RC 21473 (96901) 07 May 1999
Computer Science

IBM Research Report

DIPLOMAT: Compiling Prioritized Default Rules into Ordinary Logic Programs, for E-Commerce Applications (extended abstract of Intelligent Systems Demonstration)

Benjamin N. Grosof

IBM Research Division
T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598, USA
(914) 784-7783 direct -7455 fax -7100 main
Internet e-mail: grosof@us.ibm.com (alt. grosof@cs.stanford.edu)
Web: http://www.research.ibm.com/people/g/grosof

IBM Research Division
Almaden · Austin · Beijing · Delhi · Haifa · T.J. Watson · Tokyo · Zurich

Abstract:

Rules promise to be widely useful in Internet electronic commerce. Declarative prioritized default rule knowledge representations offer the advantage of handling conflicts that arise in updating rule sets, but have as yet had little practical deployment. DIPLOMAT is a Java library that embodies a new approach to the implementation of such prioritized default rules: to compile them into ordinary logic programs (with forward or backward inferencing, not simply Prolog). We apply the approach to a newly generalized version of courteous logic programs (LP's), a semantically attractive and computationally tractable form of prioritized default rules. Compilation enables courteous LP's functionality to be added modularly to ordinary LP rule engines, via a pre-processor, with tractable computational overhead. This takes a long step towards actual deployment of prioritized-default knowledge representation in commercially fielded technology and applications.

We give in the demo storyboard (appendix) an automated example e-commerce application scenario: inferencing in a 60-rule courteous LP that represents personalized pricing and promotions on a bookstore's Web storefront.

This paper is an extended version of [5]. It differs from that by including some extra references and material, notably the demo storyboard. Further technical details can be found in [4].

Related Papers and Material: can be found via the author's Web address and as IBM Research Reports at http://www.research.ibm.com .

Keywords: logic programming, default reasoning, priorities, non-monotonic reasoning, intelligent agents, rules, e-commerce, electronic commerce, compilation, knowledge compilation. More keywords: applications, modularity, tractability, negation-as-failure, Prolog, updating, belief revision, artificial intelligence, machine learning, deductive databases.

1 Rules in E-Commerce

Rules[1] promise to be widely useful in Internet electronic commerce as an automatically executable specification language / programming mechanism. Rules are useful, for example, to represent: seller offerings of products and services, buyer requirements, contractual agreements, authorization policies, and more generally, many e-business policies and processes. A characteristic of this realm is that, often, rules need to be modified frequently and by multiple players. Conflicts between rules often arise during updating and merging.

In e-commerce applications, rules are specified by business-domain experts such as marketing managers and are modified frequently, including by merging rules specified by different people in different organizations.

Another characteristic of this realm is that it is desirable for the rules knowledge representation (KR) to be highly declarative: to have a semantics, independent of inferencing algorithm details, that specifies which set of conclusions are entailed by a given set of premise rules. Declarativeness aids exchange of rules between heterogeneous applications or enterprises, modification of rules, understandability of rules by humans.

2 Challenge of Prioritized Default Rules

We are attracted by some virtues of prioritized default rules (in declarative KR): they handle conflicts, including arising during updating of rule sets, using partially-ordered prioritization information that is naturally available based on relative specificity, recency, and authority.

In Internet e-commerce, the prioritized default expressive features are valuable especially because they greatly facilitate incremental specification, by often eliminating the need to explicitly modify previous rules when updating or merging.

The **overall problem we address** is: how to enable prioritized default rules to be used as a widely practical knowledge representation for specification and execution of rule-based software, especially in e-commerce.

Prioritized default rules are of long-standing interest in the knowledge representation (KR) community, and have received much study. However, they have as yet had little impact on practical rule-based systems and software engineering generally, and had very few deployed serious applications.

One difficulty is getting the semantics right, including intuitively simple enough that non-experts in KR can feel comfortable specifying, and often repeatedly modifying, rule sets. Another difficulty is the complexity of implementing inferencing in a new KR. A third difficulty is facilitating a transition, which is best made incrementally, by builders and users of previous rule-based technology, to a new representation.

We take a new overall approach to remedy these three difficulties, especially the third.

