공개 집합 제한 논리 언어의 구현 방법

신 동 하† · 손 성 훈††

요 약

'집합 제한 논리 언어'는 '집합 이론'을 프로그래밍에 도입한 언어이다. 본 논문은 A. Dovier 연구팀이 제안한 집합 제한 문제 풀이(solver) 절차를 소개하고, 이 절차가 논리 언어 Prolog 상에서 어떻게 구현 가능한 지를 보인다. 이 절차는 '다시쓰기 규칙(rewrite rule)'으로 표현되어 있는데 이 표현의 특징은 일반 프로그래밍 언어가 표현하기 힘든 비결정적 규칙 적용(nondeterministic rule application)과 수학적 변수(mathematical variable)를 사용한다는 점이다. 본 연구에서는 이들 특징이 Prolog 언어에서 제공되는 비결정적 제어 (nondeterministic control), 논리적 변수(logical variable) 및 리스트(list) 자료구조의 사용으로 쉽게 구현 가능함을 보인다. 본 연구의 구현은 다음과 같은 의의를 가지고 있다. 첫째 본연구는 이 언어의 모든 기능을 완전하게 구현하였다는 점이다. 둘째 본 연구에서는 이 언어의 구현 방법을 누구나 알기 쉽게 기술하였다는 점이다. 셋째 기존의 구현이 상업적 Prolog인 SICStus Prolog를 사용하여 구현한 것과는 달리 본 구현은GNU GPL(General Public License)을 가지는CIAO Prolog를 사용하여 구현하였기 때문에 누구나 자유롭게 사용할 수 있는 점이다. 넷째 본 연구에서 개발된 소스 코드는 공개 소프트웨어이기 때문에 누구나 자유롭게 사용, 수정 및 배포할 수 있다는 점이다.

키워드 : 집합, 집합 제한 논리 언어, Prolog 언어, 공개 소스 소프트웨어

An Implementation of Open Set Constraint Logic Language

Dongha Shin† · Sunghoon Son††

ABSTRACT

Set constraints logic language is a language that adopts 'set theory' in programming. In this paper, we introduce the procedure for solving set constraints proposed by A. Dovier and show how the procedure can be implemented in logic language Prolog. The procedure is represented in 'rewriting rules' and this representation is characterized by having nondeterministic rule applicationsand mathematical variables that is difficult to be implemented in general programming languages. In this paper, we show that the representation can be easily implemented by using nondeterministic control, logical variables and data structure 'list' provided in Prolog. Our implementation has following advantages.First we have implemented the full features of the language. Second we have described the implementation detail in thisresearch. Third other used the commercial Prolog called SICStus, but we are using CIAO Prolog with GNU GPL(General Public License) and anyone can use it freely. Forth the software of our implementation is open source so anyone can use, modify, and distribute it freely.

Key Words : set, set constraints logic language, Prolog Language, Open Source Software

1. 서 론

'집합 제한 논리 언어(set constraint logic language)'[4] [6, 12, 13]는 '집합 이론(set theory)'[1]을 사용하여 프로그래밍하는 새로운 패러다임(paradigm)의 언어이다. 본 논문에서는 최근 A. Dovier 연구팀이 제안한 집합 제한 논리 언어[3]를 논리 언어[10] Prolog[11]를 사용하여 구현하는 방법을 기술한다. 본 연구에서 개발하는 집합 제한 논리 언어를 간단히 'OSCL(Open Set Constraint Logic language)'이라 명명한다. 본 논문은 A. Dovier 연구팀이 완전성을 증명[3]한 '집합 제한 문제 풀이(set constraint problem solver)' 절차인 '다시쓰기 규칙(rewriting rule)'을 사용한다. 다시쓰기 규칙은 일반 프로그래밍 언어에서는 의미 표현이 힘든 '비결정성 규칙 적용(non-deterministic rule application)'과 '수학적 변수(mathematical variable)'를 사용한다는 점이 특징이다. 본 연구에서는 이 특징이 Prolog 언어에서 제공되는 '비결정성(non-determinism), 논리적 변수(logical variable)' 및 리스트(list) 자료구조[9, 11]의 사용으로 쉽게 구현 가능함을 보인다.

본 연구의 구현은 다음과 같은 의의를 가지고 있다. 첫째 집합 제한 논리 언어의 구현 사례가 많지 않은 현실에서 본 연구는 언어의 모든 기능을 완전하게 구현하고 시험하였다는 점이다. 둘째 본 연구에서는 이 언어의 구현 방법을 비

※ 본 연구는 상명대학교 자연과학연구소로부터 연구비 지원을 받았습니다.
† 정 회 원 : 상명대학교 소프트웨어학부 부교수
†† 정 회 원 : 상명대학교 소프트웨어학부 전임강사
논문접수 : 2005년 2월 7일, 심사완료 : 2005년 7월 26일

교적 쉽게 이해할 수 있도록 기술하였다는 점이다. 셋째 기존 A. Dovier 연구팀이 상업적 Prolog인 SICStus Prolog[15]를 사용하여 구현한 것과는 달리 본 구현은 GNU GPL(General Public License)[8]을 가지는 CIAO Prolog[16, 17]를 사용하여 구현하였기 때문에 누구나 무료로 사용할 수 있는 점이다. 넷째 본 연구에서 개발된 소스 코드는 공개 소프트웨어[19]이기 때문에 누구나 자유롭게 사용하고 수정 배포할 수 있는 점이다. 현재 개발된 언어인 OSCL(Open Set Constraints Logic language)는 한국소프트웨어진흥원이 설립한 공개소프트웨어지원센터[18]의 웹 사이트에 공개하였다.

