal the argument zs_1 becomes closed; therefore the goal mode of $\texttt{reverse}(xs_1, zs_1)$ is

$$I_g^P(\texttt{reverse}(xs_1, zs_1) = \{x/v, u/w\}.$$

It is obvious that not every programmer shall think of these two different modes of `reverse` when writting such a program as this one above. ♢

Note that if we do not restrict the class of logic programs to those that are well-formed in the sense of section 3 then there still do exist definitions of n-ary predicate symbols for which our method does not find a mode. Let us consider

$$P = \{\ q(f(u_1, y)) \leftarrow \epsilon,\ q(g(s)) \leftarrow \epsilon\ \}$$

with $g \neq f$. Then $I_m^P(q) = \mathrm{lui}(f(u_1, y)/x_1, g(s)/x_2)$ has no lui.

7 Related works and conclusion

7.1 Related works

The idea of inferring the mode of an n-ary predicate symbol defined by a logic program is not new. Since mode has an influence on the operational semantics, certain languages for first order predicate logic have included annotations and static mode inference to guide the interpreter in selecting the literal in a goal. In this case the user is responsible for a correct annotation of mode and a consistent use of such a moded n-ary predicate symbol in a goal. A mode checking method may assist the user in this point. In [3, 4, 6, 9, 11, 22] the declaration of mode is automatically inferred.

Our work is distinct from these efforts in three significant points:

- First, we do not infer the set of all possible mode tuples an n-ary predicate symbol defined by a logic program may have. This is due to the fact that if there is k mode types in consideration then the total number of mode tuples is k^n for each n-ary predicate symbol. The mode tuple we do infer is most general in the sense of most general unifier and reflect the understanding of the relation defined by an n-ary predicate symbol.
- Second, our method is appropriate for modular logic programming, in the sense that an inferred mode of an n-ary predicate symbol may be used to deduce the mode of an n-ary predicate symbol which depends on it.
- Third, errors caused by user supplied mode for an n-ary predicate symbol do not occur, and there is no need for a mode checking.

7.2 Conclusion

An n-ary predicate describes the relationship between its arguments, in the sense that no argument has to be of a special mode. The unification and the resolution (SLDNF-resolution) do capture this state of affair. The declarative style of logic programming is an ease realisation of this fact of an n-ary predicate. Hence, the aim of logic programming, that is to write a program in a declarative style and to leave the control to the inplemented interpreter, is approximatively achieved if the system is able to automatically determine the descriptive mode of an n-ary predicate symbol with respect to a given program. Since the mode of an n-ary predicate symbol does influence the operational semantics of that predicate, this information may be used by a compiler for the purpose of efficiency. Our mode inference will also help

- automatically determine the literal to be selected when constructing an SLDNF-derivation.
- prove the completeness of the SLDNF-resolution for a class of logic programs, which contains the class of allowed programs. Work in this direction has been suggested by Kunen in [7]. A completeness proof of the SLDNF-resolution using prescriptive mode is given by Stärk in [20].

References

[1] K. R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 10, pages 495–574. Elsevier, 1990.

[2] M. Bruynooghe, B. Demoen, A. Callebaut, and G. Janssens. Abstract interpretation: towards the global optimization of prolog programs. In *Proceedings of the fourth IEEE Symposium on Logic Programming*, New York, 1987.

[3] M. Bruynooghe and G. Janssens. An instance of abstract interpretation integrating type and mode inferencing (extended abstract). In R. A. Kowalski and K. A. Bowen, editors, *Logic Programming – Proceedings of the fifth International Conference and Symposium*, pages 669–683. MIT Press, 1988.

[4] S. K. Debray. Flow analysis of dymanic logic programs. *Journal of Logic Programming*, 7:149–176, 1989.

[5] S. K. Debray. Static inference of modes and data dependencies in logic programs. *ACM Transactions on Programming Languages and Systems*, 11(3):418–450, 1989.

[6] S. K. Debray and D. S. Warren. Automatic mode inference for logic programs. *Journal of Logic Programming*, 5:207–229, 1988.

[7] K. Kunen. Signed data dependencies in logic programs. *Journal of Logic Programming*, 7:231–245, 1989.

[8] J. W. Lloyd. *Foundations of Logic Programming*. Springer, 1987.

[9] H. Mannila and E. Ukkonen. Flow analysis of prolog programs. In *Proceedings of the fourth IEEE Symposium on Logic Programming*, New York, 1987.

[10] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

[11] C. S. Mellish. The automatic generation of mode declaration for prolog programs. DAI research 163, Department of Artificial Intelligence, University of Edinburgh, 1981.

[12] C. S. Mellish. Some global optimization for a prolog compiler. *Journal of Logic Programming*, 2(1):43–66, 1985.

[13] C. S. Mellish. Abstract interpretation of prolog programs. *Lecture Notes in Computer Science*, 225:463–474, 1986.

[14] L. Naish. Automating control for logic programs. *Journal of Logic Programming*, 3:167–183, 1985.

[15] G. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.

[16] U. S. Reddy. On the relationship between logic and functional languages. In D. De Groot and G. Lindstrom, editors, *Logic Programming*, pages 3–35. Prentice-Hall, 1986.

[17] J. Reynolds. Transformational systems and algebraic structure of atomic formulas. *Machine Intelligence*, 5:135–152, 1970.

[18] H. Schwichtenberg. Logikprogrammierung. Vorlesungsausarbeitung, Ludwig-Maximilians-Universität München, Wintersemester, 1993.

[19] W. Snyder and J. Gallier. Higher-order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, 8:101–140, 1989.

[20] R. F. Stärk. Input/output dependencies of normal logic programs. *Journal of Logic and Computation*, 4(3):249–262, 1994.

[21] K. Stroetmann. A completeness result for sldnf–resolution. *Journal of Logic Programming*, 15(4):337–355, 1993.

[22] D. H. D. Warren. Implementing prolog – compiling predicate logic programs. Research Report 39,40, Department of Artificial Intelligence, University of Edinburgh, 1977.