Prolog lecture 7
Go to:
http://etc.ch/6Vhc
Or scan the barcode
Today's discussion

Videos

- Difference
- Empty difference lists
- Difference list example
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: ...
If the predicate `classListIsTypeSafe` is not true, the type checker must throw the exception `VerifyError` to indicate that the class file is malformed. Otherwise, the class file has type checked successfully and bytecode verification has completed successfully.

The rest of this section explains the process of type checking in detail:

- First, we give Prolog predicates for core Java Virtual Machine artifacts like classes and methods (§4.10.1.1).

- Second, we specify the type system known to the type checker (§4.10.1.2).

- Third, we specify the Prolog representation of instructions and stack map frames (§4.10.1.3, §4.10.1.4).

- Fourth, we specify how a method is type checked, for methods without code (§4.10.1.5) and methods with code (§4.10.1.6).

- Fifth, we discuss type checking issues common to all load and store instructions (§4.10.1.7), and also issues of access to protected members (§4.10.1.8).

- Finally, we specify the rules to type check each instruction (§4.10.1.9).
THE CLASS FILE FORMAT

\[
\text{classList}(\text{Class}) \leftarrow \\
\text{classIsTypeSafe}(\text{Class}), \\
\text{classClassName}(\text{Class}, \text{Name}), \\
\text{classDefiningLoader}(\text{Class}, \text{L}), \\
\text{superclassChain}(\text{Name}, \text{L}, \text{Chain}), \\
\text{Chain} \leftarrow [\text{null}], \\
\text{classSuperClassName}(\text{Class}, \text{SuperclassName}), \\
\text{loadedClass}(\text{SuperclassName}, \text{L}, \text{Superclass}), \\
\text{classListNotFinal}(\text{Superclass}), \\
\text{classListMethods}(\text{Class}, \text{Methods}), \\
\text{classList}(\text{methodIsTypeSafe}(\text{Class}), \text{Methods}).
\]

\[
\text{classList}(\text{Class}) \leftarrow \\
\text{classClassName}(\text{Class}, \text{'java/lang/Object'}), \\
\text{classDefiningLoader}(\text{Class}, \text{L}), \\
\text{isBootstrapLoader}(\text{L}), \\
\text{classListMethods}(\text{Class}, \text{Methods}), \\
\text{classList}(\text{methodIsTypeSafe}(\text{Class}), \text{Methods}).
\]

The Prolog predicate \text{classList}(\text{Class}) assumes that \text{Class} is a Prolog term representing a binary class that has been successfully parsed and loaded. This specification does not mandate the precise structure of this term, but does require that certain predicates be defined upon it.

For example, we assume a predicate \text{classListMethods}(\text{Class}, \text{Methods}) that, given a term representing a class as described above as its first argument, binds its second argument to a list comprising all the methods of the class, represented in a convenient form described later.

Iff the predicate \text{classList}(\text{Class}) is not true, the type checker must throw the exception \text{VerifyError} to indicate that the class file is malformed. Otherwise, the
Q: Attempting the towers of hanoi problem and run into stack space issues which makes me think my state representation is bad, any hints (specific and general)?
Q: Attempting the towers of hanoi problem and run into stack space issues which makes me think my state representation is bad, any hints (specific and general)?

A: if you are running out of stack space you probably have a searching-forever problem are more rules matching than you thought? General state representation hint: as little redundancy as possible.
Difference lists

Which of these is a difference list:

1. \text{diff}(A,B)
2. A-B
3. \[1,2,3|A]-A
4. \[1,2,3|A]-B
5. []-[]
6. A-A

http://etc.ch/6Vhc
Which of these is correct?

upto(0,[]-[]).
upto(N,[N|T]-A) :- M is N-1, upto(M,T-A).

upto2(0,A-A).
upto2(N,[N|T]-A) :- M is N-1, upto2(M,T-A).
What goes wrong?

upto(0,[]-[]).
upto(N,[N|T]-A) :- M is N-1, upto(M,T-A).

?- upto(3,L).
A = [3, 2, 1]-[]
What goes wrong?

\[ \text{upto}(0, [[]]) \].
\[ \text{upto}(N, [N|T]-A) :\text{ M is } N-1, \text{ upto}(M,T-A). \]

?- upto(3,L).
A = [3, 2, 1]-[];
ERROR: Out of global stack
**What goes wrong?**

```
upto(0,[]-[]).
upto(N,[N|T]-A) :- M is N-1, upto(M,T-A).
```

?- trace, upto(3, A).