3 DIPLOMAT, Compilation Approach, Generalized Courteous Logic Programs

The DIPLOMAT system is a Java[2] library that embodies our new approach [4] to the implementation of prioritized default rules: to **compile** them into ordinary logic programs (OLP's). ("Logic

[1](in the sense of knowledge representation and rule-based systems)

[2]trademark of Sun Microsystems

program" here means in the sense of declarative knowledge representation [1], not just Prolog). DIPLOMAT currently implements such compiling for one particular prioritized default rule KR: **courteous logic programs**. The compiler is run as a pre-processor, then its output is fed to an OLP inferencing engine, which may be forward-chaining or backward-chaining.

Besides the compiler, DIPLOMAT includes the **interlingua** capability [6] to translate ordinary LP's (e.g., the results of its compilation) to and fro multiple other pre-existing ordinary-LP inferencing engines, e.g., Prolog systems and intelligent agent systems. DIPLOMAT currently implements such translating for 3 such OLP engines. It also currently implements such translating of OLP to and fro Knowledge Interchange Format (KIF)[3], an ANSI draft standard for interchange between knowledge-based systems, whose semantics is based primarily on first-order logic. DIPLOMAT further currently defines and implements an XML format for OLP's, courteous LP's, and the expressive fragment of KIF that overlaps with these (i.e., LP's with classical negation but without negation-as-failure).

As a target KR, OLP's are very attractive. They are computationally tractable[4], unlike even the propositional case of classical logic, yet represent basic non-monotonicity via the negation-as-failure expressive feature. They are in widespread deployment and application, including by many programmers who know and care very little about KR generally. There are a number of highly efficient and sophisticated OLP rule systems / inferencing engines available. OLP's are also closely related to derived relations in SQL relational databases, and to several other varieties of rule-based systems.

Courteous logic programs [3] are our favorite previous prioritized default rule KR. Courteous LP's include ordinary LP's as a special case but further feature classical negation and prioritized conflict handling. Courteous LP's have a number of attractive properties, detailed at more length in [2]. They have a unique consistent conclusion set, and are computationally tractable[5], with relatively modest extra computational cost compared to OLP's. Their behavior captures many examples of prioritized default reasoning in a graceful, concise, and intuitive manner. They have a number of established well-behavior properties, including under merging.

DIPLOMAT implements a version of courteous LP's that is expressively generalized in several aspects from the previous version of courteous LP's in [3] but retains the previous version's above attractive properties. First, it enables recursion, i.e., cyclic dependence of a predicate through rules upon itself. Second, it enables reasoning about the prioritization, i.e., inferencing to conclude that one rule has higher priority than another. These first two generalization aspects are described in [4]. Third, it enables the scope of conflict to be specified not just in terms of classical negation (a proposition p versus its classical negation), but more generally as a set of **pairwise mutual exclusion constraints** (mutex's). It then enforces these mutex's and ensures consistency with respect to them. Mutex's are practically extremely useful even for otherwise relatively expressively simple rule sets, to represent reasoning about k-valued properties for k greater than 2. E.g., in a mail agent, one might wish that a message's urgency level should be inferred to be at most one of 4 levels: emergency, high, medium, or low. This can be specified via mutex's pairwise: e.g., urgency must not be both high and medium, nor both high and low, etc..

[3] `http://logic.stanford.edu/kif/` and http://www.cs.umbc.edu/kif/

[4] Inferencing is worst-case polynomial-time for the propositional case; or, more generally, given the commonly-met restrictions of (1) Datalog (no logical functions with non-zero arity) and (2) bounded number of logical variables appearing per rule.

[5] under the same restrictions mentioned earlier

4 Demo Storyboard

We give in the demo storyboard (see Appendix A) an automated example e-commerce application scenario: inferencing in a courteous LP that represents personalized pricing and promotions on a bookstore's Web storefront. The example courteous LP contains about 60 rules and facts. Conclusions are drawn about what price discounts to offer and what targeted ads to show, for a given shopper.

Here, rules are specified by business-domain experts such as marketing managers and are modified frequently, including by merging rules specified by different people in different organizations. The prioritized conflict handling facilitates representing a sequence of such modifications as a simple accumulation of rules (and facts) — without necessitating explicit revision of previous rules.

The demo storyboard illustrates running the compiler (then the interlingua) then using XSB, a pre-existing OLP engine, to perform backward inferencing on the post-compilation OLP. XSB is a Prolog-like system implemented in C, developed by David Warren and his group at SUNY Stonybrook, http://www.cs.sunysb.edu/~sbprolog .

Acknowledgements

Hoi Chan, Michael Travers, and Xiaocheng Luan have also contributed to the DIPLOMAT implementation, especially the interlingua. Manoj Kumar contributed early concept work to the storyboard example.