본 논문의 2장에서는 OSCL을 표현하기 위한 구문(syntax)과 의미(semantics)를 정의한다. 3장에서는 2장에서 정의한 언어의 구문과 의미를 가지고 다시쓰기 규칙을 기술한다. 다시쓰기 규칙은 집합 제한 문제를 풀기 위해 핵심이 되는 부분이다. 4장에서는 3장에서 기술된 다시쓰기 규칙이 Prolog 언어에서 어떻게 표현되어 구현 가능한지를 기술한다. 5장에서는 본 연구팀이 구현한 OSCL이 잘 구현되었음을 뒷받침하기 위한 동작 시험을 하고 간단한 보기 프로그램을 실제 수행한 결과를 보인다. 6장에서는 본 연구의 의의와 추후 연구 계획을 설명한다.

2. OSCL 언어

본 장에서는 OSCL의 구문(syntax)과 의미(semantics)를 정의한다.

2.1 구문

OSCL의 표현은 아래와 같은 심볼을 사용하여 텀(term)과 '집합 제한(set constraint)'을 표현한다.

- 함수 심볼: '{·|·}'(집합), 'Φ'(공집합), 사용자 정의 심볼
- 술어 심볼: '=', '≠', '∈', '∉', '$\cup_3$', '$\neg\cup_3$', '∥', '∦', 'set'
- 변수 심볼: 영문 대문자로 시작되는 단어(word)
- 연결자 심볼: '∧'(and)

앞에서 정의한 심볼을 사용하여 표현한 집합 {X|Y}의 의미는 {X}∪Y를 의미한다. 이 때, Y는 항상 집합을 나타내는 텀(집합 또는 집합 변수)이어야만 한다. 집합 $\{t_1|\{t_2|\{t_3|\cdots\{t_n|t\}\cdots\}\}\}$은 간단히 $\{t_1, t_2, t_3, \cdots, t_n|t\}$로 표현된다. 만약 t=Φ이면 간단히 $\{t_1, t_2, t_3, \cdots, t_n\}$으로 표현된다. OSCL로 구현하는 프로그램은 "집합 제한(set constraint)"의 연결('∧')로 구성된다. 예를 들어 'X={a|{b|Y}} ∧ Y∉Z ∧ Z∥X'은 OSCL로 표현된 집합 제한 논리 프로그램이다.

2.2 의미

앞 절에서 정의한 집합 제한 논리 프로그램의 의미는 수학에서 정의하는 집합 관련 기호의 의미와 동일하다. 특히 집합 등식에 대한 의미는 아래에 정의하는 공리(axiom) Ab(Absorption)와 Cl(Commutative on left)[2]을 만족해야만 한다.

- 공리 Ab: {X|{X|Z}}={X|Z}
- 공리 Cl: {X|{Y|Z}}={Y|{X|Z}}

공리 Ab는 집합의 동일성(idempotency) 성질, 공리 Cl은 집합의 교환 가능성(commutativity)성질을 나타낸다. 이 두 공리를 간단히 '(Ab)(Cl) 공리'라고 한다. '(Ab)(Cl) 공리'와 동등한 의미를 가지는 다른 표현[2]은 다음과 같으며 다음 장에서 설명하는 집합 등식의 다시쓰기 규칙에 활용된다.

$$\begin{aligned}\{Y1|V1\}=\{Y2|V2\} \leftrightarrow\ & (Y1=Y2 \wedge V1=V2) \vee \\ & (Y1=Y2 \wedge V1=\{Y2|V2\}) \vee \\ & (Y1=Y2 \wedge \{Y1|V1\}=V2) \vee \\ & (V1=\{Y2|N\} \wedge V2=\{Y1|N\})\end{aligned}$$

3. 다시쓰기 규칙(rewriting rules)

OSCL의 구현에서 핵심 되는 부분은 다시쓰기 규칙이다. 이것은 집합 제한(set constraints)으로 연결된 문제를 해결된 형식(solved form)이 될 때까지 다시쓰는(rewriting) 규칙을 정의한 것이다. 이 다시쓰기 규칙은 A. Dovier 연구팀이 제안하고 완전성을 증명하였다[3]. "다시쓰기 규칙"은 '=', '≠', '∈', '∉', '$\cup_3$', '$\neg\cup_3$', '∥', '∦', 그리고 'set' 다시쓰기 규칙으로서 총 9가지 종류로 구성되어 있다. 다시 쓰기 규칙은 다음과 같은 심볼(symbols)을 사용하여 표현한다.

