Tabling for Non-Monotonic Programming: Preference Logic

Terrance Swift

Overview

- Introduction to Preference Logics
- Introduction to Tabling
- Implementation of Preference Logic Programs
- Case study 1: Preference Logic and Data Standardization
- Case study 2: Preference Logic and Annotation Logic
- Tabling for other Non-monotonic formalisms

High-level presentation: technical details can be found in associated papers.

Preference Logic

A simple example:

```
prefer(path(X,Y,C1),path(X,Y,C2)):-
        C1 < C2.

path(X,Y,C) :- path(X,Z,C1),
        edge(Z,Y,C2),
        C is C1 + C2,
        C >= 0.
path(X,Y,C) :- edge(X,Y,C),
        C >= 0.
```

Preference Logic

- Preference Logic Programs contain
  1. An object program P
  2. A set of preference rules that relate to rules or atoms in P.
     - E.g.

```
prefer(path(X,Y,C1),path(X,Y,C2)):-
          C1 < C2.
```

- Formulations differ in
  - Whether P is definite, normal, or includes constraints.
  - Whether preferences can be computed apart from P.
- Formulated by [Govindarajan, Jayaraman, Mantha 1995]; [Brewka 1996].

Preference Logic

- Three simplifying assumptions
  1. P is (left-to-right dynamically) stratified
  2. Preferences can be computed apart from P.
  3. P has *optimal subproblem property*, e.g. preferred goals do not depend on non-preferred subgoals.
- First assumption for simplicity, second and third allow flexibility of when preferences are applied to solutions
- Semantics:
  - Possible-worlds: [Govindarajan, Jayaraman, Mantha 1995]
  - Fixed-point: [Brewka 1996], [Cui, Swift, Warren 1999]

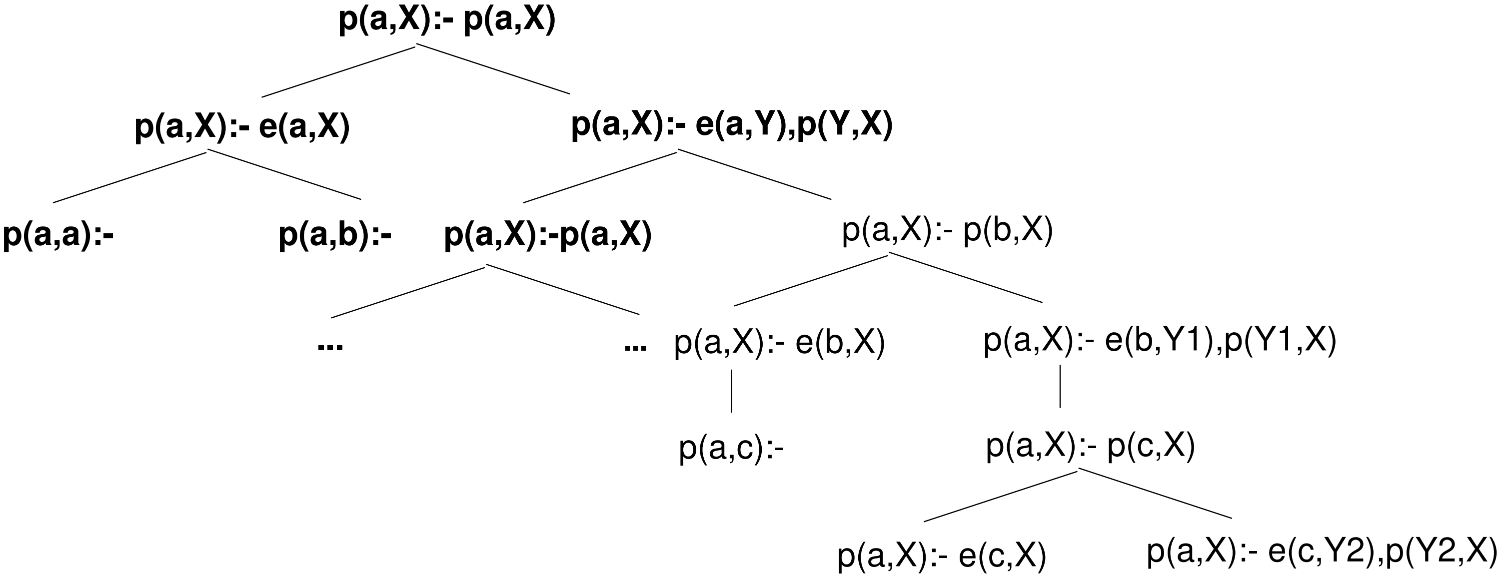
Preference Logic

How to implement preference logic programs?

