Tabled Logic Programming and Deduction

Terrance Swift

SUNY Stony Brook

The XSB Group

- Active Members
  - Faculty
    - Bart Demoen (KU Leuven)
    - Michael Kifer (SUNY Stony Brook)
    - C.R. Ramakrishnan (SUNY Stony Brook)
    - I.V. Ramakrishnan (SUNY Stony Brook)
    - Terrance Swift (SUNY Stony Brook)
    - David S. Warren (SUNY Stony Brook)
  - Current Students
    - Ernie Johnson (SUNY Stony Brook)
    - Rui Marques (Universidade Nova de Lisboa)
    - Luis Pias de Castro (SUNY Stony Brook)
    - Guizhen Yang (SUNY Stony Brook)
- Other Contributors
  - Baoqiu Cui
  - Juliana Freire
  - Prasad Rao
  - Kostis Sagonas

Problems with (Constraint) Logic Programming

- Lack of termination;
- Exponential complexity for terminating datalog programs;
- Problems with negation;
- Difficulties in supporting quantitative or multi-valued logics.

Termination and Complexity

```
:- table rewrite_star/2.
rewrite_star(A,A).
rewrite_star(A,C):-
   rewrite_star(A,B),
   rewrite(B,C).

rewrite(A,B):-
   subterm(A,B1,Pos),      % B1 is a subterm of A
                           % at position Pos
   trans(B1,B2),
   replace(Pos,A,B2,B).    % replace the subterm at position
                           % Pos in A with B2, producing B

trans(plus(X,Y),plus(Y,X)).
```

- Prolog will not terminate for cycles
- Prolog transitive closure is proportional to the number of *paths* in a graph.

Problems with Negation

- Prolog *negation-by-failure* is based on a 3-valued completion semantics;
- Other semantics, such as Well-founded Semantics may be useful.

```
normalized(Term):-
        not transformable(Term).

transformable(Term):-
        rewrite_star(Term,Term1),
        Term \== Term1.
```

- Better handling of negation also allows mixture of *explicit* negation and negation-by-failure.

```
good_topic:- interesting,
     not published.

good_topic:- interesting,
     -published.
```

Properties of Tabled Logic Programs

Many formulations of TLP.

- SLG resolution is one such formulation:
- is ideally complete for the well-founded semantics.
- terminates for programs with *bounded term size.*
- is quadratic for wfs; quasi-linear for definite programs*.
- separates tabling operations from search strategy.

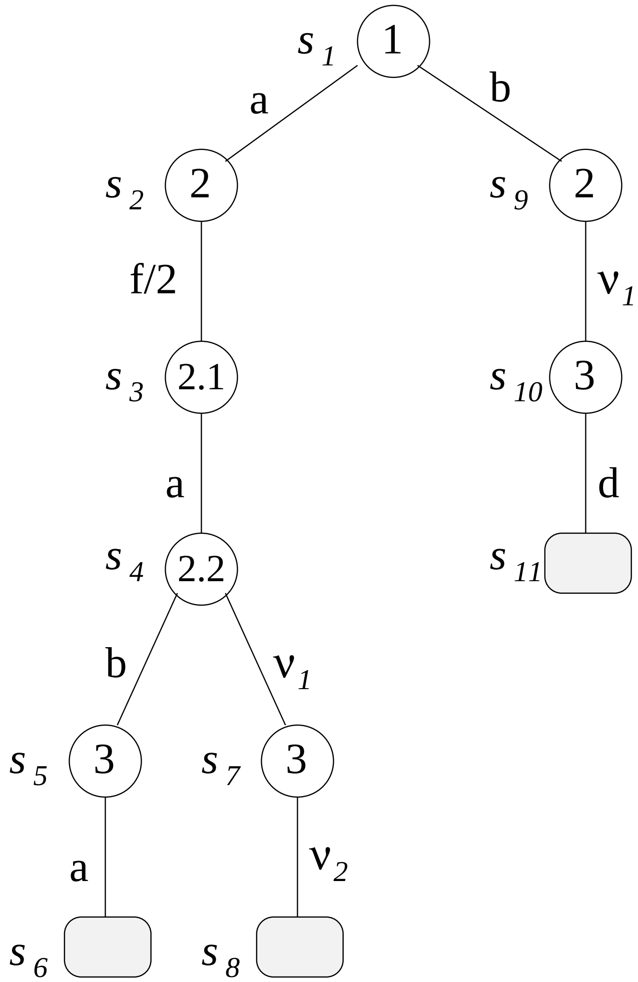
Properties of Tabled Logic Programs

- Several implementations of TLP.
  - Deductive databases and Prolog-style systems
- XSB (Prolog with Tabling and Constraints)
  - Efficient implementation of SLG
    - (a few compromises for efficiency)
  - Search strategies optimized for finding first answer and for finding best answer;
    - (fairness not built into engine)
  - Provides trie-based indexing of terms
  - Provides libraries for implementing tabled aggregation
  - Many other libraries and interfaces
    - WWW capabilities,
    - database interfaces,
    - object-oriented interfaces.
    - Smodels

