Design Patterns for Tabled Logic Programming

Terrance Swift
CENTRIA: Centro de Inteligência Artificial
Universidade Nova de Lisboa

International Conference on Applications of Declarative Programming and Knowledge Management

Outline

- Present various tabling “patterns”
  - Used as an organizing concept; not a means for automated software engineering.ß
- “Vanilla” tabling -- query evaluation for definite programs
- Tabled negation for the well-founded and stable semantics
- Beyond query evaluation -- call subsumption for (sub-)model generation
- Beyond **t,f,u** -- answer subsumption for quantitative and para-consistent programming
- State of the preceding features within XSB 3.2 (and 3.3 alpha)

The tabling methodology discussed in this talk was developed and implemented by (in alphabetical order)

Luis de Castro, Baoqiu Cui, Steve Dawson,
Ernie Johnson, Juliana Freire, Michael Kifer,
Rui F. Marques, C.R. Ramakrishnan, I.V. Ramakrishnan,
Prasad Rao, Konstantinos Sagonas, Diptikalyan Saha,
Terrance Swift,
David S. Warren

Citations for most of the work discussed here can be found in
http://www.cs.sunysb.edu/~tswift/webpapers/tplp-submit.pdf

Why Table?

- Tabling changes the evaluation strategy of a logic program -- sometimes dramatically
- This can ensure termination, and reduce (or increase) complexity
- A more sophisticated evaluation strategy allows more sophisiticated semantics to be implemented more easily

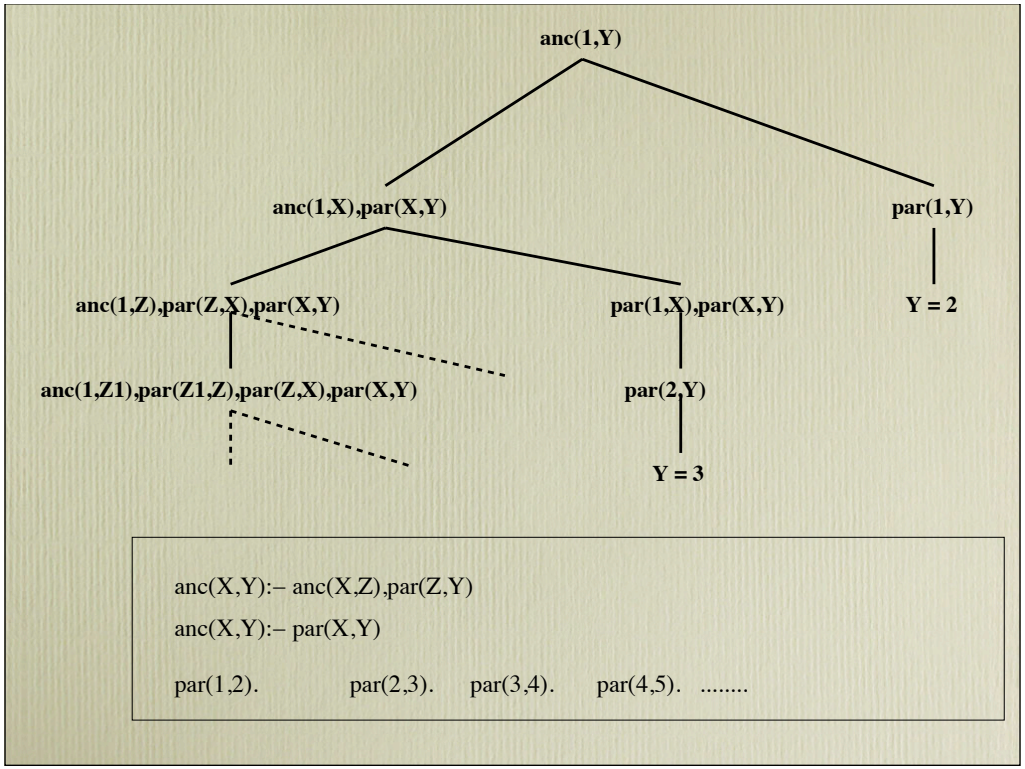
Inadequacies of Prolog (SLDNF)

* Lack of termination for e.g. Datalog programs

```
ancestor(X,Y):- ancestor(X,Z),parent(Z,Y).
ancestor(X,Y): -parent(X,Y).
```

* Although SLD is complete for all definite programs, it may not terminate for all bounded term-depth programs (e.g. Datalog programs)

* The following slide shows an SLD search tree for a query to `ancestor/2` above



anc(X,Y):– anc(X,Z),par(Z,Y)

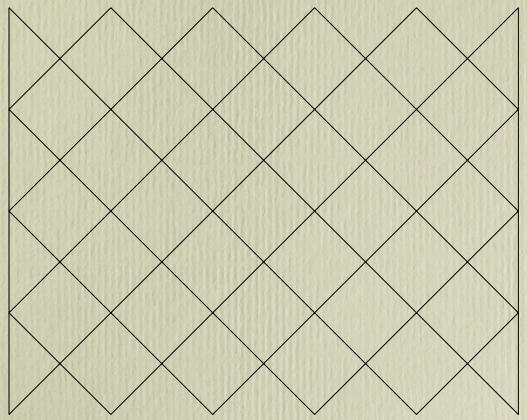
anc(X,Y):– par(X,Y)

par(1,2). par(2,3). par(3,4). par(4,5). .....

