Annotated Logic Programmingにおける矛盾の処理について

1B−6

合志和晃, 程京徳, 牛島和夫
九州大学 工学部

1. 背景

複数のエキスパートから情報を得てつくった知識ベースは矛盾した情報を含むことがある。したがって、論理プログラミングを知識ベースの開発ツールにする場合は、矛盾を扱う方法を持つ必要がある。そのためにBlairとSubrahmanianはAnnotated Logic Programming(ALP)を提案した[1]。ALPでは注釈(Annotation)で矛盾の記述を可能にしている。一般には注釈に多値の値が入るが、ここでは四値とし、注釈としてはt,f,both,noneを考える。それぞれ、tは真(true)、fは偽(false)、bothは矛盾(inconsistent)、noneは未知(unknown)を表わす。例えば、あるリテラル Aを真と考えるならA:tで、偽と考えるならA:fで、矛盾と考えるならA:bothで表わす。構文については、Prologで使われているホーン節でそれぞれのリテラルに注釈を付けたものについて考える。論理プログラミングシステムとしての動作はPrologに準ずるもの(マッチングを行い単一化代入(unify)できれば成功)とし、さらに、A:tとA:fが成功するならばA:bothも成功するといった真と偽があれば矛盾とするclone[2]という手続きも考慮する。

定義1 Aがリテラルなら、
A:μは注釈つきリテラル。 μはAの注釈で四値論理の真理値の集合の要素が入る。

定義2 $L_0,...,L_n$が注釈つきリテラルなら
L_0 :- $L_1,L_2,...,L_n$は一般ホーン節(generalized Horn Clause:gh-Clause)である。

定義3 一般ホーン節のうち、
L_0 :- .(または L_0.)の形をしたものを事実(fact)、
L_0 :- $L_1,L_2,...,L_n$.の形をしたものを規則(rule)、
:- $L_1,L_2,...,L_n$(または?-$L_1,L_2,...L_n$.)の形をしたものを質問(query)、と呼ぶ。

定義4 ALPのLogic Programは一般ホーン節の有限集合である。

例 Prologリテラル　　a　　a(b)
　　節(clause) ?-a.　b.　c :- d.
　ALP　注釈　　a:t　a(b):f　a:both
　　gh-Clause　?-a:t. b:t. c:t :- d:t.

2. 問題点

ALPには、ユーザがある一つのリテラルの注釈を知りたい場合に複数の質問をしなければその注釈が得られないという問題がある。例えばあるリテラルxについて、ユーザはx:tという質問だけでは十分な情報を得られない。なぜならx:tについて質問しただけでは、x:fが成功するかどうかわからないためである。よって十分な情報を得るにはさらにx:fについても質問しなければならない。このように、ALPでは十分な情報を得るには複数の質問が必要である。このことは、ユーザに余分な手間を与えるだけでなく、ユーザが十分に質問をしなかった場合は誤った情報を得てしまうため問題である。

3. 解決案

複数の質問が必要な原因は、真と偽の関係を無視しているためである。もし矛盾の存在を許さないなら、真ならば偽はありえないので、一方が分かれば他方も分かるが、ここでは矛盾の存在を許すので、真であるからといってそれが偽ではないとは限らない。つまり問題を解決するには、システムが真と偽が共に存在する可能性に注意する必要、つまり調べる場合には常に両方調べる必要がある。例えば質問 ?-x:tに対してシステムは常にx:tとx:fについて成功するかどうか調べ、x:tのみ成功すればxは真であり、x:fのみ成功すればxは偽であり、x:t,x:fともに成功すればxは矛盾であり、x:t,x:fともに失敗すればxはunknownであると答える。

Improving the Processing of Inconsistency in Annotated Logic Programming
Kazuaki Goshi, Jingde Cheng, and Kazuo Ushijima
Faculty of Engineering, Kyushu University
6-10-1 Hakozaki, Higasiku, Fukuoka, 812, Japan

