논리 프로그램의 타입 및 모드 분석의 계층 구조

(A Hierarchical Structure of Type and Mode Analyses of Logic Programs)

창 병 모 †

(Byeong-Mo Chang)

요 약 논리 프로그램의 타입 및 모드 분석은 실행 시간에 변수가 갖게 되는 텀의 집합에 대한 근사값을 구하는데 보통 요약 해석을 기반으로 하여 설계되고 개발되어 왔다. 이 논문에서는 타입 및 모드 분석을 위해서 설계된 다양한 요약 도메인을 고려하여 그들간의 계층 관계를 Galois insertion을 기준으로 밝힌다. 이 논문에서는 요약 도메인으로써 타입 그래프, 깊이-k 타입, 깊이-k 모드, 순환 모드, 그리고 모드를 고려할 것이다. 특히 깊이-k 모드는 타입 및 모드 분석을 통합하기 위한 도메인으로써 깊이-k 타입을 확장하여 제안하였다.

Abstract Type and mode analyses of (constraint) logic programs approximates a set of terms, with which arguments can be instantiated during execution. They have been designed and developed usually based on abstract interpretation. This paper considers various abstract domains designed for type and mode analysis, and to show a hierarchy of them in terms of Galois insertion. We are going to take the following abstract domains into consideration : type graph, depth-k type, depth-k mode, recursive mode, and mode. Depth-k mode is proposed as a new abstract domain for an integration of type and mode analysis, by extending depth-k type of Sato and Tamaki's.

1. Introduction

Type and mode analysis of (constraint) logic programs is of primary importance for high-performance compilers, since type information may lead to better indexing and to sophisticated specializations of unification and built-in predicates. For this reason, many type and mode analyses have been designed and developed [1,2,3,4], usually by considering optimization effects. They are usually based on abstract interpretation [5,6], a semantics-based global analysis framework by Cousot and Cousot.

Type analysis of logic programs find, for every program point (usually predicate argument) of interest, the type(a set of terms) which it can be instantiated with during execution. Similarly, the mode of an argument is information on whether a particular argument of a predicate is instantiated on input or on output or both. As well known, it is a characteristic of logic programs that an argument of a predicate can be used as input, output, or both. An abstract domain designed for type and mode analysis is usually a lattice or poset representing all possible type information. A mode and type analysis based on abstract interpretation computes the least fixpoint of the program's execution over an abstract domain, which is a conservative approximation to the concrete

· 이 논문은 STEPI 과제 초고속전산2G-12 "초고속 컴퓨터 병렬언어 및 과학계산용 프로그램 개발 환경"의 지원에 의한 것임

† 종신회원 : 숙명여자대학교 전산학과 교수
chang@cs sookmyung ac kr

논문접수 : 1998년 8월 17일
심사완료 1999년 1월 20일

execution. The information gathered through type analysis can be applied to compile-time optimization, debugging, and so on.

Getzinger showed lattices of mode domains and type domains *respectively*, which shows how one abstract domain approximates another [7]. The motivation of this paper is to consider various abstract domains for type and mode analysis *together*. and to show a hierarchy of them in terms of Galois insertion. We are going to take the following popular and new abstract domains into consideration : type graph [1], depth-k type [3], depth-k mode, recursive mode [4], and mode [2]. We also propose a new abstract domain called *depth-k mode* for an integration of type and mode analysis, by extending depth-k type of Sato and Tamaki's [3]. There are also many analyses of logic programs like alias analysis, and data-dependency analysis [7,8,9], which are based on abstract interpretation and can be applied to AND parallel execution. This paper, however, focuses on type and mode analysis, because type and mode analyses can be hardly compared with other analyses like data-dependency analysis, in the sense that they are interested in quite different properties.