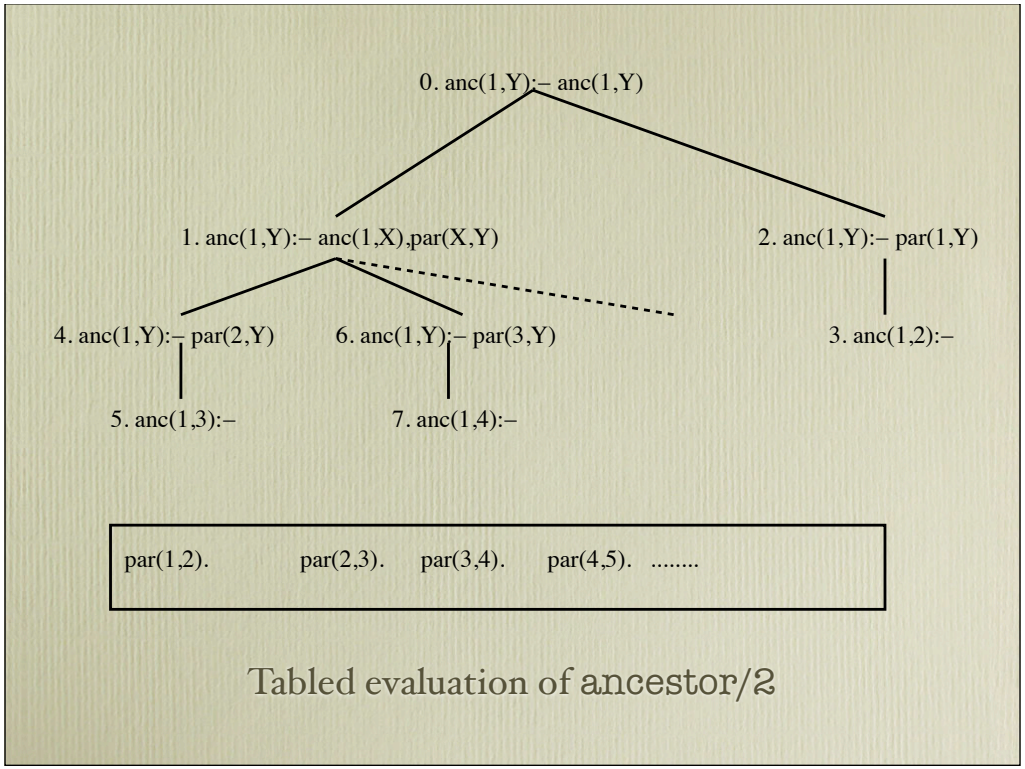
- The SLD tree contains all solutions but is infinite
- Loop-checking could achieve termination, but will not address complexity

Poor complexity, even when SLD terminates

```
sg(X,X).
sg(X,Y):- par(X,Z),sg(Z,Z1),par(Y,Z1).
```



Evaluating sg/2 with SLD will be proportional to the number of *paths* in the graph, rather than the number of edges



Tabled evaluation of ancestor/2

* Tabling factors out repeated subcomputations by keeping track of which (tabled) subgoals have been called. E.g repeated calls to

```
anc(X,Y):- anc(X,Y),par(Z,Y)
```

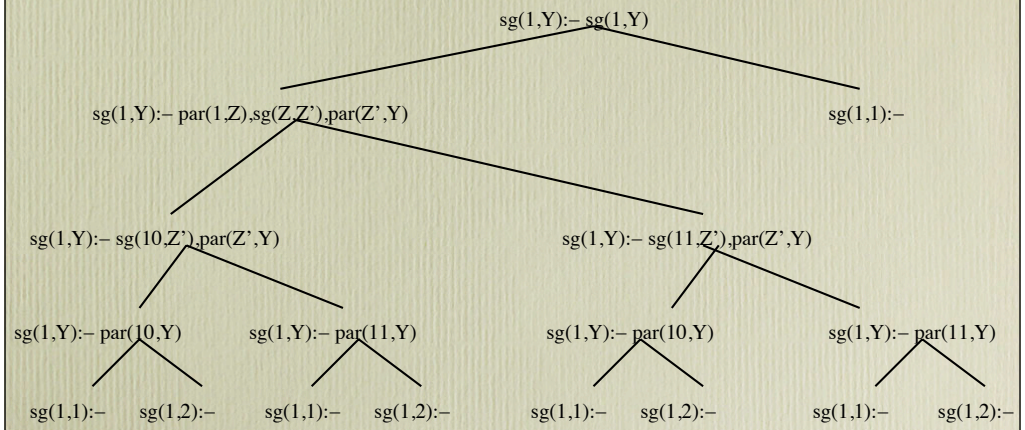
* Computation of each tabled subgoal can be modeled as a separate tree -- a computation is now a forest of trees

* A bounded term-depth (finite) program is simply one in which the size of all subgoals and answers is bounded.

    -- Such programs have a finite number of subgoals and answers

* For comparison with `ancestor/2`, the next slide shows a more general situation of `sg/2`

* The forest for ?- sg(1,X) would contain a tree similar to the following for every node in the graph



The repeated computations are thus factored into n trees -- though there is still some redundancy

But don't table all predicates!

```
append([],L,L).
append([H|T],L,[H|Tl]):- append(T,L,Tl).
```

can be seen to have a right recursive form:

```
append([],L,L).
append(Term.L,[H|Tl]):- cons(Term,H,T),append(T,L,Tl).
```

The query: append([a,b,X],[c],Y) is quadratic in the size of the goal in most implementations. It makes the subqueries

```
append([a,b,X],[c],Y).
append([b,X],[c],Y).
append([X],[c],Y).
append([],[c],Y).
```

* SLD(NF) is great for traversing Prolog terms -- which are trees (when rational terms are not implemented)

* For programs like `append/3`, there is no redundancy to factor, even if copying into the table was not an issue

* Tabling therefore augments Prolog -- it by no means replaces SLDNF.

Left Recursion

pred(...) :- pred(...),L_1,...L_n.

Linear Recursion

pred(...):- L_1,...,L_{M-1},pred(...),L_{M+1},...,L_N.