- →: 다시쓰기 심볼(rewriting symbol)로서 '→'을 중심으로 왼쪽 부분이 해결할 문제이고 오른쪽 부분은 그 문제를 다시 쓰는 부분이다.
- C': '∧'으로 연결된 제한들(constraints)
- X, Y, Z: 변수
- t, t_i, s, s_i: 일반 텀들(generic terms)
- f, g: 함수 심볼(function symbols)
- N, N1, N2: 다시쓰기 규칙을 적용하는 도중 생성되는 변수

'=' 제한은 두 집합 또는 텀(term)의 동등함을 제한한다. '=' 제한 문제는 집합 일치화 문제(set unification problem)[2]라고도 하는데, 이는 고전적 논리 언어[5, 20]에서 다루는 일치화 문제[7, 10]를 집합 표현까지 확장한 문제이다. '=' 다시쓰기 규칙은 다음과 같다.

'=' 다시쓰기 규칙
E1) X=X ∧ C' → C'
E2) t=X ∧ C', t is not a variable → X=t ∧ C'
E3) X=f(t1, ⋯, tn) ∧ C', f ≠{·|·}, X∈vars(t1, ⋯, tn) → false
E4) X={t0, ⋯, tn|t} ∧ C', t is Φ or a variable, X∈vars(t0, ⋯, tn) → false
E5) X=t ∧ C', X∉vars(t), t is a set term or set(X)∉C' → X=t ∧ C'[X/t]
E6) X={t0, ⋯, tn|X} ∧ C', X∉vars(t0, ⋯, tn) → X={t0, ⋯, tn|N} ∧ set(N) ∧ C'
E7) f(s1, ⋯, sm)=g(t1, ⋯, tn) ∧ C', f ≠g → false
E8) f(s1, ⋯, sn)=f(t1, ⋯, tn) ∧ C', f ≠{·|·} → s1=t1 ∧ ⋯ ∧ sn=tn ∧ C'
E9) {t|s}={t'|s'} ∧ C', tail(s), tail(s') are not the same var → C' ∧ any of
E9$_1$) t=t' ∧ s=s'
E9$_2$) t=t' ∧ {t|s}=s'
E9$_3$) t=t' ∧ s={t'|s'}
E9$_4$) s={t'|N} ∧ {t|N}=s' ∧ set(N)
E10) {t0, ⋯, tm|X}={t'0, ⋯, t'n|X} ∧ C' → C' ∧ any of
E10$_1$) t0=t'j ∧ {t1, ⋯, tm|X}={t'0, ⋯, t'j-1, t'j+1, ⋯, t'n|X}

$E10_2$) $t0=t'j \wedge \{t0, \cdots, tm\|X\}=\{t'0, \cdots, t'j-1, t'j+1, \cdots, t'n\|X\}$
$E10_3$) $t0=t'j \wedge \{t1, \cdots, tm\|X\}=\{t'0, \cdots, t'n\|X\}$
$E10_4$) $X=\{t0\|N\} \wedge \{t1, \cdots, tm\|N\}=\{t'0, \cdots, t'n\|N\} \wedge set(N)$ for any j in $\{0, \cdots, n\}$
E11) $X=f(t1, \cdots, tn) \wedge set(X) \wedge C', f \neq \{\cdot\|\cdot\}, f \neq \Phi \rightarrow error$

'≠' 제한은 위의 '=' 제한의 부정임을 제한한다.

'≠' 다시쓰기 규칙
EN1) $f(s_1, \cdots, s_m) \neq g(t_1, \cdots, t_n) \wedge C' \rightarrow C'$
EN2) $f(s_1, \cdots, s_n) \neq f(t_1, \cdots, t_n) \wedge C', f \neq \{\cdot\|\cdot\}, n>0 \rightarrow C' \wedge$ any of
$EN2_1$) $s_1 \neq t_1$
⋯
$EN2_n$) $s_n \neq t_n$
EN3) $s \neq s \wedge C'$, s is a constant or a variable → false
EN4) $t \neq X \wedge C'$, t is not a variable → $X \neq t \wedge C'$
EN5) $X \neq f(t_1, \cdots, t_n) \wedge C', f \neq \{\cdot\|\cdot\}, X \in vars(t_1, \cdots, t_n) \rightarrow C'$
EN6) $X \neq \{t_1, \cdots, t_n\|t\} \wedge C', X \in vars(t_1, \cdots, t_n) \rightarrow C'$
EN7) $X \neq \{t_1, \cdots, t_n\|X\} \wedge C', X \notin vars(t_1, \cdots, t_n) \rightarrow C' \wedge$ any of
$EN7_1$) $t_1 \notin X$
⋯
$EN7_n$) $t_n \notin X$
EN8) $\{s\|r\} \neq \{u\|t\} \wedge C' \rightarrow C' \wedge$ any of
$EN8_1$) $N \in \{s\|r\} \wedge N \notin \{u\|t\}$
$EN8_2$) $N \in \{u\|t\} \wedge N \notin \{s\|r\}$

'∈' 제한은 한 원소가 어떤 집합 내의 한 원소임을 제한한다.

'∈' 다시쓰기 규칙
M1) $s \in \Phi \wedge C' \rightarrow$ false
M2) $r \in \{s\|t\} \wedge C' \rightarrow C' \wedge$ any of
$M2_1$) $r=s$
$M2_2$) $r \in t$
M3) $t \in X \wedge C' \rightarrow X=\{t\|N\} \wedge set(N) \wedge C'$

'∉' 제한은 '∈' 제한의 부정(negation)임을 제한한다.

