Probabilistic CHR and an Application in Music

CHRiSM AND APOPCALESAPS

Jon Sneyers
September 2011

PART ONE

CHance Rules induce Statistical Models

CHRiSM 驰
Chance rules

- A *chance rule* is a CHR rule with a probability:
  \[ \text{prob} \ ? \ \text{khead} \ \text{\backslash rhead} \ \leftrightarrow \ \text{guard} \mid \text{body}. \]

- For example:
  \[ 0.4 \ ? \ \text{flu}(X), \ \text{friend}(X,Y) \ \Rightarrow \ \text{flu}(Y). \]

  "If X has the flu, and Y is a friend of X, then there is a 40% chance that Y also gets the flu."

- Every result from a query has some probability:
  \[ Q = \{ \text{flu}(\text{jon}), \ \text{friend}(\text{jon,thom}), \ \text{friend}(\text{thom,slim}) \} \]
  - \[ Q \rightarrow Q \quad 60\% \ \text{chance} \ (1 - 0.4) \]
  - \[ Q \rightarrow Q, \ \text{flu}(\text{thom}) \quad 24\% \ \text{chance} \ (0.4 \times 0.6) \]
  - \[ Q \rightarrow Q, \ \text{flu}(\text{thom}), \ \text{flu}(\text{slim}) \quad 16\% \ \text{chance} \ (0.4 \times 0.4) \]

Chance rules (2)

- The body of a chance rule can contain *probabilistic disjunctions*

- For example:
  \[ \text{coin\_flip} \ \leftrightarrow \ \text{head:0.5} \ ; \ \text{tail:0.5}. \]

  (you can also write “\text{coin\_flip} \ \leftrightarrow \ ?? \ \text{head} \ ; \ \text{tail}.”)

- One of the disjuncts is chosen at random (no backtracking)
PRISM

- CHriSM is implemented in CHR(PRISM)
- PRISM is a probabilistic extension of Prolog
- PRISM builtin: \texttt{msw(Experiment,Outcome)}
  - Define outcomes with \texttt{values/2}
  - Define probabilities with \texttt{set_sw/2}
- For example:
  
  \begin{verbatim}
  values(coin,[head,tail]).
  :- set_sw(coin, [0.5, 0.5]).
  coin_flip(Result) :- msw(coin,Result).
  \end{verbatim}

CHR(PRISM)

- CHriSM is translated to CHR(PRISM), e.g.
  
  \begin{verbatim}
  coin_flip <=> head:0.5 ; tail:0.5.
  \end{verbatim}

  is translated to something like this:

  \begin{verbatim}
  values(exp00, [1,2]).
  :- set_sw(exp00, [0.5, 0.5]).
  coin_flip <=> msw(exp00,X),
    (X=1 ->
      head
    ; % X=2
      tail
  ).
  \end{verbatim}

- You can also use \texttt{msw/2} directly in CHriSM
Sampling

- If you just execute a query, the random choices are made according to the probability distributions, so you are sampling the statistical model represented by the program.

- For example:

  ```prolog
  ?- sample coin_flip
  tail
  ?- sample coin_flip, coin_flip
  head, tail
  ``

- The `sample` keyword is optional.

Observations

- One sample gives a full observation, denoted with `<==>` (do not confuse with `=>=`)

  ```prolog
  coin_flip <=> tail
  coin_flip, coin_flip <=> head, tail
  ```

- If we can see only a part of the result, we have a partial observation, denoted with `==>`

  ```prolog
  flu(jon), friend(jon,thom),
  friend(thom,slim) ==> flu(slim)
  ```
Probability calculation

- The probability of an observation is the chance that a random sample agrees with the observation.