Non-Linear Recursion

pred(...):-

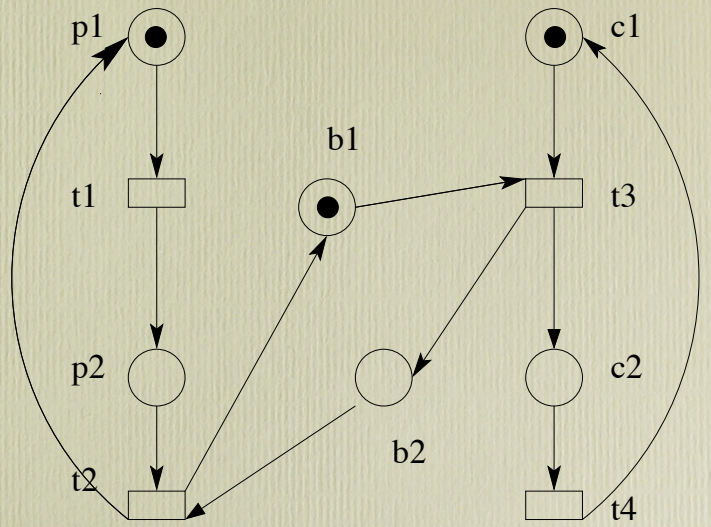
L_1,...L_{k-1},pred(...),L_{k+1},...,L_{M-1},pred(...), L_M,..., L_N.

* Left Linear Recursion is most efficient for tabling for most calling patterns

* Not all recursion can be transformed to left recursion (e.g. sg/2)

* But recursive search through general data structures represented by sets of clauses can be extremely useful

- So how does all this help?
- Tabling can be used to traverse through information stored as sets of facts (or clauses)
- One major example is that of the representation of a given transition system through a process logic
    - CCS, π-calculus, Petri nets, etc.
- Consider an example for *Elementary* Nets, where each place can have at most one token




```
% Prolog representation of the above Producer-Consumer Net
:- index(trans/2,trie).
trans([p1],[p2],t1).          trans([b2,p2],[p1,b1],t2).
trans([b1,c1],[b2,c2],t3).    trans([c2],[c1],t4).
```

```
% Program to determine reachability of an elementary net
:- table reachable/2.
reachable(InConf,NewConf):-
  reachable(InConf,Conf),
  hasTransition(Conf,NewConf).
reachable(InConf,NewConf):-
  hasTransition(InConf,NewConf).

hasTransition(Conf,NewConf):-
  get_trans_for_conf(Conf,AllTrans),
  member(Trans,AllTrans),
  apply_trans_to_conf(Trans,Conf,NewConf).

get_trans_for_conf(Conf,Flattrans):-
  get_trans_for_conf_1(Conf,Conf,Trans),
  flatten(Trans,Flattrans).

get_trans_for_conf_1([],_Conf,[]).
get_trans_for_conf_1([H|T],Conf,[Trans1|RT]):-
  findall(trans([H|In],Out,Tran),trans([H|In],Out,Tran),Trans),
  check_concession(Trans,Conf,Trans1),
  get_trans_for_conf_1(T,Conf,RT).

check_concession([],_,[]).
check_concession([trans(In,Out,Name)|T],Input,[trans(In,Out,Name)|T1]):-
  ord_subset(In,Input),
  ord_disjoint(Out,Input),!,
  check_concession(T,Input,T1).
check_concession([_Trans|T],Input,T1):-
  check_concession(T,Input,T1).

apply_trans_to_conf(trans(In,Out_Name),Conf,NewConf):-
  ord_subtract(Conf,In,Diff),
  flatsort([Out|Diff],Temp),
```

```
reachable(InConf,NewConf):-
  reachable(InConf,Conf),
  hasTransition(Conf,NewConf).
reachable(InConf,NewConf):-
  hasTransition(InConf,NewConf).

hasTransition(Conf,NewConf):-
  get_trans_for_conf(Conf,AllTrans),
  member(Trans,AllTrans),
  apply_trans_to_conf(Trans,Conf,NewConf).
```

The heart of this code is left-recursion

hasTransition/2 is a complex form of parent/2, edge/2, etc.

get_trans_for_conf/2 finds all transitions that are enabled by the current configuration -- for an elementary net, this means that each input place has a token and each output place does not have a token.

apply_trans_to_conf/2 simply applies the transition to the configuration to produce a new configuration

Tabling and Model-Checking

* A temporal logic can also be programmed using tabling in a similarly simple manner

    -- Alternation-free modal μ-calculus, CTL, LTL
    -- Uses two mutually recursive predicates
    State |= Property and State |/= Property
    -- Can be programmed with negation or without if |/= /2 is dualized.

* Tabling can produce concise but efficient model checkers

    -- In the past tabling has been used to check workflows, multi-agent systems, real-time systems, etc.

* If the model checker makes use only of definite programs, they can take advantage of YAPs or-parallel engine with tabling

```
:- table l=/2.

State s l= prop(P):-
    has prop(State s,P).

State s l= diam(Act a,F):-
    trans(State s,Act a,State_t),


State_s l= and(X1, X2)):-
    State_s l= X1, State_s l= X2.

State s l= or(X1, X2)):-
    State s l= X1 ; State s l= X2.

State s l= form(X):-
    formula def(X, Y),
    ( Y = lfp(Z),
      State s l= Z
     ;
      Y = gfp(Z),
      tnot(State s l/= Z ).
```

```
:- table l/=/2.

State s l/= prop(P):-
     \+ has prop(State s,P).

State s l/= diam(Act a,F):-
    tfindall(State_t, trans(State_s,Act_a,State_t), State_ts)
    'list l/='(State_ts, F).

State s l/= and(X1, X2)):-
   State s l/= X1 ; State s l= X2.

State s l/= or(X1, X2)):-
   State s l/= X1, State s l/= X2.

State s l/= form(X):-
    formula def(X, Y),
    ( Y = lfp(Z),
      tnot(State s l= State s, Z)
     ;
      Y = gfp(Z),
      State s l/=Z ).
```

Alternation-Free Modal-mu Calculus

Tabling and parsing

* Similar to chart parsers, early deduction

* Applications include

  -- Parsing Montague grammars for educational software

  -- Parsing in XSB, Inc's *Ontology-Driven Extraction,* which uses an ontology to drive feature extraction for catalogs or databases of airplane parts.