This paper is organized as follows: Section 2 describes some basic definitions on logic programs and abstract interpretation. Section 3 reviews the abstract domains we are going to consider. In Section 4, we show a hierarchy of the abstract domains. Section 5 concludes this paper

2. Basic definitions

We let $\wp(S)$ denote the power set of a set S. When S is a set and $\sim$ is an equivalence relation on S, $S/\sim$ is the set of equivalence classes on S with respect to $\sim$. For an element $a \in S$, $[a]_\sim$ denotes the equivalence class of a with respect to $\sim$. We will assume familiarity with the basic notions of lattice theory. A set $\sim$ partially ordered by $\sqsubseteq$ is denoted $(S, \sqsubseteq)$. We sometimes abbreviate a poset by its set of objects. If S is a poset, we usually denote $\sqsubseteq_S$ the corresponding partial order. A complete lattice A with partial ordering $\sqsubseteq$, least upper bound *lub* $\sqcup$ greatest lower bound *glb* $\sqcap$, least element $\bot = \sqcup\emptyset = \sqcap A$, and greatest element $\top = \sqcap\emptyset = \sqcup A$, is denoted by $\langle A, \sqsubseteq, \sqcup, \sqcap, \top, \bot\rangle$. When A is a lattice, $\sqsubseteq_A, \sqcup_A, \sqcap_A, \top_A$ and $\bot_A$ denote the corresponding basic operators and elements. If A is a poset, the least fixpoint is denoted by $lfp(f)$, if it exists. It is well known that, when f is a continuous function, then $lfp(f) = \sqcup_{k<\omega} f^k(\bot_A)$. Let (Π, Σ, Var) denote a first order language, namely a (finite) set Π of predicate symbols, a set Σ of function symbols, and a denumerable set of variables Var. With each function symbol $f \in \Sigma$ and predicate symbol $p \in \Pi$ is associated a unique natural number called its *arity*: a (predicate or function) symbol f with arity n is written f/n. The set of all terms constructed from Σ and Var is denoted by $Term(\Sigma, Var)$ (or $Term$ for short). An *atom* is a syntactic object of the form $p(t_1, \ldots, t_n)$ where $p/n \in \Pi$ and $t_1, \ldots, t_n \in Term(\Sigma, Var)$. We denote $Atoms$ the set of all atoms constructed from Π and $Term(\Sigma, Var)$. We also associate a tuple $\langle t_1, \ldots, t_n\rangle$ with each atom $p(t_1, \ldots, t_n)$. A *logic program* P is a finite set of *program clauses* C of the form $A \leftarrow B_1, \ldots, B_n (n \geq 0)$ where A is called a *head* atom and $B_1, \ldots, B_n$ are *body* atoms. A it *query* rm is a sequence of atoms, sometimes denoted $\leftarrow B_1, \ldots, B_n$. We also define a function gr which maps any set of syntactic objects into the corresponding set of their ground instances. A *substitution* is a mapping from Var to $Term$ which acts as the identity almost everywhere. It extends to apply to any syntactic objects in the usual way. The identity substitution is denoted ε. The set of idempotent substitutions is denoted by Sub.

An abstract interpretation of a program is an approximation of its collecting concrete semantics on an abstract domain [5]. In the following we consider the standard framework of Cousot and Cousot [5]. This framework presupposes a least

fixpoint characterization of the collecting semantics. The overall abstract interpretation methodology can be summarized as follows: (1) a concrete (collecting) semantics is selected which correctly characterizes the observable property of interest; (2) an abstract semantics is defined by providing approximated descriptions of semantic objects and operators. We assume that a concrete interpretation for a (logic) program P can be defined in terms of a monotone operator $E_P: E \to E$ on a oncrete domain E. An abstract interpretation $((E, \sqsubseteq), E_P, (D, \leq), D_P, \alpha, \gamma)$ consists of a concrete domain (a complete lattice) $(E, \sqsubseteq)$, a monotone operator $E_P: E \to E$, an abstract domain(a complete lattice) $(D, \leq)$, a monotone operator $D_P: D \to D$, an *abstraction function* $\alpha: E \to D$ and a *concretization function* $\gamma: D \to E$, such that

(1) α and γ are monotone,
(2) $d = \alpha(\gamma(d))$ for all $d \in D$,
(3) $e \sqsubseteq \gamma(\alpha(e))$ for all $e \in E$, and
(4) $E_P(\gamma(d)) \sqsubseteq \gamma(D_P(d))$ for all $d \in D$ (or $\alpha(E_P(e)) \leq D_P(\alpha(e))$) for all $e \in E$).