```prolog
?- prob coin_flip <==> head
```

The probability is 0.5

```prolog
?- prob coin_flip, coin_flip ===> head
```

The probability is 0.75

```prolog
?- prob flu(jon), friend(jon,thom), friend(thom,slim) ===> flu(thom)
```

The probability is 0.4

Unknown probabilities

- If the probabilities are not given as a number, they are unknown.
- Initially the probabilities are set to the uniform distribution.
- They can be learned from observations.
- For example, a biased coin with unknown bias:

```prolog
bias_flip <==>< my_coin ?? head ; tail.
```
Unknown probabilities

- If the probabilities are not given as a number, they are unknown
- Initially the probabilities are set to the uniform distribution
- They can be learned from observations
- For example, a biased coin with unknown bias:

  \[ \text{bias\_flip} \leftrightarrow \text{my\_coin} \?? \text{head} \; ; \; \text{tail}. \]

Experiment name
This can also be a term containing variables from the head and guard, as long as it is ground at runtime.

\[ \rightarrow \text{parametrized distributions} \]

Learning

- Given a list of observations, learning will try to set the probabilities to maximize the likelihood of the observations

  \[ ?- \text{learn}([(\text{bias\_flip} \leftrightarrow \text{head}),\]
  \[ (\text{bias\_flip} \leftrightarrow \text{head}),\]
  \[ (\text{bias\_flip} \leftrightarrow \text{tail})]),\]
  \[ \text{show\_sw}. \]

Switch choice my\_coin:

1 (p: 0.66667)  2 (p: 0.33333)
A more complicated example

- Rock-paper-scissors game
- Play for a number of rounds
- Players choose randomly
- Choice depends on:
  - The player
  - Player's previous choice
  - Opponent's previous choice

Rock-paper-scissors in CHRiSM

```prolog
:- chrism player/1, rounds/1, do/3, round/1, winner/2, win_count/2, done, overall_winner/1.

rounds(M) ==> round(1).
player(P) ==> do(nothing,0,P), win_count(P,0).

round(R), player(P), do(MyPrevMove,R1,P), do(OtherPrevMove,R1,_) ==> R1 is R-1 | strategy(P,MyPrevMove,OtherPrevMove) ?? do(rock,R,P) ; do(scissors,R,P) ; do(paper,R,P).
rounds(M) \ round(R) <= R<M -> R1 is R+1, round(R1) ; done.

do(rock,R,P1), do(scissors,R,P2) ==> winner(R,P1).
do(scissors,R,P1), do(paper,R,P2) ==> winner(R,P1).
do(paper,R,P1), do(rock,R,P2) ==> winner(R,P1).