* These applications table mutually recursive productions

There are numerous other applications of tabling definite programs in machine learning, the semantic web, etc. But for now, we've identified 3 different patterns

Tabling Pattern 1: Left recursion for transitive closure
-- ancestor/2, reachable/2 for elementary nets.

Tabling Pattern 2: Non-left recursion for transitive closure (e.g. sg/2)

Tabling Pattern 3: Mutually recursive predicates without negation (e.g. parsing Montague grammars, ontology-driven parsing alternation-free modal-μ calculus when dualized programs are used)

- "Vanilla" tabling -- query evaluation for definite programs
- *Tabled negation for the well-founded and stable semantics*
- Beyond query evaluation -- call subsumption for (sub-)model generation
- Beyond **t,f,u** -- answer subsumption for quantitative and para-consistent programming
- State of the preceding features within XSB 3.2

Tabled negation can also evaluate programs according to the 3-valued well-founded semantics.

The village barber shaves everyone in the village who does not shave himself (and only those people)

This can be expressed in a logic program (using default negation) as

```
shaves(barber,Person):-
        villager(Person),
        not shaves(Person,Person).
shaves(doctor,doctor).

villager(barber).      villager(mayor).      villager(doctor).
```

The barber shaves the mayor, does not shave the doctor, and it is unknown whether he shaves himself.

Tabling treats positive loops as a least fixed-point, but (ground) negative loops as undefined.

... so how is this useful?

It gives a semantics for *all* normal logic programs

It allows logic programs to adequately handle inconsistencies and paraconsistencies.

Diagnostic Criteria for Dementia of the Alzheimers Type

A. The development of multiple cognitive defects manifested by both
    1. memory impairment (impaired ability to learn new information or to recall previously learned information)
    2. one (or more) of the following cognitive disturbances
        1. aphasia (language disturbance)
        2. apraxia (impaired ability to carry out motor activities despite intact motor function)
        3. agnosia (failure to recognize or identify objects despite intact sensory functions)
        4. disturbance in executive functioning (i.e., planning, organizing, sequencing, abstracting)

B. 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.
C. The course is characterized by gradual onset and continuing cognitive decline.
D. The cognitive deficits in Criteria A1 and A2 are not due to any of the following
    1. other central nervous system conditions that cause progressive deficits in memory and cognition (e.g. cerebrovascular disease, Parkinson's disease, Huntington's disease, subdural hematomanormal-pressure hydrocephalus, brain tumor)
    2. systemic conditions that are known to cause dementia (e.g. hypothyroidism, vitamin B_{12} or folic acid deficiency, niacin deficiency, hypercalcemia, neurosyphilis, HIV infection)
    3. substance-induced conditions
E. The deficits do not occur exclusively during the course of a delirium
F. *The disturbance is not better accounted for by another Axis I disorder (e.g., Major Depressive Disorder, Schizophrenia)*

WFS Applications -- Medical Informatics

* The preceding text was from Diagnostic and Statistical Manual of Mental Disorders Edition IV (DSM-IV) which is the guide to how US physicians diagnose mental distorders

* Major depressive disorder has a similar exclusion condition (*not better accounted for by...)*

* Different physicians were on the dementia committee and the mood-disorder committee

The exclusion conditions can be modelled by loops through negation

```
'Alzheimers Dementia' :-
        not_better 'Major Depressive Disorder'
'Major Depressive Disorder':-
        not_better 'Alzheimers Dementia'
```

* In this formulation it could be unknown whether a given patient suffers from dementia or depression (if she fulfulled all other criteria)

* The 2-valued Stable Models semantics would give 2 (or more) scenarios: one where the patient had dementia, the other where the patient had depression

* The well-founded model is contained in the intersection of all stable models -- so why not use stable models?

* The best semantics depends on the use-case

    -- for diagnosis unknown may be best, indicating that more information is required.

    -- for planning, multiple scenarios may be best so that all contingencies can be accounted for

* Object logics may give rise to non-stratification. Consider the ``Nixon diamond'' problem in F-logic

```
% A republican has a unique policy of being a non-pacifist
republican[policy *-> nonpacifist].
% A quaker has a unique policy of being a pacifist
quaker[policy *-> pacifist].
% nixon is a quaker
nixon:quaker.
% nixon is a republican
nixon:republican.
```

* This knowledge base translates into a non-stratified program where

  -- Nixon's policy is undefined; or

  -- Nixon is a pacifist or a non-pacifist (exclusively)

* In either case, you have to handle updates to a programs inheritance structure -- somehow

3-valued or 2-valued?

* Recent work in cognitive psychology (Stenning & van Lambalgen) indicate that people interpret situations using 3-valued logics

    -- This theory is based on numerous cognitive tests (e.g. Wason test)

    -- In fact, they posit a neural net model of a 3-valued completion semantics

    -- This semantics differs from WFS in that positive loops with no answers are treated as undefined rather than false

* Stable models are highly useful. WFS is sometimes seen as a step towards stable model semantics, but WFS can be useful by itself.

* A 3-valued completion semantics is preferred by some for modelling human cognition.

* WFS is relevant, so it can be query-oriented -- you don't need to ground the entire program (which could have non-constant terms or use an external database)

* WFS has quadratic data complexity (but almost always behaves linearly)

* Still, you can often have the best of both worlds since tabling produces the well-founded residual of a query.