If the conditions (1) --- (3) are satisfied, then $((E, \sqsubseteq), \alpha, (D, \leq), \gamma)$ is said to be a (it *Galois insertion*. Condition (4) is the it correctness rm criterion that ensures that D_P faithfully mimics E_P, i.e. whenever $((E, \sqsubseteq), E_P, (D, \leq), D_P, \alpha, \gamma)$ is an abstract interpretation, then $lfp(E_P) \sqsubseteq \gamma(lfp(D_P))$.

3. Abstract domains

We first assume that type and mode analyses are based on an abstract interpretation framework as in [1], even though the result of this paper need not be restricted to the framework. Following [3], we take a view of a type as an abstraction of a set of terms. Given a term t and type T, we write $t:T$, read "t has type T", if $t \in T$. If $(\wp(Term), \subseteq), \alpha, (\boldsymbol{T}, \sqsubseteq), \gamma)$ is a Galois insertion, $(\boldsymbol{T}, \sqsubseteq)$ is a domain of types. We define an ordering relation among types such that given any two types T_1 and T_2: $T_1 \leq T_2$ iff $\gamma(T_1) \subseteq \gamma(T_2)$. Let $(\wp(Term), \subseteq), \alpha, (\boldsymbol{T}, \sqsubseteq), \gamma)$ be a Galois insertion. Then, starting from the Galois insertion, we can induce, by extending the abstraction function to substitution, a domain of abstract substitutions $(ASub_T, \sqsubseteq)$ such that $(ASub_T, \sqsubseteq)$ is a set of abstract substitutions from Var to $\boldsymbol{T}$ and $(\wp(Sub), \subseteq), \alpha, (ASub_T, \sqsubseteq), \gamma)$ is a Galois insertion. The abstraction function α maps a set of substitutions Θ over a set of variables S_X into an abstract substitution over S_X, such that $\alpha(\Theta) = \lambda_{x \in S_X} \cdot \sqcup \{\alpha(\theta(x)) | \theta \in \Theta\}$. The concretization function γ maps an abstract substitution into a set of substitutions such that $\gamma(\beta) = \sqcup \{\theta | \alpha(\theta) \sqsubseteq \beta\}$. A (concrete) substitution θ is said to *satisfy* an abstract substitution β, denoted by $\theta \in \beta$, iff for all $x \in S_X, \theta(x) \in \gamma(\beta(x))$ where θ is a substitution over a set of variables S_X. The concrete collecting semantics is a collecting fixpoint semantics which captures the top-down execution of logic programs using a left-to-right computation rule. The semantics manipulates a set of substitutions $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$. The semantics associates to each of the predicate symbol p in the program a set of tuples of the form $(\Theta_{in}, p, \Theta_{out})$ which can be interpreted as follows:

"the execution of $p(x_1, \ldots, x_n)\theta$ with $\theta \in \Theta_{in}$ produces a sequence $\theta_1, \ldots, \theta_n, \ldots$ of substitutions, all of which belongs to Θ_{out}".

Abstract semantics consists in abstracting a set of substitutions by a single abstract substitution, i.e. an abstract substitution represents a set of substitutions. As a consequence, the abstract semantics associates with each predicate symbol p a set of tuples of the form $(\beta_{in}, p, \beta_{out})$ which can be interpreted as follows:

"the execution of $p(x_1, \ldots, x_n)\theta$ with theta satisfying the property described by β_{in} produces a sequence $\theta_1, \ldots, \theta_n, \ldots$ of substitutions, all of which satisfying the property described by β_{out}."

We are going to review abstract domains introduced for type and mode analysis of logic programs.

Mode : Debray classified the set of all runtime terms into the set of modes [2]:
$\Delta = \{any, var, nv, gnd, \emptyset\}$ where *any* denotes the set of all run runtime terms, *var* denotes the set of free variables, *nv* denotes the set of non-variable terms, and *gna* denotes the set of ground terms. The set Δ forms a complete lattice under inclusion as in Figure 1. The ordering on Δ induced by inclusion is denoted by $\sqsubseteq$ and the corresponding join operation is denoted by $\sqcup$. Let $\alpha: \wp(Term) \to \Delta$ and $\gamma: \Delta \to \wp(Term)$ be the abstraction function and the concretization function defined in [2]. Then, $((\wp(Term), \subseteq), \alpha, (\Delta, \sqsubseteq), \gamma)$ is a Galois insertion.

