Journal of Southeast University (English Edition) Vol. 22, No. 3, pp. 357 – 360 Sept. 2006 ISSN 1003—7985

Method of creating ontologies for Prolog clauses

Jiang Zhihua[1,2] Jiang Yunfei[1]

([1]School of Information Science and Technology, Sun Yat-Sen University, Guangzhou 510275, China)

([2]Department of Computer Science, Jinan University, Guangzhou 510632, China)

Abstract: A method is proposed to build an ontology in the form of a conceptual graph for Prolog clauses, so that the Prolog reasoning engine can differentiate clauses semantically to some degree. The concept model of a clauses ontology is composed of predicate parameters and head sub-goals, and these sub-goals appear in the head of the rule body and precede any predicate calls. In the proposed method, a Prolog program can be transformed into a Prolog + CG program that includes a clauses ontology. Some experiments show that, with a clauses ontology, some clauses which do not match current goals are not considered early enough, and, therefore, the size of the goal solution tree may be reduced. By the capability of conceptual graph, a clauses ontology makes the semantics of the Prolog program more clear and readable, and sometimes it speeds up the solution process obviously.

Key words: ontology; clause; conceptual graph; Prolog + CG

An ontology is a concept derived from western philosophy, where it means the theory about being and its disciplines. In computer science, an ontology is defined as clear and formal specifications of shared conceptual models[1]. In the past two decades, many computer researchers have dedicated much effort to exploration in this field, such as presenting a variety of formal ontology languages, and developing primary tools used to build and comment ontologies. Ontology methodology has been applied understanding natural languages, information retrieval and integration, virtual enterprises, software requirements acquirement and so on[2-3]. Moreover, an ontology is an explicit definition of conceptualization, and concept models are involved into almost every domain. So the ontology methodology can be applied in an increasing number of fields.

The reasoning engine in a Prolog environment can reach all the solutions via a trace-back mechanism, but the goal solution tree (GST) grows exponentially as programs become more complex. It is for this reason that all subgoals in a rule body must join the GST once the condition in the rule head is satisfied in a rule clause. Typically, the more clauses a program contains, the more time is spent in building the GST. On the other hand, there often exist implicit semantic differences among different clauses in a Prolog program, for each clause is used for some specific purpose. Therefore, we may describe the internal semantic information formally by building conceptual models for clauses which can be directly processed by computers.

1 Defining Ontology for Prolog Clauses

Clauses in a Prolog program have two forms: fact clauses and rule clauses. The difference between the two kinds of clauses is: when a goal node is matched by a fact clause, the goal tree will not be expanded further; when a rule clause is matched, its body will join the goal tree as new leaf nodes and the goal tree is expanded. Therefore, to minimize the size of the goal tree, we only need to build ontologies for rule clauses. Among a variety of methods for ontology building[4-5], we use the conceptual graph[6] (CG) to express an ontology for Prolog clauses.

Secondly, what is the ontology of a clause? As mentioned above, there exist connotative semantic differences among different rules. In the head of a rule clause, all the parameters of the predicate can be used to distinguish among clauses, because we compare two predicates by means of unification. Besides, in the body of a rule clause, some relation expressions can also be used to differentiate clauses. All of the variables involved in a relation expression must be bound in constants first, so the relation expression is determinable. However, if we meet a predicate call in the body of a rule, we do not know actually whether the call is successful or not. The result of any predicate call is uncertain and we cannot use it to differentiate clauses determinately and clearly. Therefore, we can take advantages of parameters and some relation expressions to build

Received 2006-04-25.

Biographies: Jiang Zhihua (1978—), female, graduate, lecturer, jnujzh@163.com; Jiang Yunfei (1945—), male, professor, lncsri05@zsu.edu.cn.

ontology for clauses. Then, let us give the definitions of some important concepts and ontology of clauses.

Definition 1 Head sub-goals (HSGs) are a sequence of sub-goals in a rule body which starts from the head of the rule body and ends before the first predicate call.