- Could use Prolog, but
  - SLD(NF) is complete for Horn clauses, but does not terminate for *bounded term-depth* programs.
  - * e.g. finite number of subgoals and answer substitutions, as with Datalog programs.
  - may have poor complexity, even when it terminates.
    * e.g.
      ```
      sg(X,X)
      sg(X,Y):- par(X,Z),sg(Z,Z1),par(Y,Z1).
      ```
      over a directed acyclic graph.
  - SLD does not give a convenient way to *aggregate* solutions.

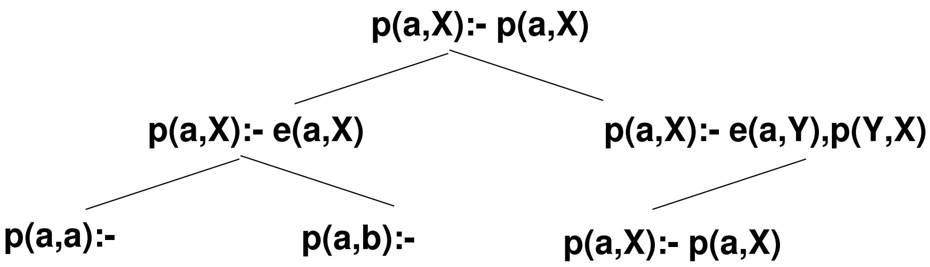
Tabling: (Definite Programs)

An Example of SLD Evaluation



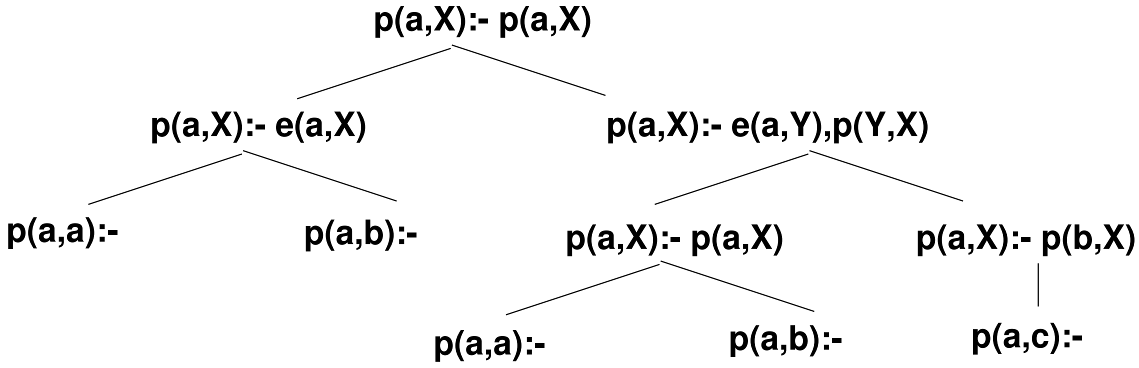

```
?- p(a,X)

p(X,Y) :- e(X,Y)
p(X,Y) :- e(X,Z),p(Z,Y)

e(a,a).  e(a,b).  e(b,c).
```

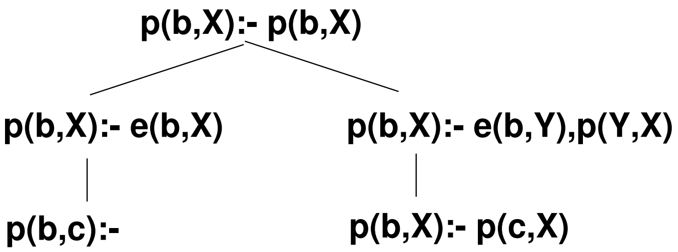
Tabling: (Definite Programs)

