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 $\rightsquigarrow$ 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 ⇝ 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

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

From accept/4 to search/1

$$\texttt{accept([],\_,Final,Q) :- }\underbrace{\texttt{member(Q,Final)}}_{\texttt{goal(Q)}}\texttt{.}$$

From accept/4 to search/1

$$\texttt{accept([],\_,Final,Q) :- }\underbrace{\texttt{member(Q,Final)}}_{\texttt{goal(Q)}}\texttt{.}$$

$$\begin{aligned}\texttt{accept([H|T],Tran,Fi,Q) :- }&\overbrace{\texttt{member([Q,H,N],Tran)}}^{\texttt{move(Q,N)}}\texttt{,}\\ &\texttt{accept(T,Tran,Fi,N).}\end{aligned}$$

From accept/4 to search/1

$$\texttt{accept([],\_,Final,Q) :- }\underbrace{\texttt{member(Q,Final)}}_{\texttt{goal(Q)}}\texttt{.}$$

$$\begin{aligned}\texttt{accept([H|T],Tran,Fi,Q) :- }&\overbrace{\texttt{member([Q,H,N],Tran)}}^{\texttt{move(Q,N)}}\texttt{,}\\ &\texttt{accept(T,Tran,Fi,N).}\end{aligned}$$

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

From accept/4 to search/1

$$\texttt{accept([],\_,Final,Q) :- }\underbrace{\texttt{member(Q,Final)}}_{\texttt{goal(Q)}}\texttt{.}$$

$$\texttt{accept([H|T],Tran,Fi,Q) :- }\overbrace{\texttt{member([Q,H,N],Tran)}}^{\texttt{move(Q,N)}}\texttt{,}$$
$$\texttt{accept(T,Tran,Fi,N).}$$

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

From accept/4 to search/1

$$\texttt{accept([],\_,Final,Q) :- }\underbrace{\texttt{member(Q,Final)}}_{\texttt{goal(Q)}}\texttt{.}$$

$$\begin{aligned}\texttt{accept([H|T],Tran,Fi,Q) :- }&\overbrace{\texttt{member([Q,H,N],Tran)}}^{\color{red}\texttt{move(Q,N)}}\texttt{,}\\ &\texttt{accept(T,Tran,Fi,N).}\end{aligned}$$

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

From accept/4 to search/1

$$\texttt{accept([],\_,Final,Q) :- }\underbrace{\texttt{member(Q,Final)}}_{\texttt{goal(Q)}}\texttt{.}$$

$$\begin{aligned}\texttt{accept([H|T],Tran,Fi,Q) :- }&\overbrace{\texttt{member([Q,H,N],Tran)}}^{\texttt{move(Q,N)}}\texttt{,}\\ &\texttt{accept(T,Tran,Fi,N).}\end{aligned}$$

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

Problem: search state space for goal.

From accept/4 to search/1

$$\texttt{accept([],\_,Final,Q) :- }\underbrace{\texttt{member(Q,Final)}}_{\texttt{goal(Q)}}\texttt{.}$$

$$\begin{aligned}\texttt{accept([H|T],Tran,Fi,Q) :- }&\overbrace{\texttt{member([Q,H,N],Tran)}}^{\texttt{move(Q,N)}}\texttt{,}\\ &\texttt{accept(T,Tran,Fi,N).}\end{aligned}$$

```
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 $\rightsquigarrow$ 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` $\rightsquigarrow$ `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` $(a \in \Sigma)$

```
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  ⇝  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` $(a \in \Sigma)$

```
goal(Q) :- final(Q).    % member(Q,Final)
move_a(Q,N) :- tran(Q,a,N).
               % member([Q,a,N],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,

$$R_i \subseteq \underbrace{U \times \cdots \times U}_{n_i \text{ copies of } U} \quad (\text{for } 1 \leq i \leq n)$$

and c_j is a member of U (for $1 \leq j \leq m$).

Models from Datalog Knowledge Bases

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

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

Example from SWISH-Prolog (click here)