Head sub goals can contain relation expressions, assignment expressions for free variables, any output expressions and so on, but never a predicate call. For instance, if we have a predicate "pp1" defined as "pp1(X, Y, Z): $-X>5, Z=3, Y+Z>6$, pp2(X, Y, Z), $Y>3$, pp3(X, Z)." its head sub goals consist of "$X>5, Z=3, Y+Z>6$". And "$Y>3$" does not belong to this sequence for the reason that it follows the predicate call "pp2". Typically, HSGs reflect the situation of carrying out a rule to some degree.

Definition 2 The ontology of a rule clause is a concept graph composed of a parameter list in a rule head and the HSGs of the rule body.

Generally speaking, for every rule clause such as "pp(pa_1, pa_2, …, pa_n): -HSGs, others", where "others" stands for the rest sub goals when HSGs are removed from the rule body, the conceptual model is depicted in Fig. 1. In this model, there are three kinds of concept nodes: First, the predicate name "pp" can be transformed into a concept node "[pp]"; secondly, each parameter "pa_i" ($1 \leqslant i \leqslant n$) of "pp" can be transformed into a concept node "[pa_i]"; thirdly, assume that HSGs is composed of "sg_1, sg_2, …, sg_k", then each sub goal "sg_j" ($1 \leqslant j \leqslant k$) can be transformed into a concept node "[sg_j]". And we can also use a relation node "[para]" to connect "[pp]" and "[pa_i]", and a relation node "[hsg]" to connect "[pp]" and "[sg_j]".

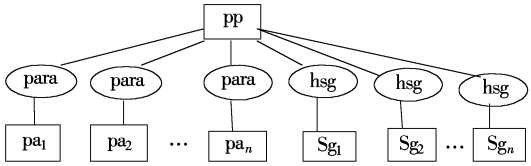


Fig. 1 Conceptual model for ontology of a rule clause

2 Algorithm of Clause Ontology Building

When a Prolog program is being processed, every clause which may match the current goal must be sent to a stack. The clause located at the top of the stack will be used to expand the goal tree, and the rest (if any exist) is set a location pointer for tracing back in the future. The prolog engine judges whether or not to match only on the basis of the syntax level of a program, such as if the predicates can be unified. However, this is not the end of the story. We should distinguish among clauses by some degree of semantic level by building an ontology. Our aim is not to change the internal trace-back mechanism, but to make the goal tree expand as little as possible.

2.1 Introduction to algorithm "clause ontology building"

We define the clause ontology as a concept graph composed of a parameters list in the rule head and the HSGs of the rule body, so, naturally, the algorithm "clause ontology building" is actually to transfer a Prolog program into a Prolog + CG program. Prolog + CG[7] is a tool used to implement a concept graph in a Prolog programming environment. The algorithm is depicted as follows:

Step 1 Build ontology for each clause separately by translating its related parts into a conceptual graph, and add them into the original program. For a rule clause shaped as "pp(pa_1, pa_2, …, pa_n): -HSGs, others", we obtain its ontology framework as follows:

pp(pa_1, pa_2, …, pa_n): : [pp] -caseof→
[npp]: -HSGs, npp(pa_1, pa_2, …, pa_n)

Step 2 Replace the old clauses "pp" with the new ones "npp", which means the clause "pp(pa_1, pa_2, …, pa_n): -HSGs, others" is changed into "npp (pa_1, pa_2, …, pa_n): -others".

Let us present some explanations for the clause concept graph. "pp(pa_1, pa_2, …, pa_n)" is a predicate object, with predicate name "pp" and parameter list "pa_1, pa_2, …, pa_n". The symbol ": :" stands for an object relation, which connects an object and its rule. In the rule, "[pp] -caseof→ [npp]" means concept relation, which becomes true only when the rule body is satisfied. And the new predicate "npp" can be used to transmit variables, which makes those not-ontology elements obtain the values which have been bound into these variables. The clause concept graph created in step 1 is called the ontology for clauses since it is helpful in differentiating clauses into some degree of semantics. Moreover if a concept relation is false, the not-ontology elements of the clause do not need to join into the goal tree, and thus the scale of the goal tree is reduced.