'∉' 다시쓰기 규칙
MN1) $s \notin \Phi \wedge C' \rightarrow C'$
MN2) $r \notin \{s\|t\} \wedge C' \rightarrow r \neq s \wedge r \notin t \wedge C'$
MN3) $t \notin X \wedge C', X \in vars(t) \rightarrow C'$

'$\cup_3$' 제한은 첫 번째 인수와 두 번째 인수의 합집합(union)이 세 번째 인수라는 것을 제한한다.

'$\cup_3$' 다시쓰기 규칙
U1) $\cup_3(s, s, t) \wedge C' \rightarrow s=t \wedge C'$
U2) $\cup_3(s, t, \Phi) \wedge C', s \neq t \rightarrow s=\Phi \wedge t=\Phi \wedge C'$
U3) $\cup_3(\Phi, t, X) \wedge C'$ or $\cup_3(t, \Phi, X) \wedge C', t \neq \Phi \rightarrow X=t \wedge C'$
U4) $\cup_3(s_1, s_2, \{t_1\|t_2\}) \wedge C', s_1 \neq s_2 \rightarrow \{t_1\|t_2\}=\{t_1\|N\} \wedge t_1 \notin N \wedge C' \wedge$ any of
$U4_1$) $s_1=\{t_1\|N1\} \wedge t_1 \notin N1 \wedge \cup_3(N1, s_2, N)$
$U4_2$) $s_2=\{t_1\|N1\} \wedge t_1 \notin N1 \wedge \cup_3(s_1, N1, N)$
$U4_3$) $s_1=\{t_1\|N1\} \wedge t_1 \notin N1 \wedge s_2=\{t_1\|N2\} \wedge t_1 \notin N2 \wedge \cup_3(N1, N2, N)$
U5) $\cup_3(\{t_1\|t_2\}, t, X) \wedge C'$ or $\cup_3(t, \{t_1\|t_2\}, X) \wedge C', t \neq \{t_1\|t_2\}, t \neq \Phi \rightarrow \{t_1\|t_2\}=\{t_1\|N1\} \wedge t_1 \notin N1 \wedge X=\{t_1\|N\} \wedge t_1 \notin N \wedge C' \wedge$ any of
$U5_1$) $t_1 \notin t \wedge \cup_3(N1, t, N)$
$U5_2$) $t=\{t_1\|N2\} \wedge t_1 \notin N2 \wedge \cup_3(N1, N2, N)$
U6) $\cup_3(X, Y, Z) \wedge Z \neq t \wedge C', X \neq Y \rightarrow \cup_3(X, Y, Z) \wedge C' \wedge$ any of
$U6_1$) $N \in Z \wedge N \notin t$
$U6_2$) $N \in t \wedge N \notin Z$
$U6_3$) $Z=\Phi \wedge t \neq \Phi$
U7) $\cup_3(X, Y, Z) \wedge X \neq t \wedge C'$ or $\cup_3(Y, X, Z) \wedge X \neq t \wedge C', X \neq Y \rightarrow \cup_3(X, Y, Z) \wedge C' \wedge$ any of
$U7_1$) $N \in X \wedge N \notin t$
$U7_2$) $N \in t \wedge N \notin X$
$U7_3$) $X=\Phi \wedge t \neq \Phi$

'$\neg\cup_3$' 제한은 위의 '$\cup_3$' 제한의 부정임을 제한한다.

'$\neg\cup_3$' 다시쓰기 규칙
UN1) $\neg\cup_3(s_1, s_2, s_3) \wedge C' \rightarrow C' \wedge$ any of
$UN1_1$) $N \in s_3 \wedge N \notin s_1 \wedge N \notin s_2$
$UN1_2$) $N \in s_1 \wedge N \notin s_3$
$UN1_3$) $N \in s_2 \wedge N \notin s_3$

'‖' 제한은 두 집합의 교집합(intersection)이 공집합임을 제한한다.

'‖' 다시쓰기 규칙
D1) $\Phi \| t \wedge C'$ or $t \| \Phi \wedge C' \rightarrow C'$
D2) $X \| X \wedge C' \rightarrow X=\Phi \wedge C'$
D3) $\{t_1\|t_2\} \| X \wedge C'$ or $X \| \{t_1\|t_2\} \wedge C' \rightarrow t_1 \notin X \wedge X \| t_2 \wedge C'$
D4) $\{t_1\|s_1\} \| \{t_2\|s_2\} \wedge C' \rightarrow t_1 \neq t_2 \wedge t_1 \notin s_2 \wedge t_2 \notin s_1 \wedge s_1 \| s_2 \wedge C'$

'∦' 제한은 위의 '‖' 제한의 부정임을 제한한다.

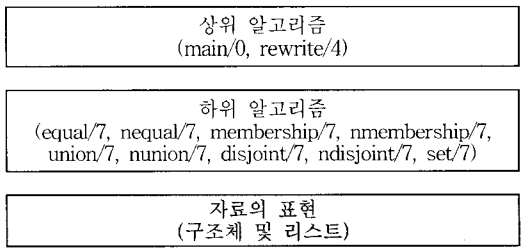
'∦' 다시쓰기 규칙
DN1) $s \nparallel t \wedge C' \rightarrow N \in s \wedge N \in t \wedge C'$

'set' 제한은 인수가 집합 또는 집합 변수라고 제한한다.