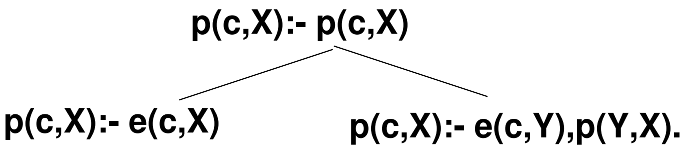



```
?- p(a,X)

:- table p/2.
p(X,Y) :- e(X,Y)
p(X,Y) :- e(X,Z),p(Z,Y)

e(a,a).  e(a,b).  e(b,c).
```

Tabling: (Definite Programs)








```
?- p(a,X)

:- table p/2.
p(X,Y) :- e(X,Y)
p(X,Y) :- e(X,Z),p(Z,Y)

e(a,a).  e(a,b).  e(b,c).
```

Tabling

- A tabled evaluation can be formulated as a sequence of forests like those in the previous slides. If the well-founded semantics is required, the sequence may be transfinite.
- Tabling factors out redundant subcomputations by maintaining record of subgoals and their answers (represented in this formulation as subgoal trees and answer nodes).
- This can be thought of as adding a New Subgoal and a Answer Clause Resolution operation (among others) to the Program Clause Resolution operation of SLD.
- Can be efficiently implemented and integrated with Prolog: XSB, YAP, B-Prolog, Mercury.

Tabling

Some Results for a brand of tabling called SLG (Chen and Warren, 1996)

- SLG is sound and complete for non-floundering evaluations of normal programs
- SLG terminates for programs with the bounded-term depth property.
- Ground queries to Datalog programs can be evaluated with polynomial data complexity.

Recently reformulated for easier presentation and extension [Swift 1999a)

Implementing Preference Logic Programs

- Naive solution: Generate all solutions and maintain only those that are preferred
- Better solution: At each stage, return only *preferred* answers. This generalizes a branch-and-bound algorithm.

Implementing Preference Logic Programs

- Change the ANSWER CLAUSE RESOLUTION step to include a computable *aggregation function*, i.e. a function from a bag of answer nodes for a subgoal tree to a set of answer nodes for a subgoal tree.
  - E.g. let the set of answer nodes for a forest $\mathcal{F}$ and subgoal S be $A_{S,\mathcal{F}}$, the aggregated set of answers is $aggregate(A_{S,\mathcal{F}})$.
  - More specifically, let

    $$\{path(a, b, 5), path(a, b, 6), path(a, b, 7)\}$$

    be the set of answer nodes for the subgoal

    $$? - path(a, b, X)$$

    in the forest $\mathcal{F}$. Then

    $$aggregate(\{path(a, b, 5), path(a, b, 6), path(a, b, 7)\})$$

    might be

    $$\{path(a, b, 5)\}$$

Implementing Preference Logic Programs

```
path(X,Y,C) :-               filterPO(pref_path(X,Y),prefer).

pref_path(X,Y,C) :-          path(X,Z,C1),
                             edge(Z,Y,C2),C is C1 + C2,C >= 0.
pref_path(X,Y,C) :-          edge(X,Y,C),C >= 0.

prefer(pref_path(F,T,X), pref_path(F,T,Y)):-
                             X < Y.
```

- The predicate **`filterPO/3`** is tabled and performs tabled aggregation on subgoals of **`path/2`** according to the aggregation function **`prefer/2`**.
- Using Local Scheduling [Freire, Swift, Warren, 1997] a minimality condition can be obtained on the propagation of non-preferred solutions
- The above program can be considered as the compiled form of the preference program for **`path/3`**

Case Study 1: Data Standardization

- Its grammatical form, Preference Logic Grammars, can be quite useful for specifying and resolving ambiguous grammars.
- Example: the "dangling-else" problem [55]:

```
ifstmt(if(C,T)) -->
        [if],cond(C),[then],stmtseq(T).
ifstmt(if(C,T,E)) -->
        [if],cond(C),[then],stmtseq(T),
        [else],stmtseq(E).
```

can be disambiguated via

```
prefer(ifstmt(C,if(C1,T,E)),
        ifstmt(if(C,if(C1,T),E))).
```

Disambiguation of multiple parses can be done via preference rules.