* Let's say you wanted to query an object database that included Nixon as previously.

* The inheritance hierarchy may be inconsistent only when applied to instances (i.e. the T-box may be consistent)

* Nixon is an instance. We may know that Nixon's attributes may not affect those of any other instance.

* It may be that most of Nixon's attributes are consistent (uniquely determined)

*A query may be performed under WFS, and only the non-stratified part would be sent to a stable model generator

SLG tabling adds delaying and simplification to evaluate WFS which allows it to find unfounded sets more effectively

The table for the query can be seen as a transformation of a program, where literals that are undefined in WFS were delayed, but could not be simplified.

Various queries about Nixon might give rise to a table like

```
hasAdministration(nixon,[1969-1975]).
:
hasPolicy(nixon,pacifist):- not hasPolicy(nixon,nonpacifist) |
hasPolicy(nixon,nonpacifist):- not hasPolicy(nixon,pacifist) |
```

* XSB now includes Smodels in its distribution. XSB's XASP library takes allows a user to specify which parts of a table are to be sent to Smodels to be solved

* Clearly, a query may not traverse all possible clauses and ground them -- so it has to be semantically reasonable to do this
    -- i.e. we know that a loop through negation concerning one instance is independent of loops through negation concerning other instances

* This gives rise to a different approach to ASP than those using lparse, gringo, etc.

* XASP allows each thread to have its own Smodels instance

* XASP is being extended to handle probabilistic reasoning using Plog (by C. Damasio and students)

Preferences

Let's go back to the dementia/depression example

Let's say that when in doubt you'd diagnose and treat depression, and you want to automate that. You could

1. rewrite your DSM-IV program
2. indicate that depression is to be preferred to dementia (perhaps under certain conditions). e.g.

```
prefer(depression,dementia):- <some conditions>
prefer(dementia,depression):- <other conditions>
```

These preferences work on top of WFS, and a program is transformed to check for preferences (ideally when compiled) by adding tabled negation

Note that we are preferring solutions (derived atoms) not rules (extends approach of Govindarajan, Jayaraman and Mantha to WFS

```
% Program to determine reachability of an elementary net
:- table reachable/2.
reachable(InConf,NewConf):-
  reachable(InConf,Conf),
  hasTransition(Conf,NewConf).
reachable(InConf,NewConf):-
  hasTransition(InConf,NewConf).

hasTransition(Conf,NewConf):-
  get_trans_for_conf(Conf,AllTrans),
  member(Trans,AllTrans),
  apply_trans_to_conf(Trans,Conf,NewConf).

get_trans_for_conf(Conf,Flattrans):-
  get_trans_for_conf_1(Conf,Conf,Trans),
  flatten(Trans,Flattrans).

get_trans_for_conf_1([],_Conf,[]).
get_trans_for_conf_1([H|T],Conf,[Trans1|RT]):-
  findall(trans([H|In],Out,Tran),trans([H|In],Out,Tran),Trans),
  check_concession(Trans,Conf,Trans1),
  get_trans_for_conf_1(T,Conf,RT).

check_concession([],_,[]).
check_concession([trans(In,Out,Name)|T],Input,[trans(In,Out,Name)|T1]):-
  ord_subset(In,Input),
  ord_disjoint(Out,Input),!,
  check_concession(T,Input,T1).
check_concession([_Trans|T],Input,T1):-
  check_concession(T,Input,T1).

apply_trans_to_conf(trans(In,Out_Name),Conf,NewConf):-
  ord_subtract(Conf,In,Diff),
  flatsort([Out|Diff],Temp),
```

* Preferences can be added to the Petri net example

* The preceding Petri net code can be extended to handle nearly all common workflow control patterns -- and to allow a net to be based on dynamic information

* In such a case, preferences allow a workflow to be configured to a local policy without recoding the workflow

```
hasTransition(Conf,NewConf):-
    get_trans_for_conf(Conf,AllTrans),
    member(Trans,AllTrans),
    sk_not(preferred(SomeTrans,Trans,AllTrans,Conf))
    apply_trans_to_conf(Trans,Conf,NewConf).
```

* Preference logic grammars add preferences to Prolog's Definite Clause Grammars

* Preference address ambiguities in the grammar
  -- Different parse (sub)trees may be encoded in the solution
  -- Preferences resolve ambiguities outside of the rules

* One project rewrote an industrial data "standardizer" written using Prolog DCGs and transformed it into prefrence logic grammars.
  -- The grammar for the resulting standardizer was about 1/5 the size of the original (though a little slower)
  -- The resulting standardizer has been used by XSB, Inc on various projects

* Simple grammar rules + preferences are conceptually easier than more complex grammar rules

* The language Silk is being funded as a rule-based knowledge representation language by the Halo Project

    -- Silk is based on Flora-2 which is based on XSB

* Silk uses a type of defeasibility logic that is similar to preferences as described here.

Tabling Pattern 4: Tabled Negation for amalgamating data
-- DSM-IV knowledge representation

Tabling Pattern 5: Tabling as grounder for an ASP solver to produce partial stable models
-- Acorda and other projects at CENTRIA

Tabling Pattern 6: Tabling as implementation for preferences (or similar WFS-based non-monotonic constructs)
-- Taking preferred transitions in a workflow

Tabling Pattern 7: Tabling as implementation of Preference Logic Grammars
-- XSB, Inc data standardizer

- "Vanilla" tabling -- query evaluation for definite programs
- Tabled negation for the well-founded and stable Semantics
- *Beyond query evaluation -- call subsumption for (sub-)model generation.*
- Beyond **t,f,u** -- answer subsumption for quantitative and para-consistent programming
- Progress on mult-threading and parallelism

Call Subsumption

* Maybe you want the full bottom-up (well-founded) model of something, rather than just queries to that model

* Program analysis -- types, shapes, etc for the entire program (module)

* RDF inferencing -- want full set of triples implied by a page

* Highly connected OWL ontologies

* Consider a query ?- sg(X,Y) to

```
sg(X,X)
sg(X,Y):- par(X,Z),sg(Z,Z1),par(Y,Z1).
```

* As discussed, this will give a number of subgoals sg(X,Y), sg(1,Y), sg(2,Y),...