```
U = [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` $=$ 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).
```

$\leadsto$

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

`U` = 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).
```

$\rightsquigarrow$

```
?- 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 $\rightsquigarrow$ model

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

Strings as finite models

$abbc \quad \rightsquigarrow \quad$ 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

$$(\exists x_1)(\exists x_2)(\exists x_3)(\exists x_4) \quad S(x_1, x_2) \wedge S(x_2, x_3) \wedge S(x_3, x_4) \wedge \neg(\exists x)(S(x, x_1) \vee S(x_4, x))$$

Strings as finite models

abbc $\rightsquigarrow$ model

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

where

$$\begin{aligned} D_4 &:= \{1,2,3,4\} \\ [\![S]\!] &:= \{\langle 1,2\rangle, \langle 2,3\rangle, \langle 3,4\rangle\} \\ [\![P_a]\!] &:= \{1\} \\ [\![P_b]\!] &:= \{2,3\} \\ [\![P_c]\!] &:= \{4\} \end{aligned}$$

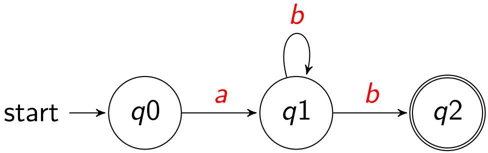
a binary relation symbol (successor) S and
a unary relation symbol P_σ for each symbol σ

$$\begin{aligned} (\exists x_1)(\exists x_2)(\exists x_3)(\exists x_4) \quad & S(x_1,x_2) \wedge S(x_2,x_3) \wedge S(x_3,x_4) \ \wedge \\ & \neg(\exists x)(S(x,x_1) \vee S(x_4,x)) \ \wedge \\ & P_a(x_1) \wedge P_b(x_2) \wedge P_b(x_3) \wedge P_c(x_4) \end{aligned}$$

`accept/4` as a relation between models?

`accept/4` as a relation between models?

reconstrue finite automaton as predicate logic sentence



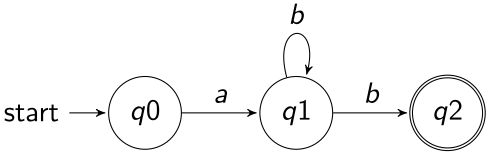
ab^*b

$$(\exists x)(\exists y)(P_a(x) \wedge P_b(y) \wedge \\ (\forall z)(\neg S(z,x) \wedge (z = x \vee P_b(z))))$$

Procedural/declarative divide

`accept/4` as a relation between models?

reconstrue finite automaton as predicate logic sentence



ab^*b

$$(\exists x)(\exists y)(P_a(x) \wedge P_b(y) \wedge$$
$$(\forall z)(\neg S(z,x) \wedge (z = x \vee P_b(z))))$$

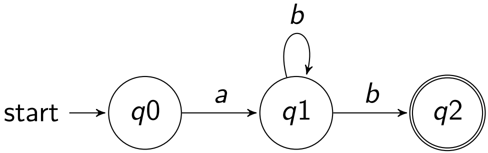
`accept/4` as notion of satisfaction in predicate logic

$$\frac{\text{program}}{\text{data}} = \frac{\text{procedural}}{\text{declarative}}$$

Procedural/declarative divide

`accept/4` as a relation between models?

reconstrue finite automaton as predicate logic sentence



ab^*b

$$(\exists x)(\exists y)(P_a(x) \wedge P_b(y) \wedge \\ (\forall z)(\neg S(z,x) \wedge (z = x \vee P_b(z))))$$

`accept/4` as notion of satisfaction in predicate logic

$$\frac{\text{program}}{\text{data}} = \frac{\text{procedural}}{\text{declarative}}$$

Logic programming is neither logic nor programming.

\- Anonymous