- A case study [Cui, Swift, Warren, 1999] of moving from DCGs to PLGs as implemented with tabled aggregation.
- Test case was a commercial name-address "standardizer" [Swift et. al, 1994] that extracts standardized name and address data from textual fields.
- Prolog version had standardized data for
  - U.S. Customs Service
  - Defense Logistics Agency
  - a large investment bank
  - a large electronics distributor

Case Study 1: Data Standardization

Example: the string

```
TO THE ORDR OF ZZZ AUTOPARTS INC
DBA PARTCO 123 WASHING
TON STREET EL SEGUNDO CA
```

Is standardized to

Organization:	`ZZZ AUTOPARTS`
Type:	`basic`

Organization:	`PARTCO`
Street:	`129 WASHINGTON ST`
PO BOX:	
City:	`EL SEGUNDO`
State:	`CA`
Zip:	`90245`
Type:	`dba`

Case Study 1: Data Standardization

- Prolog Architecture
  1. Tokenize
  2. Collect primitive tokens into "supertokens"; e.g. 'LONG','ISLAND','CITY' into 'LONG ISLAND CITY'
  3. Apply DCG parsing rules
  4. Postprocess to check consistency, and fix mistakes
- First two steps were easily maintained: supertokenization could be generated.
- Second two steps were difficult to maintain:
  - In order to avoid ambiguous parses, the parser had states in which tokens that were not part of a first or follow set (e.g. 'ZZZ','PARTCO', 'WASHINGTON') were interpreted by default as company names, streets, cities etc.

Case Study 1: Data Standardization

- Better Solution: generate all parses and keep the best.
- Preferred rules can call other preferred rules, so that pruning can be done for sub-parses.
- Parsing rules become very general
  - An *organization* is defined by any words possibly followed by a sequence of org terminators (e.g. 'INC', 'ASSOC').
  - Preference rules indicate that
    1. A parse with a fuller org title is preferred, e.g.

       org('ZZZ AUTOPARTS','INC')

       to

       org('ZZZ AUTOPARTS','')
    2. More specific parses are preferred (e.g. a parse that includes a street) to a similar one that does not.

Case Study 1: Data Standardization

Comparison as of Jan 1999

Function	Clauses	Lines
Tokenization	94	412
Bottom-up Parse	26205	26205
Domain Information	59150	59150
Control and Utilities	727	1345
(Prolog) Top-Down	724	2082
(Prolog) Post-processing	604	2838
(PLP) Top-Down	198	686
(PLP) Post-processing	7	106

Case Study 1: Data Standardization

- Preference Logic Standardizer has been used for
  - Defense Logistics Agency
  - A large pharmaceutical company
  - Two large electronics distributors
- Runtime about 3x slower than old standardizer
  - Explore better compilation, implementation of `filterPO/2`.
  - Explore use of parallelism, particularly Table-or parallelism (YAP).

Case Study 2: Annotated Logic

Consider data mining via association rules

- A data item with $independant\_values = SET_1$ has $dependant\_values = SET2$ with certainty M
  - E.g. for a database of aircraft parts:

    A data item with Nomenclature = `ROD END CLEVIS` has process = `MACHINING` with certainty 97%
- Association rules can find out interesting, trivial, or just wrong information.

Case Study 2: Annotated Logic

- Generate all association rules subject to minimal standards and only retain those that are preferred
- Preferences might be based on
  - *Support* — how many items with
    $$independant\_values = SET_1$$
    exist in the sample.
  - *Precision* — % of items with
    $$independant\_values = SET_1$$
    that have
    $$dependant\_values = SET_2.$$
  - *Specificity* — Inferring that a class of parts are made of high-cromium, corrosion-resistant steel is more interesting than inferring that it is made of metal.
  - *Conciseness* — The smaller the number of independant values, the better.