'set' 다시쓰기 규칙
S1) $set(\Phi) \wedge C' \rightarrow C'$
S2) $set(\{t\|s\}) \wedge C' \rightarrow set(s) \wedge C'$
S3) $set(f(t_1, \cdots, t_m)) \wedge C', f \neq \{\cdot\|\cdot\}, f \neq \Phi \rightarrow error$

4. OSCL의 구현 방법

본 장에서는 3장의 다시쓰기 규칙을 Prolog 언어에서 구현하는 방법을 설명한다. 먼저 구현에서 사용되는 자료 표현 방법을 설명하고, 다시쓰기 구현의 상위 레벨의 구현 알고리즘 및 하위 레벨 알고리즘을 설명한다. 아래 그림은 OSCL의 단계적 구조이다.



(그림 1) OSCL 단계적 구조

4.1 자료의 표현

표현에는 사용자(프로그래머)가 사용하는 외부 표현과 구현에 사용하는 내부 표현이 있다. 표현의 대상은 집합, 제한술어 및 연결자이다. 여기서 내부 표현은 Prolog 언어의 구조체 및 리스트를 사용한다.

집합의 표현			
외부 표현	내부 표현	외부 표현	내부 표현
공집합 {}	snil	집합 {a,b,c}	set(a,set(b,set(c,snil)))

제한 술어 및 연결자의 표현			
외부 표현	내부 표현	외부 표현	내부 표현
$\cdot = \cdot$	$\cdot = \cdot$	$\cdot \neq \cdot$	neq(·,·)
$\cdot \in \cdot$	in(·,·)	$\cdot \notin \cdot$	nin(·,·)
$\cup_3(\cdot, \cdot, \cdot)$	uni(·,·,·)	$\neg\cup_3(\cdot, \cdot, \cdot)$	nuni(·,·,·)
$\cdot \parallel \cdot$	dis(·,·)	$\cdot \nparallel \cdot$	ndis(·,·)
set(·)	set(·)	$\cdot \wedge \cdot$	[·, ·]

4.2 상위 알고리즘

상위 알고리즘은 주(main) 알고리즘과 다시쓰기 상위 알고리즘으로 구성된다. 주 알고리즘은 사용자가 입력하는(read_term/2) 프로그램을 내부 표현으로 변환하고(preprocess/2), 내부 표현을 다시쓰기 상위 알고리즘에 적용하고(rewrite/4), 최종 답을 외부 표현으로 바꾸어(postprocess/2) 출력하는 일(write_ans/2)을 반복한다. 다음은 상위 알고리즘에 소속된 주 알고리즘이다.

상위 알고리즘 - 주 알고리즘

```
main :-
  read_term(Clause,...),              % 입력
  preprocess(Clause,Clause2),         % 내부 표현 변환
  Ns=Clause2,Aux=[],So=[],            % Ns, Aux, So, Ans 초기화
  rewrite(Ns,Aux,So,Ans),             % 다시쓰기 규칙 적용
  postprocess(Ans,Ans2),              % 외부 표현 변환
  write_ans(Ans2),                    % 출력
  main.                               % 반복
```

다시쓰기 상위 알고리즘인 rewrite/4는 4개의 인수를 가지는데 이들 각각의 의미는 다음과 같다. 첫째 인수는 해결하지 않은 제한(Ns: Not solved), 두 번째 인수는 문제 풀이 시 중간 문제(intermediate problem)를 저장하는 임시 스택(Aux: Auxiliary stack), 세 번째 인수는 풀이가 저장되는 곳(So: Solved), 네 번째 인수는 답(Ans: Answer)을 의미한다. 알고리즘 rewrite/4는 다시쓰기 종료 조건을 검사하기 위한 절(clause)이 1개 있고, 각 제한에 따라 2개의 절로 구성되어 있다. 이중 첫째 절은 두 번째 인수인 Aux가 비어있을 때 Ns에 있는 제한의 종류를 점검하여 관련 하부 다시쓰기 규칙을 부르기 위한 절이고, 두 번째 절은 Aux 스택의 탑에 있는 제한의 종류에 따라 관련 하부 다시쓰기 규칙을 부르는 절이다. rewrite/4는 Prolog의 비결정적 제어를 가지고 있어서 3장에 설명한 다시쓰기 규칙의 비결정적 적용을 정확히 구현한다.

상위 알고리즘 - 다시쓰기 상위 알고리즘

```
% 다시쓰기 종료를 위한 절
rewrite([],[],Ans,Ans) :- !.
% 다시쓰기 제한 '='를 위한 절
rewrite([X=Y|NsTail],[],So,Ans):-
  !,
  equal(NsTail,[],So,X=Y,Ns2,Aux2,So2),          % 4.3 참고
  rewrite(Ns2,Aux2,So2,Ans).
rewrite(Ns,[X=Y|AuxTail],So,Ans):-
  !,
  equal(Ns,AuxTail,So,X=Y,Ns2,Aux2,So2),         % 4.3 참고
  rewrite(Ns2,Aux2,So2,Ans).
% 나머지 8개 제한도 제한 위 '='과 동일한 알고리즘임
```