2.2 Example for applying the algorithm of "clause ontology building"

We have a Prolog program, which is transferred into a Prolog + CG program containing clause ontology by applying the algorithm "clause ontology building". These two programs are compared in Tab. 1, with their query and answer at the same time. Here, para1, para2 and para3 are new predicate names which are produced randomly. "sup(y, 4), inf(y, 2), dif(x, 1), dif(x, 2)" are

relation expressions: $\sup(y, 4)$ is true when y is more than 4; $\inf(y, 2)$ is true when y is less than 2; $\mathrm{dif}(x, 1)$ is true if x cannot be unified with 1. Here, "val" is assumed to be a predicate call.

Tab. 1 Transferring a Prolog program into a Prolog + CG program

Program types	Program	Query	Answer
Prolog	action(1, y): – sup(y, 4), val(y1, add(1, y)), write(y1); action(2, y): – inf(y, 2), val(y1, add(2, y)), write(y1); action(x, y): – dif(x, 1), dif(x, 2), val(y1, add(x, y)), write("x is not 1, 2; it is"), write(x), write(y1).	? – action(2, 1).	3 {}
Prolog + CG *	Universal > action, para1, para2, para3; action(1, y): : [action] – caseof→[para1]: – sup(y, 4), para1(1, y); action(2, y): : [action] – caseof→[para2]: – inf(y, 2), para2(2, y); action(x, y): : [action] – caseof→[para3]: – dif(x, 1), dif(x, 2), para3(x, y). para1(1, y): – val(y1, add(1, y)), write(y1); para2(2, y): – val(y1, add(2, y)), write(y1); para3(x, y): – val(y1, add(x, y)), write("x is not 1, 2; it is"), write(x), write(y1). run: – action(2, 1): : G.	? – run.	3 {}

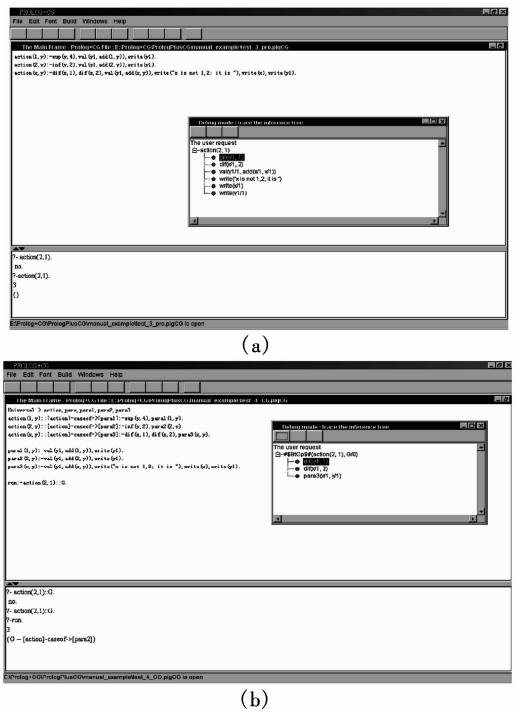
Note: * You may notice that we add a clause "run" in the second program, for the purpose of avoiding outputting the value of variable G (G = [action]-caseof→[para2]) in "action(2, 1): : G".

It is clear that the output of the two programs is the same. Actually, they are equivalent because the transformation depicted above will not change the original meaning of the program. The new program is added with some clause concept graphs which may make the program longer, but make the semantics of clauses clearer and sometimes greatly reduce the size of goal tree.

3 Experimental Analyses

In the process of solving a program, we can also use the "debug" window to display the goal tree at any given time. In Fig. 2, the scenario in the two windows is the same: When the program finds a solution successfully by matching the goal "action(2, 1)" with the second clause, it begins to search the next possible solution by trying the third clause through back-trace. We can see that the number of nodes expanded in the second window is less than that of those in the first window in this specific scenario. In Prolog + CG environment, the "edit" window displays a Prolog source program, the "output" window illustrates a query and answer, and the "debug" window shows the goal tree at any time.