Case Study 2: Annotated Logic

- Preferences gives a declarative way to weigh these factors, and others.
- Rules generated in this way have been used for Defense Logistics Agency and a large defense contractor

Case Study 2: Annotated Logic

- Preference Logics solve half the problem of data mining — they allow us to generate association rules.
- But we have not addressed the certainty measure.
  - how to we effectively reason with these association rules, that have this certainty measure?
  - How do we fold them into an intensional database?
- Consider the certainty measure as an *annotation* and reason using the formulation of Annotated Logic Programming [Kifer and Subrahmanian, 1992] which provides a means of performing quantitative reasoning in logic programs.
- Then implement the annotations using preferences!

Case Study 2: Annotated Logic

Annotated logic from the viewpoint of preference logic

- Define an *annotation domain*, $\mathcal{T}$ as a semi-lattice, i.e. a partial order with a join operation.
- Define a single (infix) predicate `:/2`, e.g. $Atom : Annotation$, in which the second argument consists of elements of $\mathcal{T}$.
- add axiom

  $(A : V_1 \wedge A : V_2) \Rightarrow (A : V_3 \wedge \ V_3 = join(V_1, V_2))$
- Define the preference rule

  `prefer(A1:Ann1,A2:Ann2):- Ann1` $>_{\mathcal{T}}$ `Ann2.`

Case Study 2: Tabled Aggregation

- $\mathcal{T}$ may denote an interval $[\infty, 0]$
- The join operation is simply max on $\mathcal{T}$ (i.e. a min relation)

```
:- import filterPO/2 from aggregs.

shortest_path(X,Y):C :- filterPO(sp(X,Y):C,sp_prefer).

sp(X,Y):C :- shortest_path(X,Z):C1,
       edge(Z,Y):C2,C is C1 + C2.
sp(X,Y):C :- edge(X,Y):C.


prefer(A1:Ann1,A2,Ann2):- Ann1 < Ann2.

A:V3 :- A:V1,A:V2,join(V1,V2,V3).
```

Case Study 2: Annotated Logic

Examples: Quantitative Reasoning

- $\mathcal{T}$ may denote a "strength of belief" (e.g. [Van Emden 1986])
  - `prefer/2` is a numerical comparison
  - the join operation is a max operation
- $\mathcal{T}$ may denote probability intervals (e.g. [Ng, Subrahmanian, 1993, Dekhtyar, Subrahmanian 1999])
  - For *Atom* : *Annotation*, *Annotation* denotes a probability interval for which *Atom* is true
  - `prefer/2` is an interval containment relation
  - The join operation is an interval intersection operation
  - E.g. for the program

    ```
    p:[0.7, 0.8].     p:[0.78, 0.92].
    ```

    The query `?- p:X` gives $X = p : [0.78, 0.8]$
- Independent probabilities can be handled in this way, as well as certain cases of ignorance and correlation.

Case Study 2: Tabled Aggregation

Examples:

- Multi-valued logics
  - $\mathcal{T}$ may denote the *Belnap* lattice: $\{\bot, \mathbf{f}, \mathbf{t}, \top\}$
    - Query `?- p:X` to program

      `p:t.        p:f`

      gives `p:`$\top$
  - $\mathcal{T}$ may denote other lattices containing default truth values, or bilattices.
  - (Definite) Bilattice Logic Programs [Fitting, 1991] can be embedded into (Definite) Annotated Logic Programs.

Case Study 2: Tabled Aggregation

- Possible-world semantics
  - For *Atom* : *Annotation*, *Annotation* denotes sets of worlds in which *Atom* is true
  - `prefer/2` is the subset relation
  - the join operation is set union
  - Useful for certain temporal logics, amalgamated databases, agents, etc.
    - Consider the database `p:[{a},f]. p:[{b},t].` The query `?- p:[{a},X]` gives $X = t$, while `?- p:[{a,b},X]` gives $X = \top$.

Case Study 2: Tabled Aggregation

- The aggregate function may also be changed, so that $aggregate(f_{A,S})$, directly computes the join of all answer nodes in $f_{A,S}$.
- This gives a more efficient implementation, using the XSB library predicate `filterReduce/3`
- A meta-interpreter for annotated programs is now a package of XSB.

Summary

- Preference Logic can be easily implemented using tabling in XSB
  - Tabling provides materialization of "best" answers
  - Scheduling provided by tabling reduces propagation of non-preferred answers.
- Preference Logic is useful commercially for Standardization and data mining
- Preference Logic can be seen to be equivalent to Annotation Logic, with similar computational requirements.