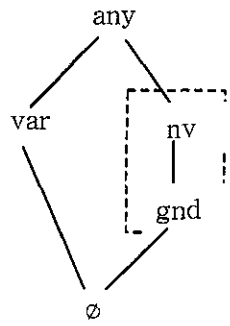

Fig. 1 The lattice of modes

Recursive mode Tan and Lin extended Debray's mode by classifying ground terms and nonvariable terms further, following the observation that *gna* and *nv* is too simple and the analyses based on the mode cannot infer the incomplete data structures accurately [4]. Their modes are defined by the following grammar.

$nv \to f(any, any^*) \mid gna$
$\lg \to f(\lg, any^*) \mid gna$
$rg \to f(any^*, rg) \mid gna$
$lrg \to f(lrg, any^*, lrg) \mid gna$
$gnd \to f(gnd, gnd^*) \mid const$

In the grammar, f can be any function symbol, *const* can be any constants, and *any* can be any terms. We denote the recursive mode by $RM = \{any, var, nv, \lg, rg, \lg \cap rg, lrg, gnd, free, \emptyset\}$.

Let $\alpha: \wp(Term) \to RM$ and $\gamma: RM \to \wp(Term)$ be the abstraction function and the concretization function defined in [4]. Then, $((\wp(Term), \subseteq), \alpha, (RM, \sqsubseteq), \gamma)$ is a Galois insertion. The ordering on RM induced by inclusion is denoted by $\sqsubseteq$ and the corresponding join operation is denoted by $\sqcup$.

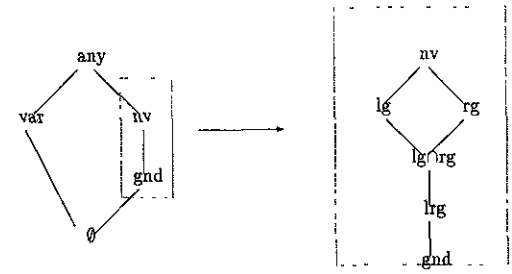

Fig. 2 The lattice of recursive modes

Depth-k type : Sato and Tamaki proposed the depth-k abstraction, in which the depth-k abstract term of a term is obtained by substituting every level k subterm of t with a special symbol *any* [1], which means any terms. In this paper, we assume the canonical depth-k abstraction in [10], in which no variable can occur more than once. Let $\rho(t)$ be a function which maps a term t to its canonical depth k abstract term. The set of all depth-k abstract terms constructed from Σ and *Var* is denoted by $Term_{D,T}$. Then, $((\wp(Term), \subseteq), \alpha, (\wp(Term_{D,T}), \sqsubseteq), \gamma)$ is a Galois insertion, where the abstraction function is $\alpha(T) = \{\rho(t) \mid t \in T\}$, and the corresponding concretization function is $\gamma(T^A) = \sqcup\{T \mid \alpha(T) \sqsubseteq T^A\}$. The ordering $\sqsubseteq$ on $Term_{D,T}$ is defined as : $T_1 \sqsubseteq T_2$ if $\gamma(T_1) \subseteq \gamma(T_2)$, and the corresponding join operation is denoted by $\sqcup$.

Depth-k mode : We propose a depth-k mode domain for an integration of type and mode analysis by adding mode information into the canonical depth-k type. We extend $\rho(t)$ of the canonical depth-k type so that it map a term t to its depth k abstract mode, which is obtained by substituting every level k subterm of t with its mode (that is, $\alpha_\Delta(t)$ if α_Δ is the abstraction

1) a fresh variable in the original definition

function for the mode Δ)

The set of all depth-k abstract modes constructed from Σ, Var, and Δ is denoted by $Term_{D_kM}$. It is easy to find that if $k=1$, then $Term_{D_1M}=M$. Then,

$((\wp(Term),\subseteq),\alpha,(\wp(Term_{D_kM}),\sqsubseteq),\gamma)$ is a Galois insertion, where the abstraction function is $\alpha(T)=\{\rho(t)|t\in T\}$ and the corresponding concretization function is $\gamma(T^A)=\bigsqcup\{T|\alpha(T)=T^A\}$. We can prove the Galois insertion by showing that $\alpha(\gamma(T^A))=T^A$. It is proved as follows :

$$\alpha(\bigsqcup\{T_i|\alpha(T_i)=T^A\})=\alpha(T_1\sqcup\ldots\sqcup T_n)$$
$$=\alpha(T_1)\sqcup\ldots\sqcup\alpha(T_n)=T^A\sqcup\ldots\sqcup T^A=T^A$$

The ordering on $Term_{D_kM}$ is defined as : $T_1\sqsubseteq T_2$ if $\gamma(T_1)\subseteq\gamma(T_2)$, and the corresponding join operation is denoted by $\sqcup$.

