From program to data

**Program:**  
```
tran/3, final/1

accept(String) :- steps(q0,String,Q), final(Q).
steps(Q,[]),Q).
steps(Q,[H|T],N) :- tran([Q,H,Qn), steps(Qn,T,N)).
```
From program to data  (predicates $\leftrightarrow$ lists)

**Program:**  tran/3, final/1

accept(String) :- steps(q0,String,Q), final(Q).

steps(Q, [], Q).

steps(Q, [H|T], N) :- tran([Q,H,Qn]), steps(Qn,T,N).

**Data:**  Tran, Final, q0

acc(String) :- setof([Q,X,N], tran(Q,X,N), Tran),
setof(Q, final(Q), Final),
accept(String, Tran, Final, q0).

finite automaton
From program to data (predicates $\mapsto$ lists)

**Program:**  

\[
\text{tran}/3, \ \text{final}/1 \\
\text{accept(String)} :\text{- steps}(\text{q0}, \text{String}, \text{Q}), \ \text{final}(\text{Q}). \\
\text{steps}(\text{Q}, [], \text{Q}). \\
\text{steps}(\text{Q}, [\text{H}|\text{T}], \text{N}) :\text{- tran}([\text{Q}, \text{H}, \text{Qn}), \ \text{steps}(\text{Qn}, \text{T}, \text{N}).
\]

**Data:**  

\[
\text{Tran, Final, q0} \\
\text{acc(String)} :\text{- setof}([\text{Q,X,N}], \ \text{tran}(\text{Q,X,N}), \ \text{Tran}), \ \\
\text{setof}(\text{Q}, \ \text{final}(\text{Q}), \ \text{Final}), \ \\
\text{accept(String, Tran,Final,q0).}
\]

finite automaton

\[
\text{accept}([], _, \text{Final}, \text{Q}) :\text{- member}(\text{Q,Final}). \\
\text{accept}([\text{H}|\text{T}], \text{Tran}, \text{Fi}, \text{Q}) :\text{- member}([\text{Q,H,N}], \ \text{Tran}), \ \\
\text{accept}(\text{T,Tran,Fi,N}).
\]
From accept/4 to search/1

accept([],_,Final,Q) :- member(Q,Final).

Problem: search state space for goal.
N.B. Finite automaton specifies initial state, goals & state space
+ String constrains moves.

accept([H|T],Tran,Fi,Q) :- member([Q,H,N],Tran),
accept(T,Tran,Fi,N).

search(Q) :- goal(Q).
search(Q) :- move(Q,N), search(N).
From accept/4 to search/1

accept([],_,Final,Q) :- member(Q,Final).

accept([H|T],Tran,Fi,Q) :- member([Q,H,N],Tran),
                          accept(T,Tran,Fi,N).

Problem: search state space for goal.

N.B. Finite automaton specifies initial state, goals & state space
    + String constrains moves.
From accept/4 to search/1

accept([],_,Final,Q) :- member(Q,Final).
  \[\text{goal(Q)}\]
accept([H|T],Tran,Fi,Q) :- member([Q,H,N],Tran),
  accept(T,Tran,Fi,N).

search(Q) :- goal(Q).
search(Q) :- move(Q,N), search(N).
From accept/4 to search/1

\[
\text{accept([],\_\_,Final,Q)} :\text{- member(Q,Final).}
\]

\[
\text{accept([H\mid T],Tran,Fi,Q)} :\text{- member([Q,H,N],Tran),}
\text{accept(T,Tran,Fi,N).}
\]

\[
\text{search(Q)} :\text{- goal(Q).}
\]

\[
\text{search(Q)} :\text{- move(Q,N), search(N).}
\]
accept([],_,Final,Q) :- member(Q,Final).
                     \begin{align*}
                        \text{goal}(Q) \\
                        \text{move}(Q,N) \end{align*}

accept([H|T],Tran,Fi,Q) :- member([Q,H,N],Tran),
                       \begin{align*}
                        \text{accept}(T,Tran,Fi,N). \\
                        \text{accept}(T,Tran,Fi,N). \end{align*}

search(Q) :- goal(Q).
search(Q) :- move(Q,N), search(N).
accept([],_,Final,Q) :- member(Q,Final).

accept([H|T],Tran,Fi,Q) :- member([Q,H,N],Tran),
    accept(T,Tran,Fi,N).

search(Q) :- goal(Q).
search(Q) :- move(Q,N), search(N).