Summary

Other non-monotonic formalisms implemented in XSB

- WFS(X): Well-founded semantics (with explicit negation), used for
  - Model-checking [Ramakrishna et. al. 1997] (research use)
  - Model-based diagnosis [Damasio, Pereira ??; Swift, 1999b] (research and commercial uses)
  - Machine learning [Lamma, Pereira, Riguzzi, 1998] (research use)
- Frame Logic: Object-oriented extension of normal programs
  - Internet agent [Davulcu, Kifer ??] (commercial use)
- Stable models (in conjunction with Smodels
  - Model-checking [Lui, Ramakrishnan, Smolka, 1998] (research use)

Summary

- XSB and source code is available via

  `http://www.cs.sunysb.edu/~sbprolog`

- Development is informal multinational collaboration — developers active in Stony Brook, Maryland, Upsalla, Leuven, Lisbon.
- Maybe Buffalo?

Diagnosis

Diagnostic Criteria for Dementia of the Alzheimer's Type

1. The development of multiple cognitive defects manifested by both
   (a) memory impairment (impaired ability to learn new information or to recall previously learned information)
   (b) one (or more) of the following cognitive disturbances
      i. aphasia (language disturbance)
      ii. apraxia (impaired ability to carry out motor activities despite intact motor function)
      iii. agnosia (failure to recognize or identify objects despite intact sensory functions)
      iv. disturbance in executive functioning (i.e., planning, organizing, sequencing, abstracting)
2. The cognitive deficits in criteria A1 and A2 each cause significant impairment in social or occupational functioning and represent a significant decline from a previous level of functioning.
3. The course is characterized by gradual onset and continuing cognitive decline.
4. The cognitive deficits in Criteria A1 and A2 are not due to any of the following
   (a) other central nervous system conditions that cause progressive deficits in memory and cognition (e.g. cerebrovascular disease, Parkinson's disease, Huntington's disease, subdural hematoma, normal-pressure hydrocephalus, brain tumor)
   (b) systemic conditions that are known to cause dementia (g. hypothyroidism, vitamin B_{12} or folic acid deficiency, niacin deficiency, hypercalcemia, neurosyphilis, HIV infection
   (c) substance-induced conditions
5. The deficits do not occur exclusively during the course of a delirium
6. *The disturbance is not better accounted for by another Axis I disorder (e.g., Major Depressive Disorder, Schizophrenia)*

Case Study 1: Psychiatric Assesment

Goal: represent DSM-IV as a logic program and implement in a system for physicians. This will aid the physician in making decisions, and revising these decisions as information becomes available.

- DSM-IV gives rise to many knowledge representation difficulties
  - How to represent conflicting diagnoses

```
'Alzheimer's Dementia' :-
          not_better 'Major Depressive Disorder'
'Major Depressive Disorder':-
          not_better 'Alzheimer's Dementia'
```

- How to represent incomplete information
  - E.g. tests have not been run, a full case history is unavailable, etc.

Case Study 1: Psychiatric Assesment

- This meta-interpreter can be partially evaluated to a more efficient program, which is a package in XSB.
- XSB has partially implemented WFSX with abduction for a psychiatric assesment system called *Diagnostica*. Web address:

  `http://www.medicinerules.com`

- Process of writing a meta-interpreter for an extension of the well-founded semantics, and then specializing the meta-interpreter is a standard technique for using tabling for non-monotonic programming (used for WFSX, Annotated Logic Programs, Abductive Logic Programs, F-logic [70]).

Case Study 2: Tabled Aggregation

- ANSWER CLAUSE RESOLUTION: Let $\mathcal{E}$ be an SLG_X evaluation with answer aggregation function *aggregate*. Let $\mathcal{F}_n$ contain a non-root node N whose selected literal S is positive, and let $Ans \in aggregate(A_{\mathcal{F}_n,S})$ and N_{child} be the SLG resolvent of N and Ans on S. Assume that in $\mathcal{F}_n$, N does not have a child N_{child}. Then add N_{child} as a child of N.