4.3 하위 알고리즘

하위 알고리즘은 3장에서 설명한 각 제한 별로 하나씩 존재하는 다시쓰기 규칙을 구현하는 알고리즘이다. 이를 구현하는 술어는 각 제한의 순서에 따라 equal/7, nequal/7, membership/7, nmembership/7, union/7, nunion/7, disjoint/7, ndisjoint/7 그리고 set/7인데, 본 구현의 특징은 이들 인수의 형식 및 의미가 같아서 구현 시 이해가 쉽고 오류 발견 및 수정이 쉽다는 점이다. 첫 번째와 인수는 해결해야 할 제한 리스트(Ns)이고, 두 번째 인수는 스택처럼 사용되는 것으로 첫 번째 인수에 있는 제한 리스트보다 먼저 해결해야 되는 제한 리스트(Aux)이다. 세 번째 인수는 문제 해결된 형식 리스트(So)이다. 첫 번째, 두 번째, 세 번째 인수는 값을 받는 입력(in) 인수이고, 다섯 번째, 여섯 번째, 일곱 번째 인수는 첫 번째, 두 번째, 세 번째 인수와 같은 의미의 출력(out) 인수이다. 네 번째 인수는 현재 해결해야 하는 제한 문제이다. 본 절에서는 OSCL의 9개의 제한 술어 중 가장 중요한 술어인 equal/7 및 membership/7 중심으로 기술한다.

4.3.1 equal/7의 구현

3장의 규칙 E1~E8은 일반 술어 논리의 일치화(unification)[10] 문제와 같으므로 자세한 설명은 생략한다. E9 및 E10의 구현이 본 구현의 핵심이다. 이들의 구현 알고리즘은 다음과 같다.

하위 알고리즘 - '=' 다시쓰기 알고리즘

```
equal(Ns1,Aux1,So1,set(T1,S1)=set(T2,S2),Ns2,Aux2,So1) :- % E9₁
  Ns2=[T1=T2|Ns1],
  Aux2=[S1=S2|Aux1].
equal(Ns1,Aux1,So1,set(T1,S1)=set(T2,S2),Ns2,Aux2,So1) :- % E9₂
  Ns2=[T1=T2|Ns1],
  Aux2=[set(T1,S1)=S2|Aux1].
equal(Ns1,Aux1,So1,set(T1,S1)=set(T2,S2),Ns2,Aux2,So1) :- % E9₃
  Ns2=[T1=T2|Ns1],
  Aux2=[S1=set(T2,S2)|Aux1].
equal(Ns1,Aux1,So1,set(T1,S1)=set(T2,S2),Ns1,Aux2,So1) :- % E9₄
  Aux2=[set(T1,N)=S2,S1=set(T2,N),set(N)|Aux1].
equal_ten(Ns1,Aux1,So1,set(T1,S1)=set(_,_),Select,Remain, % E10₁
  Ns2,Aux2,So1) :-
  Ns2=[T1=Select|Ns1],
  Aux2=[S1=Remain|Aux1].
equal_ten(Ns1,Aux1,So1,set(T1,S1)=set(_,_),Select,Remain, % E10₂
  Ns2,Aux2,So1) :-
  Ns2=[T1=Select|Ns1],
  Aux2=[set(T1,S1)=Remain|Aux1].
equal_ten(Ns1,Aux1,So1,set(T1,S1)=set(T2,S2),Select,_,    % E10₃
  Ns2,Aux2,So1) :-
  Ns2=[T1=Select|Ns1],
  Aux2=[S1=set(T2,S2)|Aux1].
equal_ten(Ns1,Aux1,So1,set(T1,S1)=set(T2,S2),_,_,         % E10₄
```

규칙 E9의 구현은 4개의 절로 구성되는데 이들 각각은 3장의 다시쓰기 규칙 $E9_1$~$E9_4$에 해당하는 비결정적 제어를 가진 절이다. 규칙 E10의 구현도 4개의 절로 구성되어 있지만 이들은 2개의 새로운 인수인 Select 및 Remain을 가지고 있는데 이들은 E10의 구현(equal_ten/9)을 용이하게 하기 위하여 도입된 변수로 Select는 입력 집합에서 비 결정적으로 선택된 하나의 원소를 저장하는 인수이고 Remain은 이 원소

를 제외한 집합을 의미한다. 이 선택에서 주어진 집합의 크기와 같은 수의 비결정적 수행이 가능하다.

4.3.2 membreship/7의 구현

membership/7은 3장의 M1~M3에 해당하는 구현이다. 구현 알고리즘은 다음과 같다.

하위 알고리즘 - '∈' 다시쓰기 알고리즘
membership(Ns1,Aux1,So1,in(Term,X),Ns1,Aux2,So1) : -% M2 var(X), !, Aux2=[X=set(Term,N),set(N)|Aux1]. membership(Ns1,Aux1,So1,in(R,set(S,T)),Ns1,Aux2,So1) :- % M3 !, (Aux2=[R=S|Aux1] ; Aux2=[in(R,T)|Aux1]). membership(_,_,_,in(X,Y),_,_,_) :- % M1 Y==snil, fail.

위 알고리즘에서 세 번째 절이 M1에 해당하는 공집합(snil) 점검 부분이고 첫 번째 및 두 번째 절이 각각 M2 및 M3 구현 부분이다.