Type graph : The type graph was proposed for an integrated analysis of type and mode [1,2]. A type graph is a directed graph with the following kinds of nodes: functor nodes, or nodes, simple nodes. A simple node has a label representing a predefined type such as any, var, gna. For simplicity of presentation, we assume that the primitive types such as Int, Real, .. are represented as gna. In this paper, we will consider normalized type graphs, which are constructed by applying the principal functor restriction to type graphs as in [1,2][2]. Let TG be the set of all normalized type graphs constructed from Σ and Δ. The abstraction $\alpha:\wp(Term)\rightarrow TG$ can be defined as in [1,2]. The meaning(concretization) $\gamma(G)$ of a type graph $G=(V,E)$ with its root r is determined as follows:

$\gamma(\langle G,r\rangle)=lfp(D)(r)$ where

$D:(V\rightarrow\wp(Term))\rightarrow(V\rightarrow\wp(Term))$ is

$D(\Phi)=\{(v,T)|v\in V, T=Denot(v,\Phi)\}$ where

$Denot(v,\Phi)=$

$$\begin{cases} Term & \text{if } type(v)=any \\ Var & \text{if } type(v)=free \\ gr(Term) & \text{if } type(v)=gnd \\ \bigcup_{1\le i\le n}\Phi(v_i) & \text{if } type(v)=or \\ \{f(t_1,\ldots,t_n)|f=functor(v), t_i\in\Phi(v_i)\} & \text{if } type(v)=f \end{cases}$$

In the definition, we assume that a node v has its child nodes $v_1,\ldots,v_n$. The ordering on TG is defined as : $G_1\sqsubseteq G_2$ if $\gamma(G_1)\subseteq\gamma(G_2)$, and the corresponding join operation is denoted by $\sqcup$. $G_1\equiv G_2$ iff $G_1\sqsubseteq G_2$ and $G_2\sqsubseteq G_1$. Then, $TG/_{\equiv}$ is a partial order, and $((\wp(Term),\subseteq),\alpha,(TG/_{\equiv},\sqsubseteq),\gamma)$ is a Galois insertion.

4. A hierarchy of the abstract domains

We now investigate Galois insertions between abstract domains and show a hierarchy of the abstract domains for type and mode analysis.

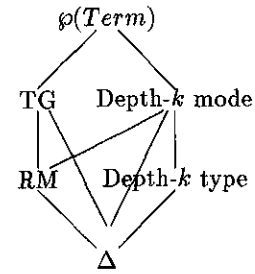

Fig. 3 A hierarchy of abstract domains

Abstraction from the recursive mode to the mode : We consider a Galois insertion $(RM,\alpha,\Delta,\gamma)$ between the recursive mode and the mode. The abstraction function α maps modes such as $\lg, rg, \lg\cap rg$, and lrg into nv, and maps as an identity for other modes.

Abstraction from the depth-k type to the mode : We consider a Galois insertion between the depth-k type and the mode: $(\wp(Term_{D_kT}),\alpha,\Delta,\gamma)$. The abstraction function, $\alpha:\wp(Term_{D_kT})\rightarrow\Delta$, is defined as

$\alpha(T)=\bigsqcup\{\alpha(t)|t\in T\}$ where

$\alpha(t)=$

2) Restricted type graphs are constructed by applying the principal functor restriction and the depth restriction to type graphs, so that type graphs are finite and only a finite number of restricted type graphs can be constructed from a finite set of functors

$$\begin{cases} gnd & \text{if } t \text{ is ground} \\ var & \text{if } t \text{ is a free variable} \\ nv & \text{if } t \text{ is a nonground term of the form } f(t_1,\ldots,t_n) \\ any & \text{if } t \text{ is a special symbol any} \end{cases}$$

Abstraction from the depth-k mode to the recursive mode : We now consider a Galois insertion from the depth-k mode to the recursive mode : $(p(Term_{D,M}), \alpha, RM, \gamma)$. The abstraction of a depth-k abstract mode, $\alpha(t): Term_{D,M} \to RM$, is defined as the least mode m such that t is an instance of the mode m. The abstraction function extended to a set of terms is $\alpha(T) = \sqcup\{\alpha(t) | t \in T\}$.