References

[1] Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148, 1994. Includes extensive review of literature.

[2] Benjamin N. Grosof. Courteous logic programs: Prioritized conflict handling for rules. Technical report, IBM T.J. Watson Research Center, http://www.research.ibm.com , search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, Dec. 1997. IBM Research Report RC 20836. This is an extended version of [3].

[3] Benjamin N. Grosof. Prioritized conflict handling for logic programs. In Jan Maluszynski, editor, *Logic Programming: Proceedings of the International Symposium (ILPS-97)*, pages 197–211, Cambridge, MA, USA, 1997. MIT Press. Held Port Jefferson, NY, USA, Oct. 12-17, 1997. http://www.ida.liu.se/~ilps97. Extended version available as IBM Research Report RC 20836 at http://www.research.ibm.com .

[4] Benjamin N. Grosof. Compiling Prioritized Default Rules Into Ordinary Logic Programs. Technical report, IBM T.J. Watson Research Center, http://www.research.ibm.com , search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598. USA, May 1999. IBM Reserach Report RC 21472.

[5] Benjamin N. Grosof. DIPLOMAT: Compiling Prioritized Default Rules Into Ordinary Logic Programs, for E-Commerce Applications (extended abstract of Intelligent Systems Demonstration). In *Proceedings of AAAI-99*, San Francisco, CA, USA, 1999. Morgan Kaufmann. Extended version available in May 1999 as an IBM Research Report RC21473, http://www.research.ibm.com, search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, USA.

[6] Benjamin N. Grosof and Yannis Labrou. An Approach to using XML and a Rule-based Content Language with an Agent Communication Language. In *Proceedings of the IJCAI-99 Workshop on Agent Communication Languages*, 1999. Held in conjunction with the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99) http://www.ijcai.org . Extended version available in May 1999 as IBM Research Report, http://www.research.ibm.com, search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, USA.

Appendix A: Demo Storyboard

```
Storyboard:  DIPLOMAT Example:  Bookstore Web E-Storefront

------------------------------------------------------------------------------
slide 1  More Overview of the Application Scenario and Storyboard.


o Business-to-Consumer personalized promotions:
  -  discounting:  per-person, e.g., 0, 5, or 10 percent price discount.
  -  showing targeted ads with incentives;
       an incentive means a specific-product price discount, per-person.

o Rules & facts from:
  -  several marketing managers:  with updates & merges
  -  priorities from recency, authority, specificity
  -  data mining
  -  relational database
  -  dynamic Web session data
------------------------------------------------------------------------------
```

---

slide 2 Background about Syntax:

DIPLOMAT's ASCII syntax for Courteous LP's:

A rule has the form "head <- body ."
"<-" is the logical-implication connective, i.e., IF.
"." (i.e., a period) indicates the end of a rule.

Rules optionally have labels; these are handles to specify prioritization.
A rule label precedes the (unlabelled part of the) rule, and
is enclosed in angle brackets: "< ... >"   .

A fact is just a special case of a rule: one in which the head is ground
(i.e., fully instantiated) and the body is empty (semantically, the body
is regarded as always True).
If a rule's body is empty, the   <-   connective may optionally be omitted.

"overrides" is the reserved prioritization predicate, it takes a pair of
rule labels as arguments. "overrides(lab1,lab2)" indicates that a
rule labelled by "lab1" has higher priority than rule labelled by "lab2".

"overrides" is otherwise an ordinary predicate symbol,
and rule labels are otherwise ordinary 0-ary function symbols; both are
part of the ordinary logical language.

"?" prefixes a logical variable.

"AND" means logical conjunction.

"CNEG" means classical negation. Intuitively, CNEG p means the
holder of the rule/fact believes/knows that p is false.

"FNEG" means negation-as-failure. FNEG p represents a
weaker belief than CNEG p. Intuitively, FNEG p means the
holder of the rule/fact does not believe/know p is true.

"AND", "FNEG", "CNEG"   are all case-insensitive.

White space does not matter.