5. 동작 시험

본 장에서는 OSCL의 각 제한 술어들이 잘 구현되었음을 시험하기 위해 각 제한 술어마다 동작 시험을 한 결과를 보인다. 또한 지도 색칠하기 문제(map coloring problem)를 OSCL로 풀어봄으로써 구현의 정확성을 검증한다.

5.1 각 제한 술어의 동작 시험 결과

5.1.1 '=' 동작 시험 결과

입력 프로그램	답
X={a,b;S},X=Z,Y={Z}	X={a,b;S}, Y={{a,b;S}}, Z={a,b;S}
X={a,b,c;X},Y={aa,bb,cc;Y}	X={a,b,c;_1},Y={aa,bb,cc;_2}, [set(_1), set(_2)]
{T;S}={T1;S}	T1=T or S={T;_1}, T1=T, [set(_1)] or S={T;_2}, T1=T, [set(_2)] or S={T,T1;_3}, [set(_3)]
{a;X}={A;X}	A=a or A=a, X={a;_1} [set(_1)] or A=a, X={a;_2} [set(_2)] or X={a,A;_3} [set(_3)]

5.1.2 '∈' 동작 시험 결과

입력 프로그램	답
in(r,{a,R1,b,R2})	R1=r or R2=r
in(t,{a,b;X})	X={t;_1} [set(_1)]
in(a,{b,X,Y;Z})	X=a or Y=a or Z={a;_1} [set(_1)]

5.1.3 '∉' 동작 시험 결과

입력 프로그램	답
nin(r,{a,b,RN})	[neq(RN,r)]
nin(N,{b,c})	[neq(N,b), neq(N,c)]

5.1.4 '≠' 동작 시험 결과

입력 프로그램	답
neq(f(a,B,c),f(A,b,C))	[neq(A,a)] or [neq(B,b)] or [neq(C,c)]
neq(X,{a,b,c;X})	[nin(a,X)] or [nin(b,X)] or [nin(c,X)]

5.1.5 '∪3' 동작 시험 결과

입력 프로그램	답
uni({X},{b},{a,Y})	X=a, Y=b
uni({X},{a,b},V)	V={X,a,b} [neq(X,a), neq(X,b)] or V={a,b}, X=a or V={b,a}, X=b
uni({X},{Y},V)	V={X,Y} [neq(X,Y)] or V={X}, Y=X

5.1.6 '⊓∪3' 동작 시험 결과

입력 프로그램	답
nuni({a},{b},Z)	Z={_1;_2} [set(_1), neq(_2,a), neq(_2,b)] or [nin(a,Z)] or [nin(b,Z)]

5.1.7 '∥' 동작 시험 결과

입력 프로그램	답
dis({a,b},X)	[nin(a,X), nin(b,X)]
dis({X,Y},{a;Z})	[neq(X,a), nin(X,Z), neq(Y,a), nin(Y,Z)]

5.1.8 '∦' 동작 시험 결과

입력 프로그램	답
ndis({a,b},{X,Y})	X=a or Y=a or X=b or Y=b
ndis({a},{X,b})	X=a

5.1.9 'set' 동작 시험 결과

입력 프로그램	답
set({t;S})	[set(S)]
set(X),X=f(a)	false

5.2 지도 색칠하기 문제

입력 프로그램	답
R={GG,GW,CC,JR,GS}, R={red,blue,green}, nin(GG,{GW,CC}), nin(GW,{GG,CC,GS}), nin(CC,{GG,GW,JR,GS}), nin(JR,{CC,GS}), nin(GS,{GW,CC,JR})	CC=green, GG=red, GS=red, GW=blue, JR=blue, R={red,blue,green,blue,red} or CC=blue, GG=red, GS=red, GW=green, JR=green,R={red,green,blue,green,red} or CC=green, GG=blue, GS=blue, GW=red, JR=red, R={blue,red,green,red,blue} or CC=red, GG=blue, GS=blue, GW=green, JR=green,R={blue,green,red,green,blue} or CC=blue, GG=green, GS=green, GW=red, JR=red, R={green,red,blue,red,green} or CC=red, GG=green, GS=green, GW=blue, JR=blue,R={green,blue,red,blue,green}

본 연구에서 구현한 OSCL을 사용하여 지도 색칠하기 문제(map coloring problem)를 수행시켜봄으로써 구현의 정확성을 검증해볼 수 있다. 지도 색칠하기 문제를 남한의 5도

로 예를 들어 설명한다. 지역은 경기도(GG), 강원도(GW), 충청도(CC), 전라도(JL) 그리고 경상도(GS)이고, 인접한 지역으로 경기도는 강원도와 충청도이고, 강원도는 경기도, 충청도, 경상도이고, 충청도는 경기도, 강원도, 전라도, 경상도이고, 전라도는 충청도와 경상도이고, 경상도는 강원도, 충청도, 전라도이다. 지도를 색칠할 색은 빨강, 파랑, 초록을 사용하였다. 두 가지 색 사용시 수행결과는 false가 나왔고 세 가지 색 사용시 수행결과는 아래와 같다. 아래의 결과로 남한의 5도를 색칠할 수 있는 최소한의 색의 수는 3가지임을 알 수 있다.