Abstraction from the depth-k mode to depth-k type : We now consider a Galois insertion from the depth-k mode to the depth-k type. The abstraction function $\alpha: p(Term_{D,M}) \to p(Term_{D,T})$ maps every occurrence of $gnd, free, nv$, and any in depth-k mode terms to any.

Abstraction from depth-k mode to depth-k′ mode : We can also consider a Galois insertion from the depth-k mode to the depth-k′ mode, if $k \geq k'$. The abstraction $\alpha: p(Term_{D,M}) \to p(Term_{D,M})$ maps the depth-k mode into the more abstract depth-k' mode.

Abstraction from the type graph to the recursive mode: We now consider a Galois insertion from the type graph to the recursive mode. The abstraction from normalized type graphs to the recursive mode is defined by the mode of the root node after labeling type graphs with recursive modes. We describe the mode labeling algorithm by means of a labeling transformer D^M. Let $G=(V,E)$ be a normalized type graph. A mode labeling is defined as a function $M: V \to RM$. A mode labeling ransformer, $D^M: (V \to RM) \to (V \to RM)$ updates mode labels of nodes using the mode labels of their child nodes.

Definition 1 A mode labeling transformer, $D^M: (V \to RM) \to (V \to RM)$ is defined as:

$D^M(M) = M'$ where

1. if v is a or node, $M'(v) = \sqcup_{v_i \in child(v)} M(v_i)$
2. if v is a functor node, $M'(v) =$

$$\begin{cases} gnd & \text{if } M(v_1) = gnd, M(v_2) = gnd, \ldots, M(v_n) = gnd \\ lrg & \text{if } M(v_1) = lrg_{\sqsubseteq} \text{ and } M(v_n) = lrg_{\sqsubseteq} \\ lg \sqcap rg & \text{if } M(v_1) = \lg_{\sqsubseteq} \text{ and } M(v_n) = rg_{\sqsubseteq} \\ lg & \text{if } M(v_1) = \lg_{\sqsubseteq} \\ rg & \text{if } M(v_n) = rg_{\sqsubseteq} \\ nv & \text{otherwise} \end{cases}$$

where $x_{\sqsubseteq}$ represents anyone of the modes less than or equal to x in the lattice of recursive modes in Figure 2.

Let M_0 be an initial mode labeling such that

$$M_0 = \begin{cases} \emptyset & \text{if } v \text{ is or node} \\ \emptyset & \text{if } v \text{ is functor node} \\ free & \text{if } v \text{ is free node} \\ gnd & \text{if } v \text{ is gnd node} \\ any & \text{if } v \text{ is any node} \end{cases}$$

The mode labeling algorithm computes the least fixpoint $lfp(D^M)$ in finite time, which is computed by $(D^M)^n(M_0)$ for some finite n.

Theorem 1 $lfp(D^M) = (D^M)^n(M_0)$ for some finite n

Proof. The least fixpoint computation is finite, since the transformer is monotone and there are a finite number of mode labelings in a type graph.

In formal setting, we have defined an abstract interpretation $((V \to p(Term)), D, (V \to RM), D^M, \alpha, \gamma)$. It is easy to show that $(V \to p(Term), \alpha, (V \to RM), \gamma)$ is a Galois insertion, where the abstraction function $\alpha: (V \to p(Term)) \to (V \to RM)$ is defined by pointwise lifting the abstraction function of the recursive mode, and the corresponding concretization function can also be defined in the same way. The correctness of the algorithm is shown in the theorem.

Theorem 2 *If* D^M is the model labeling transformer in Definition 1, then $\gamma(lfp(D^M)) \supseteq lfp(D)$.

Proof The theorem means that for all v, $\gamma(lfp(D^M)(v)) \supseteq lfp(D)(v)$. Assume that a node v has its child nodes $v_1, \ldots, v_n$. We show the theorem by showing that for all v, $\gamma(D^M(M)(v)) \supseteq D(\gamma(M))(v)$. Let M' be $D^M(M)$.