XSB syntax (which is in ASCII) for ordinary LP's:

":-" means implication, i.e., IF.
"," means conjunction, i.e., AND.
Logical variables must start with an upper-case alphabetic character.
"tnot" means negation-as-failure.

```
-----------------------------------------------------------------------------
!!!The demo proper, i.e., mechanized stuff, starts here
-----------------------------------------------------------------------------
slide 3:  some rules (a courteous LP in DIPLOMAT's ASCII syntax):

giveDiscount(zeroPercent,?Cust) <- shopper(?Cust) and ordinaryCustomer(?Cust).
giveDiscount(fivePercent,?Cust) <- shopper(?Cust) and loyalCustomer(?Cust).
giveDiscount(tenPercent,?Cust) <- shopper(?Cust) and premiereCustomer(?Cust).
MUTEX_HEAD <- premiereCustomer(?Cust) and loyalCustomer(?Cust).
MUTEX_HEAD <- premiereCustomer(?Cust) and ordinaryCustomer(?Cust).
MUTEX_HEAD <- loyalCustomer(?Cust) and ordinaryCustomer(?Cust).
<presumeOrdinary> ordinaryCustomer(?Cust).
overrides(?Any,presumeOrdinary).
<steadySpender> loyalCustomer(?Cust) <- loyalPurchaser(?Cust).
about(loyalty,steadySpender).
<bigSpender> premiereCustomer(?Cust) <-
purchaseHistoryFrequency(?Cust , "$10000" , anytimes , last5years).
overrides(bigSpender,?Any).
<slowPayer> ordinaryCustomer(?Cust) <- slowToPay(?Cust,last1year).
overrides(slowPayer, ?LoyaltyRule) <- about(loyalty,?LoyaltyRule).
<storeCard> loyalCustomer(?Cust) <- hasChargeCard(?Cust,store).
about(loyalty,storeCard).
<platinumClub> premiereCustomer(?Cust) <- memberPlatinumClub(?Cust).
overrides(platinumClub,?Any).
loyalPurchaser(?Cust) <- purchaseHistoryFrequency(?Cust,"$100",10,last1year).
loyalPurchaser(?Cust) <-purchaseHistoryGenreSpread(?Cust,"$100",5, last2years).
loyalPurchaser(?Cust) <- regressionPredicts(?Cust, "$40" , monthlyPurchases).
promote(dilbert , ?Cust) <- shopper(?Cust) and likes(humor , ?Cust).
likes(humor,?Cust) <- purchaseHistoryGenre(?Cust ,"$20",humor,last2years).
promote(?Product , ?Cust) <- currentMonth(december) and shopper(?Cust)
and calendar(?Product) and buyingGifts(?Cust).
buyingGifts(?Cust) <- gift(?Product) and inShoppingCart(?Cust, ?Product).
buyingGifts(?Cust) <-  purchased(?Cust,?Purchase,last30days)
and shipped(?Purchase,?Address) and CNEG homeAddress(?Cust, ?Address).
<goAd> showAd(?Product,?Cust) <- promote(?Product,?Cust)
                                              and seenBy(?Product, ?Cust).
<goAd> showAd(?Product, ?Cust) <- promote(?Product, ?Cust) and
        belowThreshold(probWillSee(?Product, ?Cust),adProbThreshold(?Product)).
<noNeedAd> CNEG showAd(?Product, ?Cust) <-  bought(?Product, ?Cust).
<closedWorldAd>  CNEG showAd(?Product, ?Cust).
overrides(noNeedAd, goAd).
overrides(goAd, closedWorldAd).
-----------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------

slide 4:  some facts (i.e., more courteous rules in DIPLOMAT ASCII syntax):

calendar(mountainLakes).
currentMonth(december).
gift(bigBookOfTrains).
shopper(ann).
shopper(cal).
shopper(joe).
shopper(kim).
shopper(moe).
shopper(peg).
shopper(sue).
shopper(tim).
shopper(zoe).
purchaseHistoryFrequency(cal,"$100",10,last1year).
purchaseHistoryGenreSpread(joe,"$100",5,last2years).
purchaseHistoryFrequency(joe,"$10000",anytimes,last5years).
regressionPredicts(kim,"$40",monthlyPurchases).
memberPlatinumClub(kim).
hasChargeCard(moe,store).
slowToPay(moe,last1year).
hasChargeCard(peg,store).
slowToPay(peg,last1year).
purchaseHistoryFrequency(peg,"$10000",anytimes,last5years).
inShoppingCart(sue,bigBookOfTrains).
purchaseHistoryGenre(tim,"$20",humor,last2years).
seenBy(dilbert,tim).
purchased(zoe,purchase888,last30days).