winner(_,P) ==> win_count(P,1).
win_count(P,X), win_count(P,Y) <= XY is X+Y, win_count(P,XY).
done, win_count(P,X), win_count(Q,Y) ==> X>Y | overall_winner(P).
```
Example interaction (1)

?- sample player(peter),player(leslie),rounds(4).

player(peter),player(leslie),rounds(4)<===>winner(4,leslie),winner(2,peter),winner(1,leslie),player(leslie),player(peter),do(paper,4,leslie),do(rock,4,peter),do(paper,3,peter),do(paper,3,leslie),do(rock,2,leslie),do(paper,2,peter),do(paper,1,peter),do(scissors,1,leslie),do(nothing,0,leslie),do(nothing,0,peter),rounds(4),win_count(leslie,2),win_count(peter,1),done,overall_winner(leslie).

Example interaction (2)

?- show_sw.

Switch strategy(leslie,nothing,nothing): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(leslie,paper,paper): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(leslie,paper,rock): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(leslie,paper,scissors): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(leslie,rock,paper): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(leslie,rock,rock): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(leslie,rock,scissors): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(leslie,scissors,paper): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(leslie,scissors,rock): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(leslie,scissors,scissors): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(peter,nothing,nothing): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(peter,paper,paper): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(peter,paper,rock): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Switch strategy(peter,paper,scissors): unfixed_p: 1 (p: 0.333333) 2 (p: 0.333333) 3 (p: 0.333333)
Example interaction (3)

?- prob player(peter),player(leslie),rounds(4) ===> overall_winner(peter).
Probability of player(peter),player(leslie),rounds(4)===>overall_winner(peter) is: 0.3827160493

?- prob player(peter),player(leslie),rounds(4) ===> overall_winner(leslie).
Probability of player(peter),player(leslie),rounds(4)===>overall_winner(leslie) is: 0.3827160493

?- set_sw_all(strategy(peter,_,_), [0.8,0.1,0.1]). % Peter always prefers rock
?- set_sw_all(strategy(leslie,_,_),[0.1,0.8,0.1]). % Leslie always prefers scissors
?- show_sw.

Switch strategy(leslie,nothing,nothing): unfixed_p: 1 (p: 0.100000) 2 (p: 0.800000) 3 (p: 0.100000)
Switch strategy(leslie,paper,paper): unfixed_p: 1 (p: 0.100000) 2 (p: 0.800000) 3 (p: 0.100000)
Switch strategy(leslie,paper,rock): unfixed_p: 1 (p: 0.100000) 2 (p: 0.800000) 3 (p: 0.100000)
Switch strategy(leslie,paper,scissors): unfixed_p: 1 (p: 0.100000) 2 (p: 0.800000) 3 (p: 0.100000)
Switch strategy(leslie,rock,paper): unfixed_p: 1 (p: 0.100000) 2 (p: 0.800000) 3 (p: 0.100000)

Switch strategy(peter,nothing,nothing): unfixed_p: 1 (p: 0.800000) 2 (p: 0.100000) 3 (p: 0.100000)
Switch strategy(peter,paper,paper): unfixed_p: 1 (p: 0.800000) 2 (p: 0.100000) 3 (p: 0.100000)

Example interaction (4)

?- set_sw_all(strategy(peter,_,_), [0.8,0.1,0.1]).
?- set_sw_all(strategy(leslie,_,_),[0.1,0.8,0.1]).

?- prob player(peter),player(leslie),rounds(4) ===> overall_winner(peter).
Probability of player(peter),player(leslie),rounds(4)===>overall_winner(peter) is: 0.8203113599

?- prob player(peter),player(leslie),rounds(4) ===> overall_winner(leslie).
Probability of player(peter),player(leslie),rounds(4)===>overall_winner(leslie) is: 0.0644094300

?- fix_sw(strategy(peter,_,_)).
?- learn([[10 times player(peter),player(leslie),rounds(4) ===> overall_winner(peter)),
          (90 times player(peter),player(leslie),rounds(4) ===> overall_winner(leslie))
          ]).

#goals: 0(2)
Exporting switch information to the EM routine ... done
#em-its: 0........100..........200..........300....(360) (Converged: -36.956226290)
Statistics on learning:
  Graph size: 40176
  Number of switches: 20
  Number of switch instances: 60
  Number of iterations: 360
  Final log likelihood: -36.956226290
  Total learning time: 1.600 seconds
  Explanation search time: 1.530 seconds
  Total table space used: 1655200 bytes
Type show_sw or show_sw_b to show the probability distributions.
Example interaction (5)

?- prob player(peter),player(leslie),rounds(4) ==> overall_winner(peter).
Probability of player(peter),player(leslie),rounds(4)==>overall_winner(peter) is: 0.820311359999988

?- prob player(peter),player(leslie),rounds(4) ==> overall_winner(leslie).
Probability of player(peter),player(leslie),rounds(4)==>overall_winner(leslie) is: 0.064409430000000

?- fix_sw(strategy(peter,_,_)).

?- learn([[10 times player(peter),player(leslie),rounds(4) ==> overall_winner(peter)),
(90 times player(peter),player(leslie),rounds(4) ==> overall_winner(leslie))
]].

[...]

?- prob player(peter),player(leslie),rounds(4) ==> overall_winner(peter).
Probability of player(peter),player(leslie),rounds(4)==>overall_winner(peter) is: 0.1147121539

?- prob player(peter),player(leslie),rounds(4) ==> overall_winner(leslie).
Probability of player(peter),player(leslie),rounds(4)==>overall_winner(leslie) is: 0.8436373199

PART TWO

Automatic POP Composer
And LEArner of ParameterS

APOPCALES

APOPCALES
Input constraints

% inputs

:- chrism measures(+int), meter(+int,+duration), key(+key), tempo(+int), voice(+voice), shortest_duration(+voice, duration), range(+voice, +note,+int,+note,+int), max_jump(+voice,+int), instrument(+voice,+), ...

:- chr_type key ---> major ; minor.
:- chr_type voice ---> melody ; chords ; bass ; drums.
:- chr_type note ---> c ; d ; e ; f ; g ; a ; b ; cis; dis ; fis ; gis ; ais ; r.
:- chr_type duration ---> 2 ; 4 ; 8 ; 16 ; 32.

Example query

meter(2,4), tempo(100), key(major), % 2/4 time signature, 100 beats per minute, major key
voice(bass), voice(melody), % two voices: a bass and a melody
range(bass,g,1,c,3), range(melody,g,3,e,5), % bass ranges from g1 to c3, melody from g3 to e5
instrument(bass,'contrabass'), instrument(melody, 'soprano sax'), % MIDI instruments used to render the voices
max_jump(bass,12), max_jump(melody,5), % maximal interval between consecutive notes in semitones
shortest_duration(bass,8),shortest_duration(melody,16), % shortest possible bass note is an eighth, for melody a sixteenth
measures(8) % generate 8 measures
Output constraints

% outputs
:- chrism
   mchord(+measure,+chord),
   beat(+voice,+measure,+int,+float,+duration),
% beat(V,M,B,P,D): for voice V, a note starts at measure M, beat B, position P, with duration D
   note(+voice,+measure,+int,+float,+note),
   octave(+voice,+measure,+int,+float,+),
% note and octave at that measure-beat-position
   tied(+voice,+measure,+int,+float).
% the note at this position is tied to the next note

Rules for chords

- One chord per measure (for simplicity)
- First and last measure chords correspond to key:
  - key(major) ==> mchord(1,c).
  - key(major), measures(N) ==> mchord(N,c).
  - key(minor) ==> mchord(1,am).
  - key(minor), measures(N) ==> mchord(N,am).
- Other measure chords are assigned:
  mchord(A,Chord), next_measure(A,B), measures(M)
  ==> B < M |
  msw(chord_choice(Chord),NextChord),
  mchord(B,NextChord).
Default chord transition probabilities

values(chord_choice(_), [c, g, f, am, em, dm]).
:- set_sw(chord_choice(c), [0.2, 0.3, 0.25, 0.15, 0.05, 0.05]).
:- set_sw(chord_choice(g), [0.3, 0.15, 0.2, 0.2, 0.05, 0.1]).
:- set_sw(chord_choice(f), [0.25, 0.3, 0.1, 0.15, 0.05, 0.15]).
:- set_sw(chord_choice(am), [0.1, 0.2, 0.35, 0.1, 0.05, 0.2]).
:- set_sw(chord_choice(em), [0.05, 0.2, 0.3, 0.35, 0.05, 0.05]).
:- set_sw(chord_choice(dm), [0.05, 0.2, 0.35, 0.2, 0.05, 0.15]).

Rhythm: “Beat splitting”

Markov Chain of order 1
(straightforward to extend to higher order or HMM)
Rhythm: “Beat splitting”