Trie-based indexing of terms

- Store variant, non-linear terms via a left-to-right traversal.
- Used in tabling and as an option for assert
- In tabling, stores only the substitution for a call

```
p(a,f(a,b),a).
p(a,f(a,X),Y).
p(b,X,d).
```



Quantitative and Multi-valued Logics

- Simple example illustrates the general principle:

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

Quantitative and Multi-valued Logics

- Solution is to retain only "preferred" answers:

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


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

- For partial orders, need to perform a join explicitly:

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

Quantitative and Multi-valued Logics

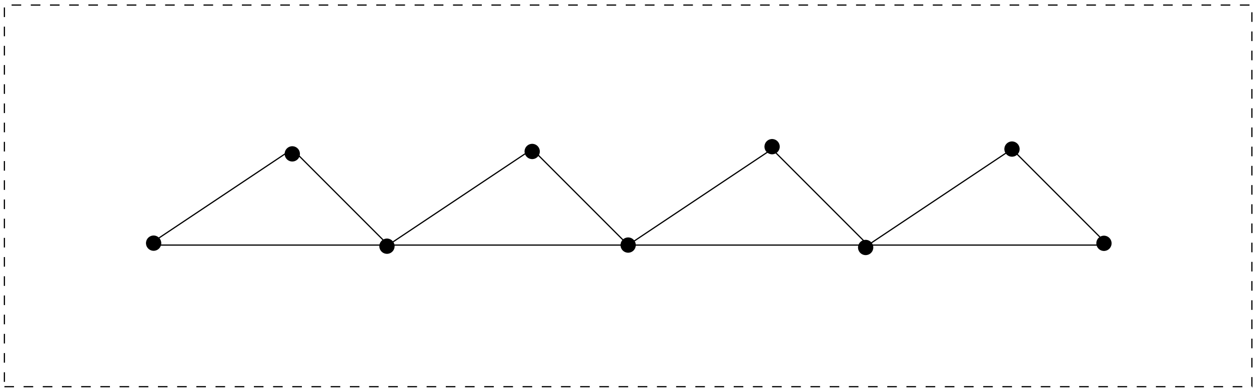
Simplified presentation, but can implement:

- Preference Logic Programs
  - Useful for grammars, data mining
- Annotated Logic Programs
  - Multi-valued programs, e.g. bilattice logic programs
  - Certainty Measures, e.g. Probabilistic Logic Programs; Possibilistic Logic Programs; Fuzzy Logic Programs.
- In actuality may be implemented through negation or through tabled aggregation.

Search Strategies

XSB has two search strategies:

- Batched evaluation:
  - Reflects Prologs evaluation strategy
  - Useful for finding *first* answer to a query
- Local evaluation
  - Fully evaluates (mutually dependant sets of) subqueries before returning any answers
  - Useful for finding *best* answer to a subquery.



Example: Modal-μ Calculus

- Branching time temporal logic obtained from a modal logic.
  - structure underlying modal logic represents a process system as formulated in CCS, π-calculus, etc.
  - reachable states modelled by boxes, diamonds, gfp, lfp operators
- Expressive for non real-time systems (but can be difficult to understand)
- Example: $\nu Z.\mu Y.[a]((Q \wedge Z) \vee Y)$
  - "On all a-labelled paths, Q holds infinately often"
  - Representable by Prolog terms
    - `fDef(nu,gfp(box(a,or(and(prop(q),form(nu)),form(mu)))))`
    - `fDef(mu,lfp(box(a,or(and(prop(q),form(nu)),form(mu)))))`
- Invented by Kozen, presentation following sequent-style formulation of Sterling, Walker, 93.

