

Defining Language TWO

- Extend Language ONE with:
 - Variables
 - An ML-style `let` expression for defining them

TWO: Syntax

TWO:

$\langle exp \rangle^* ::= \langle exp \rangle + \langle mulexp \rangle \mid \langle mulexp \rangle$
 $\langle mulexp \rangle ::= \langle mulexp \rangle * \langle rootexp \rangle \mid \langle rootexp \rangle$
 $\langle rootexp \rangle ::= \text{let val } \langle variable \rangle = \langle exp \rangle \text{ in } \langle exp \rangle \text{ end}$
 $\mid (\langle exp \rangle) \mid \langle variable \rangle \mid \langle constant \rangle$

- A subset of ML expressions
- This grammar is unambiguous
- A sample Language TWO expression:
 let val y = 3 in y*y end
- What does the parse tree for the above expression look like?

TWO: Abstract Syntax

Additional abstract syntax nodes for language TWO:

- (1) var(X) dereferences a variable X
- (2) let(X,E1,E2) binds the variable X to expression E1 in the context of expression E2.

Example: the TWO program

```
let val y = 3 in y*y end
```

will result in the AST

```
let(y, const(3), times(var(y), var(y)))
```

From Parse Tree to Prolog AST

- Consider: `2 * let x = 5 in 1+x end`
 - Parse tree?
 - AST?
 - Prolog AST?

TWO: Semantics

In order to provide semantics we need to remember the values assigned to variables -- binding environments (fancy word for dictionary!)

In our case, for the Prolog based semantics, we let the terms `bind(X,K)` represent the binding of variable `X` to value `K`. A context is simply a list of these binding terms:

```
[bind(y,3), bind(q,20), bind(z,5)]
```

Given this binding structure, we can write a predicate, `lookup/3`, that returns a variable binding for a particular `Var`

```
lookup(Var, [bind(Var, Value) | _ ], Value) .  
lookup(Var, [ _ | Rest], Value) :- lookup(Var, Rest, Value) .
```

Finds the most recent binding of variable `Var` if there is one.

TWO: Prolog Interpreter

```
val2 (plus (X,Y) ,B,Value) :-  
    val2 (X,B,XValue) ,  
    val2 (Y,B,YValue) ,  
    Value is XValue + YValue.
```

```
val2 (times (X,Y) ,B,Value) :-  
    val2 (X,B,XValue) ,  
    val2 (Y,B,YValue) ,  
    Value is XValue * YValue.
```

```
val2 (const (X) ,_,X) .
```

```
val2 (var (X) ,B,Value) :-  
    lookup (X,B,Value) .
```

```
val2 (let (X,Exp1,Exp2) ,B,Value) :-  
    val2 (Exp1,B,XValue) ,  
    val2 (Exp2, [bind (X,XValue) | B] ,Value) .
```

val2 / 3 - interpretation predicate
first argument: AST
second argument: binding env;
third argument: semantic value.

Examples

```
let val y = 3 in y*y end
```

```
?- val2(let(y,const(3),times(var(y),var(y))),[ ],X) .
```

```
X = 9
```

```
Yes
```

```
let val y = 3 in
  let val x = y*y in
    x*x
  end
end
```

```
let val y = 1 in
  let val y = 2 in
    y
  end
end
```

Exercises

- Use the semantics of TWO to show the following:
 - Assume that the context $B = [\text{bind}(y, 3)]$ then the semantic value of $'2 * y'$ is 6
 - The semantic value of $'2 * \text{let } x = 3 \text{ in } x * x \text{ end}'$ is 18
 - The semantic value of $'\text{let } x = 1 \text{ in let } y = x + 1 \text{ in } y \text{ end end}'$ is 2

Exercises

- Use the semantics to compute the meaning of the following expressions in TWO (use the rules given in the notes, the book has many typos):

1) `let val y = 3 in 2*y end`

2) `let val y = 1 in
 let val y = 2 in
 y
 end
end`

Note: first construct an abstract syntax tree, then give the representation in Prolog notation, and then show the computation in our semantics.