このシステムについて注意する必要があるのは、リテラルの引数に変数を含む場合である。この場合は、真と偽のそれぞれ同じ変数は同じ代入をされたものについて調べなくてはならない。つまり、真について質問された場合は、真のものを探し存在すれば変数に注釈が代入される。次に偽も探す際に、その同じ代入をした偽のものについて探す必要がある。

例 真と偽を両方調べる手続き（大文字の X は変数）

a(X):t を捜す場合

1: もし a(X):t. が 成功 X=x1 ならば
　もし a(x1):f. が 失敗 ならば
　（ここで偽を調べる際に、a(X):f に
　ついてではなく a(x1):f について調べる）
　a(X) は X=x1 で真 (t)
　もし a(x1):f. が 成功 X=x1 ならば
　X=x1 のつぎの X について 1 へ戻る
もし a(X):t. が 失敗 ならば
　a(X):t は無い

4. さらに便利にするための提案

上記の真と偽を両方調べるという方法でユーザは一度の質問で、質問しているリテラルの注釈がわかる。ここでは、このシステムをユーザにとってさらに便利にする方法を考える。質問をして、リテラルの注釈が矛盾 (both) と出た場合に、なんらかの基準で矛盾の真と偽のどちらか一方を正しそうだと助言出来るなら、ユーザがその矛盾について考える助けとなる。この基準については、関連するものに矛盾があるかどうか、ということを考えている。例えば、あるリテラル x について x:t と x:f がともに成功する場合に、もし x:t に関連するものに矛盾があり、x:f に関連するものに矛盾がなければ、x:t はあやしいとし、x:f を正しそうだとする。このように、ユーザの判断を助けるために矛盾の存在と共にシステムが判断したどちらが正しそうだという情報を提供し、それが無理な場合は、矛盾の存在のみ知らせる、という矛盾の処理方法を提案する。今回の目的は、複数の質問が必要であるという問題の解決であるので、この方法の細部についてはふれない。

5. 課題

今後の課題は以下の通り。

NF の扱い

ここでは NF(Nagation as Failure: 失敗としての否定：探して失敗した場合は、偽とする) の扱いについてはふれていない。しかし記述の面で全ての偽を明示することは不可能であるので必要であろう。NF を取り入れる場合は、注釈で明示してある f と NF による f の扱いを区別するかしないか、という課題がある。

関連する矛盾による矛盾の処理

関連の実現方法について、なにを関連する矛盾とするか決める必要がある。一つの方法としては、その真なり偽なりを導出する途中にあらわれた矛盾を指すようにするというものがある。また、導出をするのに用いた節と同じ属性（例えばリテラルの名や引数の定数）をもつ矛盾を指すようにするという方法も考えられる。真にも偽にも関連する矛盾がある場合の扱いについては、真偽両方に矛盾が現れたら、判断不能であるとする方法や、関連する矛盾になんらかの差をつけてどちらかを選ぶ方法が考えられる。

演繹途中の矛盾の扱い

ここで提案している矛盾の処理はユーザがシステムに質問をする部分での矛盾の処理であり、演繹途中の矛盾についてではない。ALP では、例えば x:t. x:f. y:t :- x:t. があるとき y:t を導出する途中の x の矛盾にかかわらず y:t は成功する。これでは途中の矛盾の処理は不可能である。もし処理すれば ALP の fact や rule は元の意味から外れてしまう。しかし、無理にでも矛盾の処理を演繹途中にも適用したらその ALP から外れたものの意味はどうなるのかということを考えるのは興味深い。

参考文献

[1] H.A.Blair and V.S.Subrahmanian, "Paraconsistent Logic Programming," Theoretical Computer Science 68,North-Holland, pp.135-154, 1989.

[2] James J.Lu, Lawrence J.Henshen, V.S.Subrahmanian, Newton C.A. da Costa, "Reasoning In Paraconsistent Logics," in R.S.Boyer (ed.), "Automated Reasoning Essays," Kluwer Academic, pp. 181-207,1991.