* If call variance is used, the table for `sg(1,Y)` cannot reuse the information for `sg(X,Y)` because the two terms are not variants

* If call subsumption is used, then `sg(1,Y)` can reuse the information from `sg(X,Y)`, which will reduce space and time.

* In XSB (and other systems) the declaration :- `table sg/2` means to table using call variance

* The declaration :- `table sg/2 as subsumptive` specifies call subsumption. Call subsumption can also be set up as the default tabling mode by default.

A Fragment of the OWL Wine Ontology

```
<owl:Class rdf:ID="PinotBlanc">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasColor" />
      <owl:hasValue rdf:resource="#White" />
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Wine" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape" />
      <owl:hasValue rdf:resource="#PinotBlancGrape" />
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape" />
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</
owl:maxCardinality>
    </owl:Restriction>
  </owl:intersectionOf>
 </owl:Class>
```

The ontology is translated by KAON2 to a definite program with about 1000 clauses

```
pinotblanc(X) :- q24(X).
pinotblanc(X) :- pinotblanc(Y),kaon2equal(X, Y).
pinotblanc(X) :- wine(X),madefromgrape(X, Y),ot____nom21(Y).

madefromgrape(Y, X) :- madeintowine(X, Y).
madefromgrape(X, X) :- riesling(X),kaon2namedobjects(X).
madefromgrape(X, X) :- wine(X),kaon2namedobjects(X).
% 18 others

wine(X) :- q14(X).
wine(X) :- texaswine(X).
% 24 others
wine(X) :- q24(X).
% 31 others

q24(X) :- pinotblanc(X).
q24(X) :- muscadet(X).
q24(X) :- q24(Y),kaon2equal(X, Y).
```

Regardless of whether this program can be optimized, it is highly recursive

pinotblanc(yellowTail) depends on
    pinotblank(X) depends on wine(X)
  and on
    wine(yellowTail)

Nearly every concept depends on nearly every other concept (more or less)

Each predicate is called with multiple instantiations

* When using call variance to evaluate a query XSB throws a memory error

* When using call subsumption to evaluate a query, XSB terminates

* The time is comparable with ontoBroker and is much faster than some ASP systems (Liang et. al 2009)

* The program can use the declaration :- `autotable` and default call subsumption. However analysis tools are needed to help decide whether to use call subsumption or call variance.

* WFS cannot fully evaluate *all* ontologies -- a theorem prover may be needed for that

Tabling Pattern 8: Evaluation of generated recursive code through autotable (with or without call subsumption)

-- Wine ontology translated to logic programming clauses

Outline

- "Vanilla" tabling -- query evaluation for definite programs
- Tabled negation for the well-founded and stable Semantics
- Beyond query evaluation -- call subsumption for (sub-)model generation
- *Beyond* ***t,f,u*** *-- answer subsumption for quantitative and para-consistent programming*
- Progress on multi-threading and parallelism

* When a new answer *A* is derived for a subgoal *S*, it is added to the table for *S* only if *A* is not a variant of some other answer for *S*

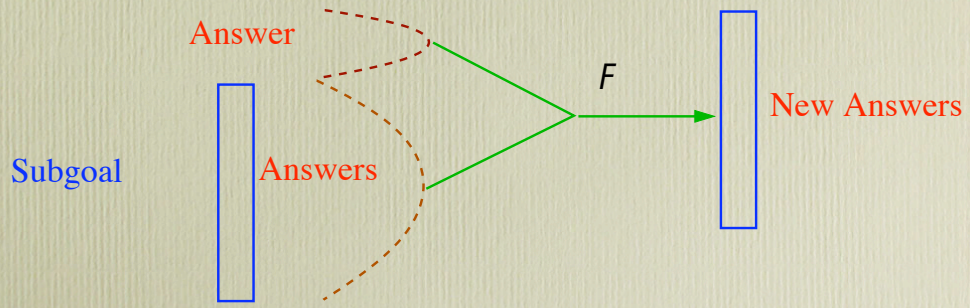
* On the other hand, you might add an answer only if it were not s*ubsumed* by some other answer for *S*

* When talking about the lattice of terms, this does not seem to be very useful

* However, the idea could be generalized to other partial orders or lattices

* In general, apply a function F to a set of answers to get a new set of answers

* Or, incrementally, apply F to a new answer plus a set of answers



* In a simple case F = *keep all maximal answers w.r.t. a given partial order*

* But this is just what we did with preferences
  -- Operationally, rather than maintaining a set of answers which we destructively update, we only derive preferred answers
  -- A reflexive partial order gives rise to undefined truth values

* Consider a (small!) portion of an ontology used by XSB, Inc on a project to help the US Government find sources for medical supplies (this can be thought of as monotonic inheritance as opposed to the non-monotonic inheritance of Nixon)

* Code on next page is a simplified syntax of XSB's Coherent Description Framework library (used on the project)

* A query about the material of a DemaTechPGA suture should return polyglyconicAcid

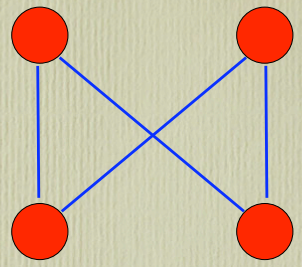
* A query about the material of some other suture for which no information is known should return absSutMaterial