Example: Modal-μ Calculus

$$\frac{s \vdash_\Delta \neg\neg A}{s \vdash_\Delta A} \qquad \frac{s \vdash_\Delta A \wedge B}{s \vdash_\Delta A, s \vdash_\Delta B}$$

$$\frac{s \vdash_\Delta \neg(A \wedge B)}{s \vdash_\Delta A} \quad \frac{s \vdash_\Delta \neg(A \wedge B)}{s \vdash_\Delta B}$$

$$\frac{s \vdash_\Delta [a]A}{s_1 \vdash_\Delta A, \ldots, s_n \vdash_\Delta A} \; \{s_1, \ldots, s_n\} = \{s' | s \xrightarrow{a} s'\}$$

$$\frac{s \vdash_\Delta \neg[a]A}{s_1 \vdash_\Delta \neg A} \; s \xrightarrow{a} s'$$

$$\frac{s \vdash_\Delta \nu Z.A}{s_1 \vdash_{\Delta'} U} \; \Delta' \text{ is } \Delta \cdot U = \nu Z.A$$

$$\frac{s \vdash_\Delta U}{s_1 \vdash_\Delta A[Z := U]} \; member((U = \nu Z.A), \Delta)$$

$$\frac{s \vdash_\Delta \mu Z.A}{s_1 \vdash_\Delta A} \; \Delta(U) \neq \mu Z.A$$

Example: Modal μ-calculus

Sequent-style notation, but...

- Box-style notation leads to an indefinite conjunction
- Need to maintain definition lists (Δs)

Example Modal μ-calculus

- Implements MMC interpreter in about 150 lines of code. (see `http://www.cs.sunysb.edu/~tswift`)

$$\frac{s \vdash_{\Delta} A \wedge B}{s \vdash_{\Delta} A, s \vdash_{\Delta} B}$$

```
:- index(models/3,2).

models(State_s, and(X_1, X_2),Dl):-
   models(State_s, X_1,Dl),
   models(State_s, X_2,Dl).
```

Example Modal μ-calculus

- Uses Prolog constructs, like `findall/3`;

$$\frac{s \vdash_{\Delta} [a]A}{s_1 \vdash_{\Delta} A, \ldots, s_n \vdash_{\Delta} A} \; \{s_1, \ldots, s_n\} = \{s' | s \xrightarrow{a} s'\}$$

```
models(State_s, box(Act_a, X),Dl):-
   findall(State_t,
               trans(State_s,Act_a,State_t),
                    State_ts),
   list_model(State_ts, X,Dl).
```

Example Modal μ-calculus

$$\frac{s \vdash_\Delta \neg A}{s \nvdash_\Delta A}$$

```
models(State_s, not(X_1),Dl):-
   not_models(State_s, X_1,Dl).
```

$$\frac{s \nvdash_\Delta \neg(A \wedge B)}{s \nvdash_\Delta A} \quad \frac{s \nvdash_\Delta \neg(A \wedge B)}{s \nvdash_\Delta B}$$

```
not_models(State_s, and(X_1, X_2),Dl) :-
   not_models(State_s, X_1,Dl)
   ;  not_models(State_s, X_2,Dl).
```

Example Modal μ-calculus

```
models(State_s, form(Name),Dl) :-
   fDef(Name,Y),
   rec_models(State_s,Name, Y,Dl).
```

$$\frac{s \vdash_\Delta \mu Z.A}{s_1 \vdash_\Delta A}\ \Delta(U) \neq \mu Z.A$$

```
rec_models(State_s, _Name, lfp(Z),Dl_in) :-
    tab_models(State_s, Z,Dl_in).

:- table tab_models/3.
tab_models(A,B,Dl_in) :-
    models(A,B,Dl_in).
```

Example Modal μ-calculus

$$\frac{s \vdash_{\Delta} \nu Z.A}{s_1 \vdash_{\Delta'} U} \; \Delta' \text{ is } \Delta \cdot U = \nu Z.A$$

$$\frac{s \vdash_{\Delta} U}{s_1 \vdash_{\Delta} A[Z := U]} \; member((U = \nu Z.A), \Delta)$$

```
rec_models(S, Name,gfp(Z),Dl_in) :-
   (memberchk(p(S,Name),Dl_in) ->
        true
        ;  append([p(S,Name)],Dl_in,Dl1),
           tab_models(S, Z,Dl1)).
```

Example Modal μ-calculus

- Tabling used to check for lfps, and to ensure similar paths are not re-explored.
- Programming task becomes writing a meta-interpreter;
- Meta-interpreters can be specialized by partial evaluation;
- A separate interpreter for alternation-free Modal mu-calculus uses XSB's negation;
- Leads to competetive model checkers;
- Much more can be found at LMC home page (`http://www.cs.sunysb.edu/~lmc`)

Current Work

- Implementing Abduction over the Well-founded Semantics
  - Abduction provides a way to *specify* rather than program.
- Multi-threaded and Distributed Evaluation
  - Threads/Processes communicate via tables
- Relevant Evaluations
  - Once an existential goal has obtained an answer, work on other parts of the derivation