1. $type(v) = any, nv, free, gna$, it is obvious.
2. $type(v) = or$,
   $\gamma(M'(v)) = \gamma(\sqcup_{1 \leq i \leq n} M(v_i)) \supseteq \cup_{1 \leq i \leq n} \Phi(v_i)$, where $\Phi = \gamma(M)$. This is proved by the definition of $\sqcup$: $\gamma(a \sqcup b) \supseteq \gamma(a) \cup \gamma(b)$.

3. $type(v)=f$,
$\gamma(M'(v)) \supseteq \{f(t_1,\ldots,t_n) | t_i \in \Phi(v_i), 1 \le i \le n\}$, where $\Phi=\gamma(M)$. This is proved by showing that $\gamma(a \sqcup b) \supseteq \{f(t_1,t_2) | t_1 \in \Phi(a), t_2 \in \Phi(b)\}$. This can be shown for all the cases of $a \in RM$ and $b \in RM$.

Abstraction from the type graph to the mode : Abstraction from the type graph to the mode can be defined in two ways. The first is done through the abstraction from the type graph to the recursive mode, and the abstraction from the recursive mode to the mode. The second is to consider a Galois insertion from the type graph to the mode directly by slightly modifying the abstraction from the type graph to the recursive mode.

Let $G=(V,E)$ be a normalized type graph. A mode labeling is defined as a function $M: V \to \Delta$. The mode labeling algorithm is described by means of a mode labeling transformer, $D^M: (V \to \Delta) \to (V \to \Delta)$, which is defined as :

$$D^M(M)=M' \text{ where}$$

1. if v is a or node, $M'(v) = \bigsqcup_{v_i \in child(v)} M(v_i)$

2 if v is a functor node, $M'(v) =$

$$\begin{cases} gnd & \text{if } M(v_1)=gnd,\ M(v_2)=gnd,\ \ldots,\ M(v_n)=gnd \\ nv & \text{otherwise} \end{cases}$$

The labeling algorithm computes the least fixpoint, $lfp(D^M)$, which is computed by $(D^M)^n(M_0)$ for some finite n. The computation is finite, since the the labeling transformer is monotone and there are a finite number of mode labelings on a type graph.

Theorem 3 *If D^M is the model labeling transformer defined for the mode Δ, then $\gamma(lfp(D^M)) \supseteq lfp(D)$.*

Proof Assume that a node v has its child nodes $v_1,\ldots,v_k$.

We show the theorem by showing that for all v, $\gamma(D^M(M)(v)) \supseteq D(\gamma(M))(v)$. Let M' be $D^M(M)$.

1. $type(v)=any, nv, free, gnd$, it is obvious.
2. $type(v)=or$,
$\gamma(M'(v)) = \gamma(\bigsqcup_{1 \le i \le} M(v_i)) \supseteq \bigcup_{1 \le i \le n} \Phi(v_i)$, where $\Phi=\gamma(M)$. This is proved by the definition of $\sqcup$.
3. $type(v)=f$,
$\gamma(M'(v)) \supseteq \{f(t_1,\ldots,t_n) | t_i \in \Phi(v_i), 1 \le i \le n\}$, where $\Phi=\gamma(M)$. This is proved by showing that $\gamma(a \sqcup b) \supseteq \{f(t_1,t_2) | t_1 \in \Phi(a), t_2 \in \Phi(b)\}$. This can be shown for all the cases of $a \in \Delta$ and $b \in \Delta$.