```prolog
% make initial beats (one beat per count)
meter(\text{N}, \text{D}), \text{voice}(<V>), \text{measure}(<M>) \Rightarrow \text{make
deadline}(<N>, \text{D}, <M>, <V>).
m\text{make\_beats}(<N>, \text{D}, <M>, <V>) \Leftrightarrow <N> > 0 |
\quad N1 \text{ is } N-1, \text{ next\_beat}(<V>, \text{M}, N1, 0, M, N, 0),
\quad \text{beat}(<V>, \text{M}, N1, 0, \text{D}), \text{ make\_beats}(<N1>, \text{D}, <M>, <V>).

% split some of the beats in two
\text{split\_beat}(<V>)
\text{meter}(_{\text{N}}, \text{OD}), \text{ shortest\_duration}(<V>, \text{SD})
\quad \text{beat}(<V>, M, N, X, D), \text{ next\_beat}(<V>, M, N, X, NM, NN, NX)
\quad \Leftrightarrow D<\text{SD} | D2 \text{ is } D*2, X2 \text{ is } X+1/(D2/\text{OD}),
\quad \text{next\_beat}(<V>, \text{M}, N, X, M, N, X2), \text{ next\_beat}(<V>, \text{M}, N, X2, NM, NN, NX),
\quad \text{beat}(<V>, \text{M}, N, X, \text{D2}), \text{ beat}(<V>, \text{M}, N, X2, \text{D2}).
```

Abstract beat positions

```prolog
% abstract\_beat(+meter, +Beat, +SubBeat, -AbstractBeat)
abstract\_beat(_, 0, 0, \text{first}) \:- !.

