Prolog lecture 8

Go to:

http://etc.ch/3CQQ

Or scan the barcode

Today's discussion

Videos

- Sudoku
- Constraints

Q: What does Prolog allow us to do (other than coding in a different way) that other languages can't? Not meaning to sound dismissive just curious of applications!

A: ...

DOOP

```
void run(A a) {
    a.f();
}

static void main(String[] args) {
    run(new B());
}
```

Does this program crash?

```
class A {
    void f() {
        throw new AssertionError();
    }
}

class B extends A {
    void f() {
        System.out.println("hi!");
    }
}
```

DOOP

```
void run(A a) {
    a.f();
}

static void main(String[] args) {
    run(new B());
}
```

Need to know the set of objects that a could point to.

This is called 'points-to' analysis

DOOP

Introduction

Versatile. DOOP is a framework for pointer, or points-to, analysis of Java programs. DOOP implements a range of algorithms, including context insensitive, call-site sensitive, and object-sensitive analyses, all specified modularly as variations on a common code base.

Fast. Compared to alternative context-sensitive pointer analysis implementations (such as Paddle) DOOP is much faster, and scales better. Also, with comparable context-sensitivity features, DOOP is more precise in handling some Java features (for example exceptions) than alternatives.

Declarative. DOOP builds on the idea of specifying pointer analysis algorithms declaratively, using Datalog: a logic-based language for defining (recursive) relations. DOOP carries the declarative approach further than past work (such as bddbddb) by describing the full end-to-end analysis in Datalog and optimizing aggressively through exposition of the representation of relations (for example indexing) to the Datalog language level.

Doop

$$\textsc{InterProcAssign}(\mathit{to},\ \mathit{from}) \leftarrow$$
$$\textsc{CallGraph}(\mathit{invo},\ \mathit{meth}),$$
$$\textsc{FormalArg}(\mathit{meth},\ n,\ \mathit{to}),\ \textsc{ActualArg}(\mathit{invo},\ n,\ \mathit{from}).$$

$$\textsc{InterProcAssign}(\mathit{to},\ \mathit{from}) \leftarrow$$
$$\textsc{CallGraph}(\mathit{invo},\ \mathit{meth}),$$
$$\textsc{FormalReturn}(\mathit{meth},\ \mathit{from}),\ \textsc{ActualReturn}(\mathit{invo},\ \mathit{to}).$$

$$\textsc{VarPointsTo}(\mathit{to},\ \mathit{heap}) \leftarrow$$
$$\textsc{InterProcAssign}(\mathit{to},\ \mathit{from}),$$
$$\textsc{VarPointsTo}(\mathit{from},\ \mathit{heap}).$$

Implement list reverse (without an accumulator)

Vote when done

http://etc.ch/3CQQ

DirectPoll

Implement list reverse (without an accumulator)

```
reverse([],[]).

reverse([H|T],R) :- reverse(T,R1), append(R1,[H],R).
```

Implement list reverse (with an accumulator)

Vote when done

http://etc.ch/3CQQ

DirectPoll

Implement list reverse (with an accumulator)

```
reverseAcc([],Acc,Acc).

reverseAcc([H|T],R,Acc) :- reverseAcc(T,R,[H|Acc]).
```

Implement reverse with difference lists

Which version of reverse should we start with?

1. reverse without an accumulator
2. reverse with an accumulator

http://etc.ch/3CQQ

DirectPoll

Implement reverse with difference lists

Vote when finished

http://etc.ch/3CQQ

DirectPoll

Implement reverse with difference lists

1) Replace all lists in the append with difference lists
2) Choose the correct form of empty list:
    a) if you are generating then use A-A
    b) if you are testing then use []-[]
3) Manually unify the variables involved in the append in the places that append would make them equal
4) Remove the append because its now redundant

http://etc.ch/3CQQ

DirectPoll

Implement reverse with difference lists

```
reverseD([],[]).
reverseD([H|T],R) :- reverseD(T,R1),
                     append(R1,[H],R).
```

Implement reverse with difference lists

```
reverseD([],A-A).
reverseD([H|T],R-S) :- reverseD(T,R1-S1),
                       append(R1-S1,[H|H1]-H1,R-S).
```

Implement reverse with difference lists

unify S1 with [H|H1]

```
reverseD([],A-A).
reverseD([H|T],R-S) :- reverseD(T,R1-[H|H1]),
                       append(R1-[H|H1],[H|H1]-H1,R-S).
```