Hierarchy of domains for abstract substitutions : Let $((T_1,\sqsubseteq_1),\alpha,(T_2,\sqsubseteq_2),\gamma)$ be a Galois insertion from a type domain T_1 to another type domain T_2. Then we can show that, in general, there is a Galois insertion from a domain $ASub_1$ of *abstract substitutions induced from* T_1 *to another* domain $ASub_2$ of abstract substitutions induced from T_2. We define domains of abstract substitutions $(ASub_1,\sqsubseteq_1)$ and $(ASub_2,\sqsubseteq_2)$ by extending $(T_1,\sqsubseteq_1)$ and $(T_2,\sqsubseteq_2)$ to substitutions as follows: $ASub_1 = Var \to T_1$ and $Asub_2 = Var \to T_2$ It is also possible to define $ASub = \wp(Var \to T)$. Given a Galois insertion $(T_1,\sqsubseteq_1),\alpha,(T_2,\sqsubseteq_2),\gamma)$, we can easily show that $((Asub_1,\sqsubseteq_1),\alpha,(Asub_2,\sqsubseteq_2),\gamma)$ is a Galois insertion by extending the abstraction function to substitutions. The abstraction function alpha maps an (abstract) substitution β over a set of variables S_X into an abstract substitution over S_X, such that $\alpha(\beta)=\lambda_{x \in S_X} \cdot \alpha(\beta(x))$. The corresponding concretization function is $\gamma(\beta^A)=\sqcup_1 \{\beta | \alpha(\beta) \sqsubseteq_2 \beta^A\}$

5. Conclusion

We have shown a hierarchy of abstract domains for type and mode analysis in terms of Galois insertion. In particular, as far as we know, the mode labeling algorithm is very new and formal. We have also proposed an abstract domain depth-k *mode for an integration of type and mode analysis*. It is not an approximation of type graph as shown in the hierarchy, even though it is simpler to maintain than type graphs, and it cann't represent recursive data structures as type graphs. There are also many abstract domains introduced for alias and data-dependency analysis of logic programs

[7,8,9]. There would be a hierarchical structure among the abstract domains, and it is also an interesting research topic to exploit the structure.

References

[1] M. Bruynooghe and G. Janssens. An Instance of Abstract Interpretation Integrating Type and Mode Inferencing. *Proc. of 5th Int'l Conference on Logic Programming*} 1988.

[2] M. Bruynooghe and G. Janssens. Deriving descriptions of possible values of program variables by means of abstract interpretation. *Proc. of 5th Int'l Conference on Logic Programming*} 1988.

[2] S. K. Debray. Static Inference of Modes and Data Dependencies in Logic Programs, *ACM Trans. on Programming Languages and Systems, Vol. 11*, No. 3, July, 1989, pp. 418-450.

[3] C. Cardelli and P. Wegner. On understanding types, data abstraction and polymorphism. *ACM Computing Surveys, Vol. 17*(4), 1985, pp. 471-522.

[3] T. Sato and H. Tamaki, Enumeration of success patterns in logic programs, *Theoretical Computer Science 34*, pp. 227-240, 1984.

[4] J. Tan and I-P. Lin. Recursive Modes for Precise Analysis of Logic Programs, *Proc. of 1997 Int'l Conference on Logic Programming*, 1997.

[5] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, *Proc. of the 4th ACM Symp. on Principles of Programming Languages*}, Jan. 1977, pp. 238-252.

[6] P. Cousot and R. Cousot. Abstract Interpretation and Application to Logic Programs. *Journal of Logic Programming*}, Special Issue on Abstract Interpretation, 1992.

[7] T. W. Getzinger. The Costs and Benefits of Abstract Interpretation-Driven Prolog Optimization. *Proc. of 1994 Int'l Static Analysis Symposium*} 1994.

[8] D. Jacobs and A. Langen, Accurate and Efficient Approximation of Variable Aliasing in Logic Programs. *Proc. of North American Conference on Logic Programming*, 1989.

[9] H. Xia and W. Gilloi, A New Application of Abstract Interpretation in Prolog Programs: Data-dependency analysis: *Proc. of IFIP WG 10.0 Workshop on Concepts and Characteristics of Declarative Systems*, 1988.

[10] K. Marriott and H. Sondergaard, Bottom-up abstract interpretation of logic programs, *Proc. of 1988 Joint Int'l Conf. and Symp. on Logic Programming*, pp. 733-748.

창 병 모

1988년 서울대학교 컴퓨터공학과 졸업(학사). 1990년 한국과학기술원 전산학과에서 공학석사 학위 취득. 1994년 한국과학기술원 전산학과에서 공학박사 학위 취득. 1994년 ~ 1995년 한국전자통신연구소 박사후 연수 연구원. 1995년 ~ 현재 숙명여자대학교 전산학과 조교수. 관심분야는 컴파일러 구성론(정적 분석, 코드 최적화), 논리 프로그래밍, 연역 데이타베이스