Here, we have a conclusion about how clause ontology can influence the scale of the goal tree. Suppose that in a rule clause, the number of sub-goals in the rule body is N_{sub}, among which the number of sub-goals belonging to HSGs is N_{hsg}. When a concept relation is selected, an additional node is added to the current goal tree (because of the new predicate "npp"). However, if the concept relation is not true (for maybe some HSG is not satisfied), there are $N_{\mathrm{sub}} - (1 + N_{\mathrm{hsg}})$ nodes which do not need to join the goal tree (Generally speaking, N_{sub} is much greater than N_{hsg}). For example, in Fig. 2, N_{sub} is six and N_{hsg} is two, so Fig. 2(a) expands three nodes more than Fig. 2(b).



(a)

(b)

Fig. 2 The goal tree in the debug window. (a) The old program; (b) The new program

4 Conclusion

In this paper, we give a definition of ontology for Prolog clauses in the form of a concept graph. And we also present an algorithm of "clause ontology building" to create ontologies for clauses in the Prolog + CG environment. The clause ontology may make a program longer, but it makes the semantics of clauses clearer and sometimes greatly reduces the size of goal tree. At the same time, the trace-back mechanism has

undergone little change, because the trace-back for objects is built into clause ontology, not the clause itself. Another function is to enrich the definition of the clause ontology and describe it in other formal ontology languages and integrate into the Prolog engine. Anyway, we think that our work is significant and creative, and we hope that this paper will have a positive effect on research and applications of ontology methodology.

References

[1] Li Shanping, Hu Yujin. Overview of researches on ontology [J]. *Journal of Computer Research and Development*, 2004, **41**(7): 1041 - 1052. (in Chinese)

[2] Liu Jin, He Keqing, Li Bing. Researches on logical semantic analyses of web ontology languages [J]. *Computer Engineering*, 2005, **31**(9): 7 - 10. (in Chinese)

[3] Uschold M, Gruninger M. Ontologies: principles, methods, and applications [J]. *Knowledge Engineering Review*, 1996, **11**(2): 93 - 155.

[4] Fernandez M. Overview and analysis of methodologies for building ontologies [J]. *Knowledge Engineering Review*, 2002, **17**(2): 129 - 156.

[5] Bisson G, Nedellec C, Canamero L. Designing clustering methods for ontology building[A]. In: *Proceedings of the ECAI, Ontology Learning Workshop*[C]. Berlin, Germany, 2000. 13 - 19.

[6] Fargues J, Landau M C. Conceptual graphs for semantics and knowledge processing [J]. *IBM Journal of Research and Development*, 1986, **30**(1): 70 - 79.

[7] Kabbaj A, Janta-Polczynski M. From PROLOG + + to PROLOG + CG: a CG object-oriented logic programming language[A]. In: Ganter B, Mineau G W, eds. *Proceedings of ICCS'00, LNAI*[C]. Springer, 2000, **1867**: 540 - 554.

一种构造 Prolog 程序子句本体的方法

蒋志华[1,2]　　姜云飞[1]

([1] 中山大学信息科学与技术学院, 广州 510275)
([2] 暨南大学计算机系, 广州 510632)

摘要: 为了使 Prolog 推理引擎可以从一定程度的语义上来区分子句,通过概念图的形式对 Prolog 程序的子句建立本体. 子句本体的概念模型通过其谓词参数和头子目标来建立,而头子目标是指出现在规则体首部并且位于任何谓词调用之前的子目标集合. 所提出的方法把一个 Prolog 程序转换成包含其子句本体的 Prolog + CG 程序. 实验表明,通过对 Prolog 子句建立本体,可以尽早地去除与当前目标明显不匹配的子句,从而使得目标求解树的规模减小. 在概念图的描述形式下,子句本体使得 Prolog 程序的语义更清晰可读,在某些情况下能明显加快程序的求解过程.

关键词: 本体;子句;概念图;Prolog + CG

中图分类号: TP18