Implement reverse with difference lists

unify R with R1

```
reverseD([],A-A).
reverseD([H|T],R1-S) :- reverseD(T,R1-[H|H1]),
                        append(R1-[H|H1],[H|H1]-H1,R1-S).
```

Implement reverse with difference lists

unify S with H1

```
reverseD([],A-A).
reverseD([H|T],R1-H1) :- reverseD(T,R1-[H|H1]),
                         append(R1-[H|H1],[H|H1]-H1,R1-H1).
```

Implement reverse with difference lists

remove the append

```
reverseD([],A-A).
reverseD([H|T],R1-H1) :- reverseD(T,R1-[H|H1]).
```

What's the difference?

```
reverse([],[]).
reverse([H|T],R) :- reverse(T,R1), append(R1,[H],R).


reverseAcc([],Acc,Acc).
reverseAcc([H|T],R,Acc) :- reverseAcc(T,R,[H|Acc]).


reverseD([],A-A).
reverseD([H|T],R1-H1) :- reverseD(T,R1-[H|H1]).
```

Q: Is writing CLP programs using the library strictly examinable, or is it more about the concepts of CLP?

A: The concepts. Given the relatively short time devoted to it any question on this would be about the principles and you would be given the syntax if you needed it.

Challenge: Plan your day (CLP)

Supervision work: 55 minutes

Email: 10 minutes

Laundry: 5 minutes to start it, 60 mins wash/dry, 10 mins to put away.

Plan your day (CLP)

```
:- use_module(library(clpfd)).

?- Tasks = [(Sv,55),(E,15),(Ls,5),(Lf,10)],
```

Add the constraint that the laundry takes 60 minutes

http://etc.ch/3CQQ

DirectPoll

Plan your day (CLP)

```
:- use_module(library(clpfd)).

?- Tasks = [(Sv,55),(E,15),(Ls,5),(Lf,10)],
   [Sv,E,Ls,Lf] ins 0..100,
   Ls+65 #=< Lf,
```

Plan your day (CLP)

```
:- use_module(library(clpfd)).

?- Tasks = [(Sv,55),(E,15),(Ls,5),(Lf,10)],
   [Sv,E,Ls,Lf] ins 0..100,
   Ls+65 #=< Lf,
```

Add the constraint that we must finish all jobs in 100 minutes

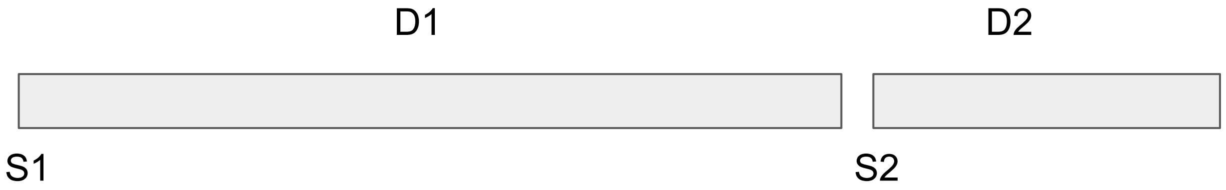
http://etc.ch/3CQQ

DirectPoll

Plan your day (CLP)

```
:- use_module(library(clpfd)).

notlate([]).
notlate([(S1,D1)|T]) :- S1 + D1 #=< 100, notlate(T).

?- Tasks = [(Sv,55),(E,15),(Ls,5),(Lf,10)],
   [Sv,E,Ls,Lf] ins 0..100,
   Ls+65 #=< Lf,
   notlate(Tasks),
```