% binary meter: middle is strong
abstract\_beat(M, 0, N, \text{strong}) \:- 0 \text{ is } M \text{ mod } 2, N \text{ is } M//2, !.
abstract\_beat(3, _, 0, \text{strong}) \:- !.

abstract\_beat(2, _, 0.75, \text{prestrong}) \:- !.
abstract\_beat(3, 2, 0.5, \text{prestrong}) \:- !.
abstract\_beat(4, 1, 0.5, \text{prestrong}) \:- !.
abstract\_beat(N, M, 0.5, \text{prestrong}) \:- N \text{ =:= } M+1, !.

abstract\_beat(2, 0, 0.5, \text{weak}) \:- !.
abstract\_beat(2, 1, 0.5, \text{weak}) \:- !.
abstract\_beat(3, _, 0.5, \text{weak}) \:- !.
abstract\_beat(_, _, 0, \text{weak}) \:- !.

abstract\_beat(_, _, _, \text{weakest}).
```
Melody Notes

\begin{align*}
\text{beat} & (V,M,N,X,D), \\
\text{mchord} & (M,C), \\
\text{next}\text{\_beat} & (V,M_1,N_1,X_1,M,N,X), \\
\text{octave} & (V,M_1,N_1,X_1,OO) \\
\implies & \\
\text{V} & \text{\textasciitilde\textasciitilde} \text{drums, V} \text{\textasciitilde\textasciitilde} \text{chords} \\
\mid & \\
\text{abstract}\text{\_beat} & (M,N,X,AB), \\
\text{soft}\_\text{msw} & (\text{note}\_\text{choice}(V,C,AB),\text{Note}), \\
\text{note} & (V,M,N,X,\text{Note}), \\
(\text{Note} & = r \rightarrow \text{octave}\_d(V,M,N,X,0) \\
& ; \text{find}\_\text{octave}\_d(V,M,N,X,OO)) .
\end{align*}
Melody Notes

```
beat(V,M,N,X,D),
mchord(M,C),
nextBeat(V,M1,N1,X1,M,N,X),
octave(V,M1,N1,X1,OO)
==>
  V \== drums, V \== chords
  | abstractBeat(M,N,X,AB),
     soft_msw(note_choice(V,C,AB),Note),
     note(V,M,N,X,Note),
     ( Note == r -> octave_d(V,M,N,X,0)
     ; find_octave_d(V,M,N,X,OO) ) .
```

Soft_msw ?
This is a variant of msw that allows backtracking (even during sampling).

Constraints

- For example, maximum interval between two consecutive notes:

```
max_jump(V,MInt), octave(V,M1,N1,X1,OO),
note(V,M1,N1,X1,ON), note(V,M,N,X,NN),
nextBeat(V,M1,N1,X1,M,N,X), octave(V,M,N,X,NO)
==>
  interval(ON,OO,NN,NO,Interval), Interval > Mint
  |
  fail.
```
join_notes(V, cond M=M2, cond N=N2) ?
note(V, M, N, X, Note), note(V, M2, N2, X2, Note),
next_beat(V, M, N, X, M2, N2, X2)
==> V \== drums | tied(V, M, N, X).

The keyword cond evaluates its argument to a boolean.
Full program

- About 50 rules (no time to show them all)
  - Plus some auxiliary rules and predicates
    - For writing the output LilyPond file
    - For more efficient learning
    - To compute intervals, abstract beats, etc.
- All in all about 500 lines of code → very small!
- 107 probability distributions (8 parametrized experiments), 324 parameters to be learned

Demo
Other potential uses

- Current system: generation and learning
- Other tasks can also be done (with the same underlying program!)
  - Classification
  - Improvisation
  - Automatic accompaniment
  - Complete an unfinished composition
  - ...

Classification

- Preparation steps:
  - Learn from training set A (e.g. *The Beatles*) → model $M_A$
  - Learn from training set B (e.g. *The Rolling Stones*) → model $M_B$

- Classification:
  - Given an unknown song, compute its probability in $M_A$ and in $M_B$
  - Classify accordingly (highest probability wins)
  - Categories can be anything (style, composer, …)
Improvisation, Accompaniment

- Given a partial song (bass, drums, chords), generate a melody that “fits”
  - Put the known voices in the query, the rest will be generated
- Given a song, find underlying chord sequence
  - Compute Viterbi explanation of a partial observation

Conclusion

- Very high-level probabilistic programming
  - Easy to extend and refine
- “Statistical model”: same program for both synthesis (sampling) and analysis (probability computation, learning)
- Powerful combination of different paradigms:
  - Probabilistic Logic Learning
  - Constraint Programming
  - Rule-based Programming