

Logic Programming

Cezary Kaliszyk **Georg Moser**

Institute of Computer Science @ UIBK

Winter 2015



Overview

Outline of the Lecture

Monotone Logic Programs

introduction, basic constructs, unification, database and recursive programming, termination

Incomplete Data Structures and Constraints

incomplete data structures, definite clause grammars, constraint logic programming, answer set programming

Full Prolog

semantics, correctness proofs, meta-logical predicates, cuts nondeterministic programming, efficient programs, complexity

Summary of Last Lecture

Definition

given a list $[1, 2, 3]$ it can be **represented** as the **difference** of two lists; the difference of two lists is denoted as $As \setminus Bs$ and called **difference list**

Example

```
expr(knoten(1, [])) ->
    "1".
expr(knoten(+, [Expr1, Expr2])) ->
    "(",
    expr(Expr1),
    "+",
    expr(Expr2),
    ")".
```

Difference-structures

Example

convert the sum $(a + b) + (c + d)$ into $(a + (b + (c + (d + 0))))$

Definition

we make use of **difference-sums**: $E1++E2$, where $E1, E2$ are incomplete; the empty sum is denoted by 0

Example

```
normalise(Exp, Norm) :- normalise_ds(Exp, Norm ++ 0).
normalise_ds(A+B, Norm ++ Norm0) :-
    normalise_ds(A, Norm ++ NormB),
    normalise_ds(B, NormB ++ Norm0).
normalise_ds(A, (A + Norm) ++ Norm) :-
    constant(A).
```

Example (Definite Clause Grammars)

```

sentence(sentence(N,V)) → noun_phrase(N), verb_phrase(V).
noun_phrase(np(D,N)) → determiner(D), noun_phrase2(N).
noun_phrase(np(N)) → noun_phrase2(N).
noun_phrase2(np2(A,N)) → adjective(A), noun_phrase2(N).
noun_phrase2(np2(N)) → noun(N).
verb_phrase(vp(V,N)) → verb(V), noun_phrase(N).
verb_phrase(vp(V)) → verb(V). determiner → [the].
determiner → [a].
noun → [pie-plate].
noun → [surprise].
adjective → [decorated].
verb → [contains].

```

```

sentence(PT)  $\overset{*}{\Rightarrow}$  "the decorated pie-plates contain a surprise"

```

Example

```

sentence(PT)  $\overset{*}{\Rightarrow}$  "the decorated pie-plate contains a surprise"
sentence(PT)  $\overset{*}{\Rightarrow}$  "the decorated pie-plates contain a surprise"

```

Example

```

determiner(det(the)) → [the].
determiner(det(a)) → [a].
noun(noun(pie-plate)) → [pie-plate].
noun(noun(pie-plates)) → [pie-plates].
noun(noun(surprise)) → [surprise].
noun(noun(surprises)) → [surprises].
adjective(adj(decorated)) → [decorated].
verb(verb(contains)) → [contains].
verb(verb(contain)) → [contain].

sentence(PT)  $\overset{*}{\Rightarrow}$  "the decorated pie-plates contains a surprise"

```

Extension: Number Agreement

Example

```

sentence(sentence(NP,VP),Num) →
    noun_phrase(N,Num), verb_phrase(V,Num).
:
determiner(det(the),Num) → [the].
determiner(det(a),singular) → [a].
noun(noun(pie-plate),singular) → [pie-plate].
noun(noun(pie-plates),plural) → [pie-plates].
noun(noun(surprise),singular) → [surprise].
noun(noun(surprises),plural) → [surprises].
adjective(adj(decorated)) → [decorated].
verb(verb(contains),singular) → [contains].
verb(verb(contain),plural) → [contain].

sentence(PT)  $\overset{*}{\Rightarrow}$  "the decorated pie-plates contain a surprise"

```

Example

```

sentence →
    subject ,
    predicate.

subject →
    [the], [big], [bear].
subject →
    "the", "little", "lion".

predicate →
    [roars].
predicate →
    [is, happy].
predicate →
    [lives, in, the, golden, city].

:- phrase(sentence,Text), Text = [the, big, bear, roars].
:- phrase(sentence,Text), Text = [116, 104, 101, 108, 105|_].