shipped(purchase888,address777).
CNEG homeAddress(zoe,address777).
bought(mountainLakes,zoe).
belowThreshold(probWillSee(mountainLakes,zoe),adProbThreshold(mountainLakes)).
-------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
slide 5:  running the (courteous-to-OLP) compiler and interlingua
to generate output rule set in XSB-syntax OLP,
then loading this rule set in XSB rule system,
then querying XSB to generate its inferences.

Conclusions are drawn about what price discounts to offer and what targeted
ads to show, for a given shopper.


F:\grosof\dip\proto\current\examples>
    java TestTransformer < bookstore1.clp
                            > \progra~1\xsb\xsb\grosof\bookstore1.P
F:\Program Files\xsb\xsb\grosof>xsb -i
| ?- [bookstore1].
[bookstore1 compiled, cpu time used: 1.3200 seconds]
| ?- ordinaryCustomer(ann).
yes
| ?- loyalCustomer(cal).
yes
| ?- giveDiscount(fivePercent,cal).
yes
| ?- premiereCustomer(joe).
yes
| ?- ordinaryCustomer(joe).
no
| ?- loyalCustomer(joe).
no
| ?- premiereCustomer(kim).
yes
| ?- ordinaryCustomer(moe).
yes
| ?- giveDiscount(X,peg).
X = tenPercent
yes
| ?- promote(mountainLakes,sue).
yes
| ?- showAd(mountainLakes,sue).
no
| ?- showAd(dilbert,tim).
yes
| ?- n_showAd(dilbert,tim).
no
| ?- showAd(mountainLakes,zoe).
no
| ?- halt.
End XSB
--------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------
slide 6:  the XSB-syntax OLP rule set that is the output of the
(courteous-to-OLP) compiler.  Rules participating in conflict are munged.
Unconflicted rules are unaltered.  (Semantics after transformation are
equivalent:  the same set of conclusions is sanctioned.)

New auxiliary predicates are introduced whose names are prefixed or
suffixed versions of the original predicates.  These auxiliary
predicates appear in the munged rules; intuitively, they represent
intermediate phases of the process of prioritized argumentation in the
courteous semantics.  The prefix "n_", however, corresponds to classical negation.

The source courteous LP (slides 3 and 4) has 62 rules and facts.
The compiled output OLP rule set has 114 rules and facts.
For reasons of focus and readability, we show a selected (and re-sequenced)
subset of the output OLP rules below on this slide;
"..." indicates skipped parts of the OLP.
The entire compiled output OLP is given below after this slide.

giveDiscount(zeroPercent, Cust) :- shopper(Cust), ordinaryCustomer(Cust).
giveDiscount(fivePercent, Cust) :- shopper(Cust), loyalCustomer(Cust).
...
calendar(mountainLakes).
shopper(ann).
...
overrides(noNeedAd, goAd).
overrides(slowPayer, LoyaltyRule) :- about(loyalty, LoyaltyRule).
...
showAd(X1, X2) :- showAd_u(X1, X2), tnot(showAd_o(X1, X2)).
showAd_o(X1, X2) :- n_showAd_u(X1, X2).
showAd_c_24(Product, Cust) :- promote(Product, Cust), seenBy(Product, Cust).
showAd_u(Product, Cust) :- showAd_c_24(Product, Cust),
                           tnot(showAd_r_24(Product, Cust)).
showAd_r_24(X1, X2) :- n_showAd_c_26(X1, X2), overrides(noNeedAd, goAd).
...
n_showAd(X1, X2) :- n_showAd_u(X1, X2), tnot(n_showAd_o(X1, X2)).
n_showAd_o(X1, X2) :- showAd_u(X1, X2).
n_showAd_c_26(Product, Cust) :- bought(Product, Cust).
...
ordinaryCustomer(X1) :- ordinaryCustomer_u(X1), tnot(ordinaryCustomer_o(X1)).
ordinaryCustomer_o(Cust) :- loyalCustomer_u(Cust).
ordinaryCustomer_o(Cust) :- premiereCustomer_u(Cust).
...
loyalCustomer(X1) :- loyalCustomer_u(X1), tnot(loyalCustomer_o(X1)).
loyalCustomer_o(Cust) :- premiereCustomer_u(Cust).
loyalCustomer_o(Cust) :- ordinaryCustomer_u(Cust).
loyalCustomer_c_6(Cust) :- loyalPurchaser(Cust).
loyalCustomer_u(Cust) :- loyalCustomer_c_6(Cust),tnot(loyalCustomer_r_6(Cust)).