6. 결 론

본 논문은 A. Dovier 연구팀이 제안한 집합 제한 문제 풀이(solver) 절차[3]를 바탕으로, 논리 언어 Prolog[9, 11]를 사용하여 집합 제한 논리 언어를 구현하였다. 구현의 핵심인 다시쓰기 규칙은 일반 프로그래밍 언어에서는 구현이 매우 힘들지만 Prolog 언어에서 제공되는 비결정적 제어(non-deterministic control), 논리적 변수(logical variable) 및 리스트(list) 자료구조의 사용으로 쉽게 구현 가능하였다. 본 연구의 구현은 다음과 같은 의의를 가지고 있다. 첫째 본 연구는 유사 언어의 구현이 많지 않은 현실에서 언어의 모든 기능을 완전하게 구현하였다는 점이다. 둘째 본 연구에서는 이 언어의 구현 방법을 누구나 알기 쉽게 기술하였다는 점이다. 셋째 기존 A. Dovier 연구팀이 상업적 Prolog인 SICStus Prolog[15]를 사용하여 구현한 것과는 달리 본 구현은 GNU GPL(General Public License)[8]을 가지는 CIAO Prolog[16, 17]를 사용하여 구현하였기 때문에 누구나 자유롭게 사용할 수 있는 점이다. 넷째 본 연구에서 개발된 소스 코드는 공개 소프트웨어[19]이기 때문에 누구나 자유롭게 사용, 수정 및 배포할 수 있다는 점이다. 본 연구는 정확한 구현에 중점 두었지만 앞으로 효율적인 구현에 관한 연구가 남아있다. 본 연구에서 구현된 OSCL는 한국소프트웨어진흥원이 설립한 공개소프트웨어지원센터[18]에 공개하였다.

참 고 문 헌

[1] A. Dovier, Computable Set Theory and Logic Programming, PhD Thesis TD-1/96, Universita degli Studi di Pisa, dip. di Informatica, March, 1996.

[2] A. Dovier, E. Pontelli, and G. Rossi, Set Unification, Rapporto di Ricerca, Dipartimento di Matematica, Universita di Parma, n.310, 2002.

[3] A. Dovier, C. Piazza, E. Pontelli, and G. Rossi, Sets and Constraint Logic Programming, "ACM Transactions on Programming Languages and Systems," Vol.22, No.5, pp.861-931, 2000.

[4] A. Dovier, E.G. Omodeo, E. Pontelli, and G. Rossi, {log}: A Language for Programming in Logic with Finite Sets, "The Journal of Logic Programming," Vol.28, No.1, pp.1-44, 1996.

[5] Chin-Liang Chang and Richard Lee, "Symbolic Logic and Mechanical Theorem Proving," Academic Press, 1973.

[6] Escher NG Manual, http://users.unimi.it/~ddl/vega/manual/escher_ng/

[7] F. Baader and W. Snyder, Unification theory, "Handbook of Automated Reasoning," Elsevier Science Publishers B. V., 1999.

[8] GNU General Public License, http://www.gnu.org/copyleft/gpl.html

[9] I. Bratko, "PROLOG Programming for Artificial Intelligence," Addison-Wesley, 2000.

[10] J. W. Lloyd, "Foundations of Logic Programming," Springer-Verlag, 1984.

[11] M. F. Clocksin and C. S. Mellish, "Programming in Prolog, fourth edition," Springer-Verlag, 1994.

[12] Mozart Documentation, http://www.mozart-oz.org/documentation/index.html

[13] Pat M. Hill and John W. Lloyd, The Godel Programming Language, MIT Press, 1994.

[14] Quintus Prolog Homepage, http://www.sics.se/quintus/

[15] SICStus Prolog Homepage, http://www.sics.se/isl/sicstuswww/site/index.html

[16] The CIAO Prolog Development System WWW Site, http://clip.dia.fi.upm.es/Software/Ciao/

[17] The CIAO Prolog System Manual, http://clip.dia.fi.upm.es/Software/Ciao/ciao_html/ciao_toc.html

[18] The Open Source Software Promotion Center, http://www.oss.or.kr

[19] The Open Source Definition, http://opensource.org/docs/definition.php

[20] V. Sperschneider and G. Antoniou, "Logic: A Foundation for Computer Science", Addison-wesley, 1991.

신 동 하

e-mail : dshin@smu.ac.kr
1980년 경북대학교 전자계산학과(학사)
1982년 서울대학교 전자계산기공학과(석사)
1994년 University of South Carolina, Dept of Computer Science(박사)
1982년~1996년 한국전자통신연구원 책임연구원
1997년~현재 상명대학교 소프트웨어학부 부교수
관심분야 : 프로그래밍 언어, 컴파일러, 임베디드 컴퓨팅, 인공지능 등

손 성 훈

e-mail : shson@smu.ac.kr
1991년 서울대학교 계산통계학과(학사)
1993년 서울대학교 전산과학과(석사)
1999년 서울대학교 전산과학과(박사)
1999년~2004년 한국전자통신연구원 선임연구원
2004년~현재 상명대학교 소프트웨어학부 전임강사
관심분야 : 멀티미디어, 시스템소프트웨어, 임베디드 컴퓨팅 등