```

Regular Predicate from Within a DCG

Task

write a DCG for `number(N)` that recognised numbers in English:

```
?- phrase(number(N), "onehundredandseventyfive").
   N = 175 ;
```

Definition

Prolog provides an arithmetical interface

Value is Expression

Example

<code>X is 3+5</code>	<code>8 is 3+5</code>	<code>N is N+1</code>
<code>X ↦ 8</code>	<code>true</code>	<code>nonsensical</code>

Solution

```
number(0) → "zero".
number(N) → xxx(N).
```

```
xxx(N) → digit(D), "hundred", rest_xxx(N1),
        {N is D * 100 + N1}.
```

```
rest_xxx(0) → "".
rest_xxx(N) → "and", xx(N).
```

```
xx(N) → digit(N).
xx(N) → teen(N).
xx(N) → tens(T), rest_xx(N1), {N is T + N1}.
```

```
rest_xx(0) → "".
rest_xx(N) → digit(N).
```

```
digit(1) → "one".
digit(2) → "two".
teen(10) → "ten".
tens(20) → "twenty".
```

Generate and Test

Theorem (Four Colour Theorem)

no more than four colours are required to colour the regions of a map so that no two adjacent regions have the same color

Example



Auxiliary Predicates

Example

```
select(X, [X|Xs], Xs).
select(X, [Y|Ys], [Y|Zs]) :-
    select(X, Ys, Zs).
```

```
member(X, [X|_Xs]).
member(X, [_Y|Xs]) :-
    member(X, Xs).
```

```
:- sublist_of([b,d], [a,b,c,d]).
```

```
sublist_of([], _Ys).
sublist_of([X|Xs], Ys) :-
    member(X, Ys),
    sublist_of(Xs, Ys).
```

Generate and Test

Example

```
is_map([region(a,A,[B,C,D]), region(b,B,[A,C,E]),
       region(c,C,[A,B,D,E,F]), region(d,D,[A,C,F]),
       region(e,E,[B,C,F]), region(f,F,[C,D,E])]).

coloured_map([Region|Regions], Colours) :-
    coloured_region(Region,Colours),
    coloured_map(Regions,Colours).
coloured_map([],Colours).

coloured_region(region(Name,Colour,Neighbours), Colours) :-
    select(Colour,Colours,Colours1),
    sublist_of(Neighbours,Colours1).

test_colour(Map) :-
    is_map(Map),
    is_colours(Colours),
    coloured_map(Map,Colours).
```

Nondeterministic Programming

Example

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	\emptyset	$\{q_2\}$
$*q_2$	\emptyset	\emptyset

Definition

A **NFA** is quintuple $(Q, \Sigma, \Delta, I, F)$ such that

- 1 Q is a set of states
- 2 Σ is an alphabet
- 3 Δ is relation on $(Q \times \Sigma) \times Q$
- 4 I are the initial states
- 5 F are the final states

Example

```
accept(S) :-
    initial(Q),
    accept(Q,S).

accept(Q, [X|Xs]) :-
    delta(Q,X,Q1),
    accept(Q1,Xs).

accept(Q, []) :-
    final(Q).

initial(q0).
final(q2).

delta(q0,0,q0).
delta(q0,0,q1).
delta(q0,1,q0).
delta(q1,1,q2).

:- accept([0,0,0,1,0,1]).
```

Type Predicates

Recall

type predicates are unary relations concerning the type of a term

Definition

- **is_list**: type check for a list
- **integer**: type check for an **integer**
- **atom**: type check for an **atom**
- **compound**: type check for a **compound** term

Example

```
constant(X) :-
    integer(X).
constant(X) :-
    atom(X).
```

Example

```
flatten([X|Xs],Ys) :-
    is_list(X), flatten(X,Ys1),
    flatten(Xs,Ys2), append(Ys1,Ys2,Ys).
flatten(X,[X]) :-
    flatten([],[]).
:- flatten([[a],[b,[c,d]],e],[a,b,c,d,e])
```

Example

```
flatten(Xs,Ys) :- flatten(Xs,[],Ys).
flatten([X|Xs],S,Ys) :-
    is_list(X), flatten(X,[Xs|S],Ys).
flatten([X|Xs],S,[X,Ys]) :-
    constant(X), X ≠ [], flatten(Xs,S,Ys).
flatten([], [X|S],Ys) :- flatten(X,S,Ys).
flatten([], [], []).
```

Once Again: Difference Lists and DCGs

Example

```
constant(X) :- listen_zusammen ([[1,2],[4,5]],[1,2,4,5]).
```

```
listen_zusammen(Xss,Xs) :-
    phrase(seqq(Xss),Xs).
```

```
seqq([]) →
```

```
[].
```

```
seqq([Xs|Xss]) →
```

```
seq(Xs),
seqq(Xss).
```

```
seq([]) →
```

```
[].
```

```
seq([C|Cs]) →
```

```
[C],
seq(Cs).
```

GPU

Example (definite clause grammars)

- Example 47 ...