loyalCustomer_r_6(Cust) :- premiereCustomer_c_8(Cust),
                                     overrides(bigSpender, steadySpender).
premiereCustomer(X1) :- premiereCustomer_u(X1), tnot(premiereCustomer_o(X1)).
...
-----------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------
slides 7ff.:  The ENTIRE  XSB-syntax OLP rule set that is the output of the
(courteous-to-OLP) compiler.  This includes a preamble of "table" commands
that instruct the XSB system to appropriately store, for use in its own
internal processing, the results of inferencing for the predicates.  These
``table'' commands makes XSB's negation-as-failure processing work correctly
for inferencing.

:- table showAd/2.
:- table premiereCustomer_r_8/1.
:- table premiereCustomer_c_8/1.
:- table loyalCustomer_r_6/1.
:- table calendar/1.
:- table n_showAd_o/2.
:- table premiereCustomer_r_14/1.
:- table premiereCustomer_c_14/1.
:- table giveDiscount/2.
:- table bought/2.
:- table loyalCustomer_o/1.
:- table overrides/2.
:- table showAd_r_25/2.
:- table premiereCustomer_o/1.
:- table premiereCustomer_u/1.
:- table slowToPay/2.
:- table showAd_c_24/2.
:- table hasChargeCard/2.
:- table loyalPurchaser/1.
:- table regressionPredicts/3.
:- table purchaseHistoryGenre/4.
:- table purchased/3.
:- table n_showAd_r_27/2.
:- table ordinaryCustomer_o/1.
:- table ordinaryCustomer_u/1.
:- table n_showAd_c_27/2.
:- table ordinaryCustomer/1.
:- table purchaseHistoryFrequency/4.
:- table promote/2.
:- table ordinaryCustomer_r_4/1.
:- table ordinaryCustomer_c_4/1.
:- table loyalCustomer/1.
:- table purchaseHistoryGenreSpread/4.
:- table ordinaryCustomer_r_10/1.
:- table ordinaryCustomer_c_10/1.
:- table shopper/1.
:- table inShoppingCart/2.
:- table showAd_u/2.
:- table shipped/2.
:- table seenBy/2.
:- table showAd_c_25/2.
:- table gift/1.
:- table premiereCustomer/1.
:- table showAd_r_24/2.
:- table memberPlatinumClub/1.
:- table n_showAd_u/2.
```

```
:- table n_homeAddress/2.
:- table buyingGifts/1.
:- table loyalCustomer_u/1.
:- table n_showAd/2.
:- table n_showAd_r_26/2.
:- table about/2.
:- table belowThreshold/2.
:- table loyalCustomer_c_12/1.
:- table showAd_o/2.
:- table likes/2.
:- table loyalCustomer_c_6/1.
:- table currentMonth/1.
:- table loyalCustomer_r_12/1.
:- table n_showAd_c_26/2.
showAd(X1, X2) :- showAd_u(X1, X2), tnot(showAd_o(X1, X2)).
showAd_o(X1, X2) :- n_showAd_u(X1, X2).
showAd_c_24(Product, Cust) :- promote(Product, Cust), seenBy(Product, Cust).
showAd_u(Product, Cust) :- showAd_c_24(Product, Cust),
                             tnot(showAd_r_24(Product, Cust)).
showAd_r_24(X1, X2) :- n_showAd_c_26(X1, X2), overrides(noNeedAd, goAd).
showAd_r_24(X1, X2) :- n_showAd_c_27(X1, X2), overrides(closedWorldAd, goAd).
showAd_c_25(Product, Cust) :- promote(Product, Cust),
                                belowThreshold(probWillSee(Product, Cust),
                                adProbThreshold(Product)).
showAd_u(Product, Cust) :- showAd_c_25(Product, Cust),
                             tnot(showAd_r_25(Product, Cust)).
showAd_r_25(X1, X2) :- n_showAd_c_26(X1, X2), overrides(noNeedAd, goAd).
showAd_r_25(X1, X2) :- n_showAd_c_27(X1, X2), overrides(closedWorldAd, goAd).
calendar(mountainLakes).
giveDiscount(zeroPercent, Cust) :- shopper(Cust), ordinaryCustomer(Cust).
giveDiscount(fivePercent, Cust) :- shopper(Cust), loyalCustomer(Cust).
giveDiscount(tenPercent, Cust) :- shopper(Cust), premiereCustomer(Cust).
bought(mountainLakes, zoe).
overrides(Any, presumeOrdinary).
overrides(bigSpender, Any).
overrides(slowPayer, LoyaltyRule) :- about(loyalty, LoyaltyRule).
overrides(platinumClub, Any).