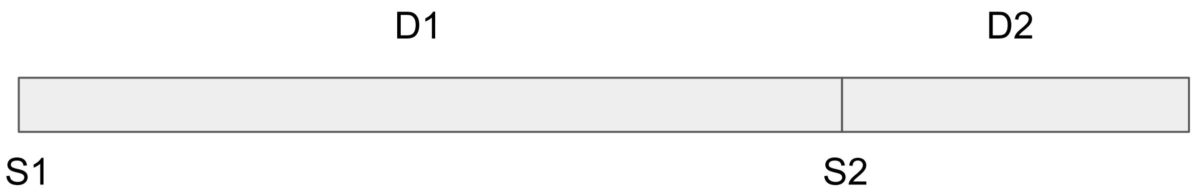
We need to model a sequence of tasks



Write a constraint that the tasks are in sequence

http://etc.ch/3CQQ

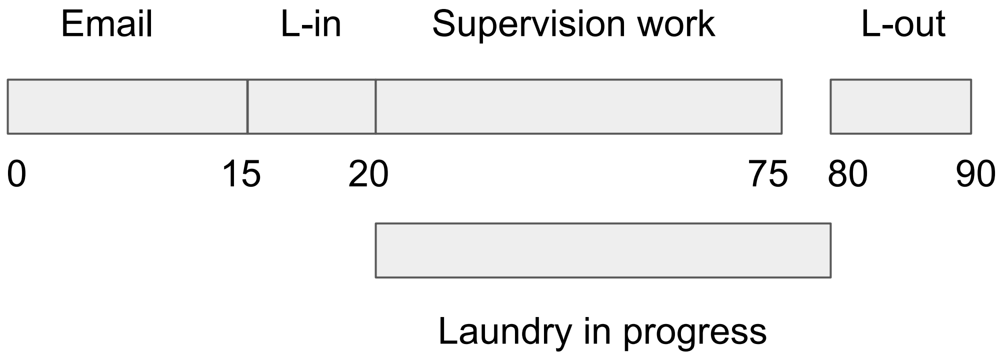
We need to model a sequence of tasks



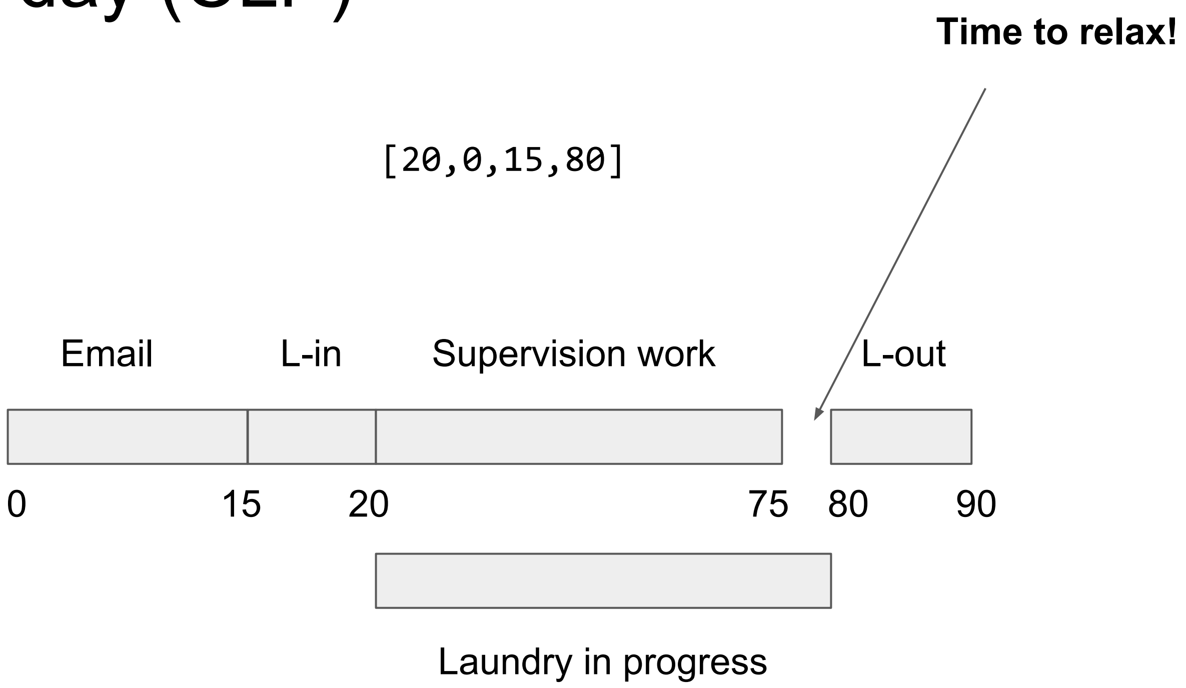

```
sequence([_]).
sequence([(S1,D1),(S2,D2)|T]) :- S1 + D1 #=< S2,
                                  sequence([(S2,D2)|T]).

… perm(Tasks,Order), sequence(Order) …
```

Plan your day (CLP)

[20,0,15,80]



Plan your day (CLP)



End of the course

I hope you found the format helpful - please fill out the feedback forms!

Thank you for coming to the lectures!