---

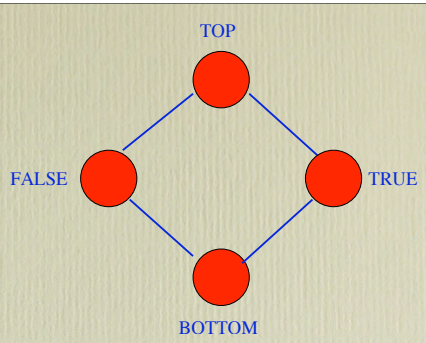
```
% Classes
hasAttr(absSutPart,hasMaterial,absSutMaterial)
maxAttr(absSutPart,hasMaterial,absSutMaterial,1)

isa(material,domainTypes)
  isa(absSutMaterial,material)
    isa(catgut,absSutMaterial)              % obsolete in US
    isa(vicryl,absSutMaterial)
    isa(vicryl_rapide,absSutMaterial)
    isa(monocryl,absSutMaterial)
    isa(polyglyconicAcid,absSutMaterial)
    :

% Objects
object_hasAttr('DemaTechPGA',hasMaterial,polyglyconicAcid)
```



vs.



* When maintaining ordering over a lattice keeping (or deriving) only maximal answers is no longer sifficient

* Lattices may require destructive operations on the answers

* If one `p(a):true` and `p(a):false` are both derived, then the table should only contain `p(a):top` -- which wasn't directly derived

* Consider a model of quantitative degrees of belief [van Emden '86]. An annotated atom $A:[E_T,E_F]$ is an atom A is annotated with
  -- E_T, a number between 0 and 1 indicating a measure of evidence that A is true; and
  -- E_F, a number between 0 and 1 indicating that A is false.
  -- $[E_T\ E_F]$ *join* $[E_T\ E_F] = [max(E_T\ E_T), max(E_F\ E_F)]$

* Resolution for these annotated literals can be defined. The main idea is:
  - A goal $A:[E_T,E_F]$ is true in an interpretation $\mathcal{J}$ of a program P if there is are rules

$$A^I:[E^I_T,E_{IF}] \text{ :- } Body^I$$

  in P such that each A^I unifies with A, each $Body^I$ is true in $\mathcal{J}$, $[E'_T,E'_F] = join\ [E^I_T,E^I_F]$, $E'_T >= E_T$ and $F_T =< F'_T$

* Generalizing this approach to upper semi-lattices, you get *Generalized Annotated Programs* (GAPs) that can model many kinds of quantitative, paraconsistent, and temporal reasoning.

* From our perspective, GAPs can be implemented using answer subsumption. To illustrate on ground programs, when an answer $A{:}[an_{new}]$ is derived:

  -- Add $A{:}[an_{new}]$ if the table does not have an answer with substitution A; or

  -- Add $A{:}[an_{join}]$ -- the join of an_{new} and an_{old} where $A{:}[an_{old}]$ is the answer for A in the table.

* This formalism is similar to others, such as Residuated Lattice Programs

* XSB has a meta-interpreter for stratified GAP's in its gap library

* This is a general approach that can be applied to other domains

* Time or probability lattices
  -- join may be intersection of two intervals where the partial order indicates a precision of knowledge
  -- can help in implementing Hybrid Probabilistic Programs, perhaps Logic Programs with Annotated Disjunctions

* Bilattices of knowledge and truth

* How can answer subsumption be used for process logics?

* Reachability in a (finite) elementary net is clearly decidable: since each place can hold at most 1 token there are a finite number of states

* General Petri nets (Place-Transition Nets) can have any number of tokens in a place (although the tokens are not distinguished)

* Reachability in Place-Transition nets is still decidable through a method called ω-sequences

    -- Define a partial order $\geq_\omega$ on configurations where $C \geq_\omega C'$ if C and C' have tokens in the same set Pl of places, C has at least as many tokens as C' for all places, and for some non-empty $Pl_{sub} \subseteq Pl$ places C has strictly more tokens than C'

    -- If this happens, abstract the tokens in each place in Pl_{sub} to have the special token ω (which is greater than any integer)

```
% Program to determine omega abstraction.
:- table reachable(_,gte_omega/3-bottom/1).
reachable(InConf,NewConf):-
  reachable(InConf,Conf),
  hasTransition(Conf,NewConf).
reachable(InConf,NewConf):-
  hasTransition(InConf,NewConf).

hasTransition(Conf,NewConf):-
  get_trans_for_conf(Conf,AllTrans),
  member(Trans,AllTrans),
  apply_trans_to_conf(Trans,Conf,NewConf).

get_trans_for_conf(Conf,Flattrans):-
  get_trans_for_conf_1(Conf,Conf,Trans),
  flatten(Trans,Flattrans).

get_trans_for_conf_1([],_Conf,[]).
get_trans_for_conf_1([H|T],Conf,[Trans1|RT]):-
  findall(trans([H|In],Out,Tran),trans([H|In],Out,Tran),Trans),
  check_concession(Trans,Conf,Trans1),
  get_trans_for_conf_1(T,Conf,RT).

check_concession([],_,[]).
check_concession([trans(In,Out,Name)|T],Input,[trans(In,Out,Name)|T1]):-
  ord_subset(In,Input),
  ord_disjoint(Out,Input),!,
  check_concession(T,Input,T1).
check_concession([_Trans|T],Input,T1):-
  check_concession(T,Input,T1).

apply_trans_to_conf(trans(In,Out_Name),Conf,NewConf):-
  ord_subtract(Conf,In,Diff),
  flatsort([Out|Diff],Temp),
```

* table reachable(_,gte_omega/3) indicates that
  - reachable/2 is tabled
  - gte_omega/3 is a join operator (on configurations)

* This is compiled into an XSB predicate filterReduce/3
  -- Its only in the CVS version of XSB, and we're still working out the kinks.

* If p/3 is tabled, then a constrained goal p(X,Y,Z): X > Z
can also be tabled.