overrides(noNeedAd, goAd).
overrides(goAd, closedWorldAd).
slowToPay(moe, last1year).
slowToPay(peg, last1year).
hasChargeCard(moe, store).
hasChargeCard(peg, store).
loyalPurchaser(Cust) :- purchaseHistoryFrequency(Cust, "$100", 10, last1year).
loyalPurchaser(Cust) :- purchaseHistoryGenreSpread(Cust, "$100", 5, last2years).
loyalPurchaser(Cust) :- regressionPredicts(Cust, "$40", monthlyPurchases).
purchaseHistoryGenre(tim, "$20", humor, last2years).
regressionPredicts(kim, "$40", monthlyPurchases).
purchased(zoe, purchase888, last30days).
ordinaryCustomer(X1) :- ordinaryCustomer_u(X1), tnot(ordinaryCustomer_o(X1)).
ordinaryCustomer_o(Cust) :- loyalCustomer_u(Cust).
ordinaryCustomer_o(Cust) :- premiereCustomer_u(Cust).
ordinaryCustomer_c_4(Cust).
ordinaryCustomer_u(Cust) :- ordinaryCustomer_c_4(Cust),
```

```
                              tnot(ordinaryCustomer_r_4(Cust)).
ordinaryCustomer_r_4(Cust) :- loyalCustomer_c_6(Cust),
                                overrides(steadySpender, presumeOrdinary).
ordinaryCustomer_r_4(Cust) :- loyalCustomer_c_12(Cust),
                                overrides(storeCard, presumeOrdinary).
ordinaryCustomer_r_4(Cust) :- premiereCustomer_c_8(Cust),
                                overrides(bigSpender, presumeOrdinary).
ordinaryCustomer_r_4(Cust) :- premiereCustomer_c_14(Cust),
                                overrides(platinumClub, presumeOrdinary).
ordinaryCustomer_c_10(Cust) :- slowToPay(Cust, last1year).
ordinaryCustomer_u(Cust) :- ordinaryCustomer_c_10(Cust),
                              tnot(ordinaryCustomer_r_10(Cust)).
ordinaryCustomer_r_10(Cust) :- loyalCustomer_c_6(Cust),
                                 overrides(steadySpender, slowPayer).
ordinaryCustomer_r_10(Cust) :- loyalCustomer_c_12(Cust),
                                 overrides(storeCard, slowPayer).
ordinaryCustomer_r_10(Cust) :- premiereCustomer_c_8(Cust),
                                 overrides(bigSpender, slowPayer).
ordinaryCustomer_r_10(Cust) :- premiereCustomer_c_14(Cust),
                                 overrides(platinumClub, slowPayer).
purchaseHistoryFrequency(cal, "$100", 10, last1year).
purchaseHistoryFrequency(joe, "$10000", anytimes, last5years).
purchaseHistoryFrequency(peg, "$10000", anytimes, last5years).
promote(dilbert, Cust) :- shopper(Cust), likes(humor, Cust).
promote(Product, Cust) :- currentMonth(december), shopper(Cust),
                            calendar(Product), buyingGifts(Cust).
loyalCustomer(X1) :- loyalCustomer_u(X1), tnot(loyalCustomer_o(X1)).
loyalCustomer_o(Cust) :- premiereCustomer_u(Cust).
loyalCustomer_o(Cust) :- ordinaryCustomer_u(Cust).
loyalCustomer_c_6(Cust) :- loyalPurchaser(Cust).
loyalCustomer_u(Cust) :- loyalCustomer_c_6(Cust),
                           tnot(loyalCustomer_r_6(Cust)).
loyalCustomer_r_6(Cust) :- premiereCustomer_c_8(Cust),
                             overrides(bigSpender, steadySpender).
loyalCustomer_r_6(Cust) :- premiereCustomer_c_14(Cust),
                             overrides(platinumClub, steadySpender).
loyalCustomer_r_6(Cust) :- ordinaryCustomer_c_4(Cust),
                             overrides(presumeOrdinary, steadySpender).
loyalCustomer_r_6(Cust) :- ordinaryCustomer_c_10(Cust),
                             overrides(slowPayer, steadySpender).
loyalCustomer_c_12(Cust) :- hasChargeCard(Cust, store).
loyalCustomer_u(Cust) :- loyalCustomer_c_12(Cust),
                           tnot(loyalCustomer_r_12(Cust)).
loyalCustomer_r_12(Cust) :- premiereCustomer_c_8(Cust),
                              overrides(bigSpender, storeCard).
loyalCustomer_r_12(Cust) :- premiereCustomer_c_14(Cust),