Call: (9) upto(3, _402) ? creep
Call: (10) upto(2, _704-_698) ? creep
Call: (11) upto(1, _722-_698) ? creep
Call: (12) upto(0, _740-_698) ? creep
Exit: (12) upto(0, []-[]) ? creep
Exit: (11) upto(1, [1]-[]) ? creep
Exit: (10) upto(2, [2, 1]-[]) ? creep
Exit: (9) upto(3, [3, 2, 1]-[]) ? creep

A = [3, 2, 1]-[] ;
What goes wrong?

```prolog
upto(0,[]-[]).
upto(N,[N|T]-A) :- M is N-1, upto(M,T-A).

A = [3, 2, 1]-[] ;
  Redo: (12) upto(0, _740-_698) ? creep
  Call: (13) upto(-1, _758-_698) ? creep
  Call: (14) upto(-2, _776-_698) ? creep
...
```
upto2 is also broken

?- trace, upto2(3,A).
    Call: (9) upto2(3, _890) ? creep
    Call: (10) upto2(2, _1182-_1176) ? creep
    Call: (11) upto2(1, _1200-_1176) ? creep
    Call: (12) upto2(0, _1218-_1176) ? creep
    Exit: (12) upto2(0, _1176-_1176) ? creep
    Exit: (11) upto2(1, [1|_1176]-_1176) ? creep
    Exit: (10) upto2(2, [2, 1|_1176]-_1176) ? creep
    Exit: (9) upto2(3, [3, 2, 1|_1176]-_1176) ? creep
    A = [3, 2, 1|_1176]-_1176 ;

upto2(0,A-A).
upto2(N,[N|T]-A) :- M is N-1,
        upto2(M,T-A).
upto2 is also broken

\[
A = [3, 2, 1] \times 1176 ;
\]

Redo: (12) upto2(0, _1218-_1176) ? creep

Call: (13) upto2(-1, _1236-_1176) ? creep

Call: (14) upto2(-2, _1254-_1176) ? creep

Call: (15) upto2(-3, _1272-_1176) ? creep

...
How do we fix it?

upto2(0,A-A).
upto2(N,[N|T]-A) :- M is N-1,
                 upto2(M,T-A).
How do we fix it?

upto2(0,A-A) :- !.
upto2(N,[N|T]-A) :- M is N-1,
               upto2(M,T-A).
Difference list length

Which of these is correct?

difflen([]-[],0).
difflen([_|T]-A,N) :- difflen(T-A,M), N is M+1.

difflen2(A-A,0).
difflen2([_|T]-A,N) :- difflen2(T-A,M), N is M+1.
Difflen is wrong

difflen([],[],0).
difflen([T|T]-A,N) :- difflen(T-A,M), N is M+1.

?- difflen([1,2,3|A]-A,B).
A = [], B = 3 ;
ERROR: Out of local stack

Don't forget backtracking behaviour!
Difflen/2 is also wrong (twice)

difflen2(A-A,0).
difflen2([_|T]-A,N) :- difflen2(T-A,M), N is M+1.

?- difflen2([1,2,3|A]-A,B).
A = [1, 2, 3|A], B = 0 ;
A = [2, 3|A], B = 1 ;
A = [3|A], B = 2 ;
B = 3 ; A = [__7791920|A], B = 4 ;
A = [__7791920, __7791932|A], B = 5 ;
...
How should I fix difflen?

difflen([],[],0).
difflen([_|T]-A,N) :- difflen(T-A,M), N is M+1.
How should I fix `difflen`?

difflen([],[],0) :- !.
difflen([_|T]-A,N) :- difflen(T-A,M), N is M+1.
What's the difference-list difference?

upto2(0,A-A) :- !.
upto2(N,[N|T]-A) :- M is N-1, upto2(M,T-A).

difflen([],[],0) :- !.
difflen([_|T]-A,N) :- difflen(T-A,M), N is M+1.
What's the difference-list difference?

generating an empty list

upto2(0,A-A) :- !.
upto2(N,[N|T]-A) :- M is N-1, upto2(M,T-A).

testing for an empty list

difflen([],[],0) :- !.
difflen([_|T]-A,N) :- difflen(T-A,M), N is M+1.
What's the difference-list difference?

generating an empty list

testing for an empty list

and destroying the list!
Challenge: Implement Quicksort

Partition the list into two pieces

Quicksort each half
Implement Quicksort

% partition(Pivot, List, Left, Right) succeeds if Left is all the elements in List less than or equal to the pivot and Right is all the elements greater than the pivot

% quicksort(L1, L2) succeeds if L2 contains the elements in L1 in ascending order
Implement partition

% partition(Pivot,List,Left,Right) succeeds if Left is all the
% elements in List less than or equal to the pivot and Right is
% all the elements greater than the pivot

partition(_,[],[],[]).
partition(P,[H|T],[H|L],R) :- P <= H, partition(P,T,L,R).
partition(P,[H|T],L,[H|R]) :- P > H, partition(P,T,L,R).
Implement quicksort

partition(_, [], [], []). 
partition(P, [H|T], [P|L], R) :- P <= H, partition(P, T, L, R).

quicksort([], []). 
quicksort([P|T], Sorted) :-
    partition(P, T, L, R),
quicksort(L, L1), quicksort(R, R1),
append(L1, R1, Sorted).
Is it useful to turn this into difference lists?

partition(_,[],[],[]).
partition(_,[],[],[]).
partition(P,[H|T],[P|L],R) :- P <= H, partition(P,T,L,R).
partition(P,[H|T],L,[P|R]) :- P > H, partition(P,T,L,R).

quicksort([],[]).
quicksort([P|T],Sorted) :-
    partition(P,T,L,R),
quicksort(L,L1), quicksort(R,R1),
    append(L1,[P|R1],Sorted).

http://etc.ch/6Vhc
Next time

Videos

  Sudoku

Constraints

Ask questions at [www.slido.com](http://www.slido.com) with event code E508