Problem: search state space for goal.
accept([],_,Final,Q) :- member(Q,Final).
  goal(Q)

accept([H|T],Tran,Fi,Q) :-
  member([Q,H,N],Tran),
  accept(T,Tran,Fi,N).

search(Q) :- goal(Q).
search(Q) :- move(Q,N), search(N).

**Problem**: search state space for goal.

**N.B.** Finite automaton specifies initial state, goals & state space
  + String constrains moves.
Finite automaton as a finite model

Tran,Final,q0 ⇝ U, move_a/2, goal/1, q0

universe U is set of states given by q0 and Tran

setof(Q,(Q=q0; state(Tran,Q)), U).

state(Tran,Q) :- member([Q,_,_],Tran);
               member([_,_,Q],Tran).
Finite automaton as a finite model

Tran, Final, q0 ← U, move_a/2, goal/1, q0

universe U is set of states given by q0 and Tran

setof(Q,(Q=q0; state(Tran,Q)), U).

state(Tran,Q) :- member([Q,_,_],Tran);
member([_,_,Q],Tran).

predicates: goal/1, move_a/2 (a ∈ Σ)

goal(Q) :- final(Q). % member(Q,Final)
move_a(Q,N) :- tran(Q,a,N).
% member([Q,a,N],Tran)
Finite automaton as a finite model

Tran, Final, q0 \rightarrow U, \text{move}_a/2, \text{goal}/1, q0

universe U is set of states given by q0 and Tran

\text{setof}(Q, (Q=q0; \text{state}(\text{Tran},Q)), U).

\text{state}(\text{Tran},Q) :- \text{member}([Q,_,_],\text{Tran});
\quad \text{member}([_,_,Q],\text{Tran}).

predicates: \text{goal}/1, \text{move}_a/2 \ (a \in \Sigma)

\text{goal}(Q) :- \text{final}(Q). \quad \% \text{member}(Q,\text{Final})
\text{move}_a(Q,N) :- \text{tran}(Q,a,N).
\quad \% \text{member}([Q,a,N],\text{Tran})

Focus on models \langle U, R_1, \ldots, R_n, c_1, \ldots, c_m \rangle where U is a finite set, 
R_i is an \(n_i\)-ary relation on \(U\),

\[ R_i \subseteq U \times \cdots \times U \quad (\text{for } 1 \leq i \leq n) \]

\( n_i \) copies of \( U \)

and \( c_j \) is a member of \( U \) (for \( 1 \leq j \leq m \)).
Models from Datalog Knowledge Bases

Datalog KB \approx \text{declarative Prolog program with constants and predicates but NO functions of non-zero arity}

\mathcal{U} = \text{set of constants mentioned in KB}
Models from Datalog Knowledge Bases

Datalog KB $\approx$ declarative Prolog program with constants and predicates but NO functions of non-zero arity

$U = \text{set of constants mentioned in KB}$

Example from SWISH-Prolog (click here)

$U = \{\text{vincent, mia, marcellus, pumpkin, honey\_bunny}\}$
Models from Datalog Knowledge Bases

Datalog KB $\approx$ declarative Prolog program with constants and predicates but NO functions of non-zero arity

$U = \text{set of constants mentioned in KB}$

Example from SWISH-Prolog (click here)

$U = [\text{vincent, mia, marcellus, pumpkin, honey\_bunny}]$

Bound search: instantiate variables in $U$

?- loves(X,Y).