                              overrides(platinumClub, storeCard).
loyalCustomer_r_12(Cust) :- ordinaryCustomer_c_4(Cust),
                              overrides(presumeOrdinary, storeCard).
loyalCustomer_r_12(Cust) :- ordinaryCustomer_c_10(Cust),
                              overrides(slowPayer, storeCard).
purchaseHistoryGenreSpread(joe, "$100", 5, last2years).
shopper(ann).
shopper(cal).
```

```
shopper(joe).
shopper(kim).
shopper(moe).
shopper(peg).
shopper(sue).
shopper(tim).
shopper(zoe).
inShoppingCart(sue, bigBookOfTrains).
seenBy(dilbert, tim).
shipped(purchase888, address777).
gift(bigBookOfTrains).
premiereCustomer(X1) :- premiereCustomer_u(X1), tnot(premiereCustomer_o(X1)).
premiereCustomer_o(Cust) :- ordinaryCustomer_u(Cust).
premiereCustomer_o(Cust) :- loyalCustomer_u(Cust).
premiereCustomer_c_8(Cust) :- purchaseHistoryFrequency(Cust, "$10000",
                                                       anytimes, last5years).
premiereCustomer_u(Cust) :- premiereCustomer_c_8(Cust),
                              tnot(premiereCustomer_r_8(Cust)).
premiereCustomer_r_8(Cust) :- ordinaryCustomer_c_4(Cust),
                                overrides(presumeOrdinary, bigSpender).
premiereCustomer_r_8(Cust) :- ordinaryCustomer_c_10(Cust),
                                overrides(slowPayer, bigSpender).
premiereCustomer_r_8(Cust) :- loyalCustomer_c_6(Cust),
                                overrides(steadySpender, bigSpender).
premiereCustomer_r_8(Cust) :- loyalCustomer_c_12(Cust),
                                overrides(storeCard, bigSpender).
premiereCustomer_c_14(Cust) :- memberPlatinumClub(Cust).
premiereCustomer_u(Cust) :- premiereCustomer_c_14(Cust),
                              tnot(premiereCustomer_r_14(Cust)).
premiereCustomer_r_14(Cust) :- ordinaryCustomer_c_4(Cust),
                                 overrides(presumeOrdinary, platinumClub).
premiereCustomer_r_14(Cust) :- ordinaryCustomer_c_10(Cust),
                                 overrides(slowPayer, platinumClub).
premiereCustomer_r_14(Cust) :- loyalCustomer_c_6(Cust),
                                 overrides(steadySpender, platinumClub).
premiereCustomer_r_14(Cust) :- loyalCustomer_c_12(Cust),
                                 overrides(storeCard, platinumClub).
memberPlatinumClub(kim).
n_homeAddress(zoe, address777).
buyingGifts(Cust) :- gift(Product), inShoppingCart(Cust, Product).
buyingGifts(Cust) :- purchased(Cust, Purchase, last30days),
                       shipped(Purchase, Address), n_homeAddress(Cust, Address).
n_showAd(X1, X2) :- n_showAd_u(X1, X2), tnot(n_showAd_o(X1, X2)).
n_showAd_o(X1, X2) :- showAd_u(X1, X2).
n_showAd_c_26(Product, Cust) :- bought(Product, Cust).
n_showAd_u(Product, Cust) :- n_showAd_c_26(Product, Cust),
                               tnot(n_showAd_r_26(Product, Cust)).
n_showAd_r_26(X1, X2) :- showAd_c_24(X1, X2), overrides(goAd, noNeedAd).
n_showAd_r_26(X1, X2) :- showAd_c_25(X1, X2), overrides(goAd, noNeedAd).
n_showAd_c_27(Product, Cust).
n_showAd_u(Product, Cust) :- n_showAd_c_27(Product, Cust),
                               tnot(n_showAd_r_27(Product, Cust)).
n_showAd_r_27(X1, X2) :- showAd_c_24(X1, X2), overrides(goAd, closedWorldAd).
n_showAd_r_27(X1, X2) :- showAd_c_25(X1, X2), overrides(goAd, closedWorldAd).
```

```
belowThreshold(probWillSee(mountainLakes, zoe), adProbThreshold(mountainLakes)).
about(loyalty, steadySpender).
about(loyalty, storeCard).
likes(humor, Cust) :- purchaseHistoryGenre(Cust, "$20", humor, last2years).
currentMonth(december).
---------------------------------------------------------------------------
```