* Constrained answers may also be returned, such as
p(X,Y,Z): X > Z + Y, Y > 0

* In XSB, attributed variables are copied into and out of tables as any other term

* Subsumption works for constrained goals in an analagous manner.
p(X): X > 2 subsumes p(X): X > 3.
    -- In case this is confusing, note that the set X > 2 is smaller than the set X > 3, so that any answer to X > 2 is also an answer to X > 3

* Answer subsumption with constraints has not yet been fully tested in XSB

* Constrained Variables in calls are not presently handled in call subsumption.

* Many Constraint Logic Programs do not benefit from tabling, as the logic program is used primarily to set up the constraints.

* Tabling can help search through a state space where the states can be labelled with constraints.

* Examples in natural language analysis, program analysis, verification, ILP

* Another example concerns a type of Colored Petri Net which generalizes place-transition nets by allowing tokens to be distinguished

* Rather than collecting tokens, a transition collects constraints until its constraint set entails a formula (and does not fire otherwise). Once the constraint fires, new constraints may be applied to the resulting configuration.

```
% Program to determine reachability of an elementary net
:- table reachable/2.
reachable(InConf,NewConf):-
  reachable(InConf,Conf),
  hasTransition(Conf,NewConf).
reachable(InConf,NewConf):-
  hasTransition(InConf,NewConf).

hasTransition(Conf,NewConf):-
  get_trans_for_conf(Conf,AllTrans),
  member(Trans,AllTrans),
  apply_trans_to_conf(Trans,Conf,NewConf).

get_trans_for_conf(Conf,Flattrans):-
  get_trans_for_conf_1(Conf,Conf,Trans),
  flatten(Trans,Flattrans).

get_trans_for_conf_1([],_Conf,[]).
get_trans_for_conf_1([H|T],Conf,[Trans1|RT]):-
  findall(trans([H|In],Out,Tran),trans([H|In],Out,Tran),Trans),
  check_concession(Trans,Conf,Trans1),
  get_trans_for_conf_1(T,Conf,RT).

check_concession([],_,[]).
check_concession([trans(In,Out,Name)|T],Input,[trans(In,Out,Name)|T1]):-
  ord_subset(In,Input),
  ord_disjoint(Out,Input),!,
  check_concession(T,Input,T1).
check_concession([_Trans|T],Input,T1):-
  check_concession(T,Input,T1).

apply_trans_to_conf(trans(In,Out_Name),Conf,NewConf):-
  ord_subtract(Conf,In,Diff),
  flatsort([Out|Diff],Temp),
```

The original

```
apply_trans_to_conf(trans(In,Out_Name),Conf,NewConf):-
  ord_subtract(Conf,In,Diff),
  flatsort([Out|Diff],Temp)
```

is rewritten to

```
apply_trans_to_conf(trans(In,Entailment,Out),Conf,NewConf):-
    unify_for_entailment(In,Conf,MidConf),
    entailed(Entailment),
    call_new_constraints(Out,OutPlaces),
    flatsort([OutPlaces|MidConf],NewConf).
```

unify_for_entailment(In,Conf,MidConf) unifies the variables in the configuration with the variables in the transition

Tabling Pattern 9: Tabled preferences to return only the best answers according to a partial order

-- Medical Parts Ontology

Tabling Pattern 10: Answer subsumption for paraconsistent and quantitative reasoning

Tabling Pattern 11: Tabling to explore a state space where constraints are associated with states

-- Constraint-based analysis of security protocols (Sarna-Starosta)

Multi-threading

Each XSB thread has its own facilities for creating and releasing private tables, and reclaiming their space

-- Private threads have all functionality discussed here including their own private version smodels

-- Answer Subsumption requires a synchronization mechanism that does not currently scale

XSB threads can also communicate through message queues as can SWI and YAP (Ciao has a similar mechanism)

Tables can also be shared among threads in XSB to support concurrency (work is being done to have them support parallelism)

-- No call subsumption with shared tables yet.

State of Implementation

* Definite Tabling: Stable

* Tabling for WFS: Stable

* XASP: Stable but persnickety

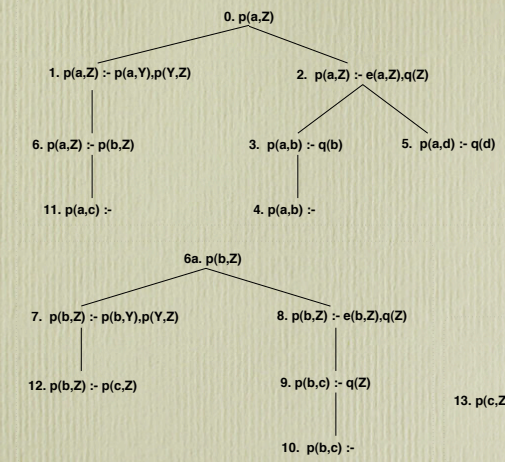
* Call Subsumption: Stable (recently extended to WFS, but extension has been well tested)

* Multi-threading: Reasonably stable for reasonable programs :-)
This includes private and shared tables

* Answer Subsumption: Reasonably stable, but still hard to use

* Tabled Constraints: Attributed variables have been used for a Ph. D. thesis. Large projects using tabling with CLP(R) or CHR could uncover bugs

Program, Forest, and Table for the query ?- p(a,Z).



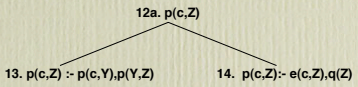

```
:- table p/2.

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

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

q(a).  q(b).  q(c).
```

Subgoal	Answers	State
p(a,Z)	p(a,b) p(a,c)	*Incomplete*
p(b,Z)	p(b,c)	*Incomplete*
p(c,Z)		*Incomplete*



This example uses double recursion (2 occurrences of a/2 in the body). Tabling is more efficient with left recursion.