$\Rightarrow$

?- member(X,[vincent, mia, ..., honey\_bunny]),
    member(Y,[vincent, mia, ..., honey\_bunny]),
    loves(X,Y).

Complications from loops – e.g.

loves(X,Y) :- loves(Y,X).
Models from Datalog Knowledge Bases

Datalog KB \approx \text{declarative Prolog program with constants and predicates but NO functions of non-zero arity}

U = \text{set of constants mentioned in KB}

Example from SWISH-Prolog (click here)

U = [vincent, mia, marcellus, pumpkin, honey\_bunny]

Bound search: instantiate variables in \( U \)

?- loves(X,Y).

\Rightarrow

?- member(X,[vincent, mia, ..., honey\_bunny]),
   member(Y,[vincent, mia, ..., honey\_bunny]),
   loves(X,Y).

Complications from loops – e.g.

loves(X,Y) :- loves(Y,X).

[ CSU33061: Constraint satisfaction, Herbrand models ]
Strings as finite models

\[ abbc \rightsquigarrow \text{model} \]

\[ \langle D_4, [S], [P_a], [P_b], [P_c] \rangle \]
Strings as finite models

$abbc \leadsto \text{model}$

$\langle D_4, [S], [P_a], [P_b], [P_c] \rangle$

where

$D_4 := \{1, 2, 3, 4\}$

$[S] := \{\langle 1, 2 \rangle, \langle 2, 3 \rangle, \langle 3, 4 \rangle\}$

a binary relation symbol (successor) $S$

$(\exists x_1)(\exists x_2)(\exists x_3)(\exists x_4) \ S(x_1, x_2) \land S(x_2, x_3) \land S(x_3, x_4) \land \neg (\exists x)(S(x, x_1) \lor S(x_4, x))$
Strings as finite models

\( \text{abbc} \rightarrow \text{model} \)

\[ \langle D_4, [S], [P_a], [P_b], [P_c] \rangle \]

where

\[ D_4 := \{1, 2, 3, 4\} \]

\[ [S] := \{\langle 1, 2\rangle, \langle 2, 3\rangle, \langle 3, 4\rangle\} \]

\[ [P_a] := \{1\} \]

\[ [P_b] := \{2, 3\} \]

\[ [P_c] := \{4\} \]

a binary relation symbol (successor) \( S \) and

a unary relation symbol \( P_\sigma \) for each symbol \( \sigma \)

\[
(\exists x_1)(\exists x_2)(\exists x_3)(\exists x_4) \quad S(x_1, x_2) \land S(x_2, x_3) \land S(x_3, x_4) \land \\
\neg(\exists x)(S(x, x_1) \lor S(x_4, x)) \land \\
P_a(x_1) \land P_b(x_2) \land P_b(x_3) \land P_c(x_4)
\]
accept/4 as a relation between models?
accept\(\rightarrow\) as a relation between models?
reconstrue finite automaton as predicate logic sentence

\[
(\exists x)(\exists y)(P_a(x) \land P_b(y) \land \\
(\forall z)(\neg S(z, x) \land (z = x \lor P_b(z))))
\]
Procedural/declarative divide

accept/4 as a relation between models?
reconstrue finite automaton as predicate logic sentence

\[
(\exists x)(\exists y)(P_a(x) \land P_b(y)) \land \\
(\forall z)(\neg S(z, x) \land (z = x \lor P_b(z))))
\]

accept/4 as notion of satisfaction in predicate logic

program \quad = \quad \text{procedural}
\phantom{=}
\text{data} \quad = \quad \text{declarative}
Procedural/declarative divide

accept/4 as a relation between models?
reconstrue finite automaton as predicate logic sentence

\[(\exists x)(\exists y)(P_a(x) \land P_b(y) \land (\forall z)(\neg S(z, x) \land (z = x \lor P_b(z))))\]

accept/4 as notion of satisfaction in predicate logic

\[\text{program} \quad = \quad \text{procedural} \quad \frac{\text{data}}{\text{declarative}}\]

Logic programming is neither logic nor programming.

- Anonymous