

Defining Language TWO

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

TWO: Syntax

TWO:

```
<exp>* ::= <exp> + <mulexp> | <mulexp>  
<mulexp> ::= <mulexp> * <rootexp> | <rootexp>  
<rootexp> ::= let val <variable> = <exp> in <exp> end  
           | (<exp>) | <variable> | <constant>
```

- A subset of ML expressions
- This grammar is unambiguous
- A sample Language TWO expression:
`let val y = 3 in y*y end`

TWO: Abstract Syntax

Additional abstract syntax nodes for language TWO:

- (1) `var(X)` refers or dereferences a variable X
- (2) `let(X,E1,E2)` defines or 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, contexts.

In our case, for the Prolog based semantics, we let the terms `bind(X,K)` represent the binding of variables `X` to values `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

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

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

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

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

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

`val2 / 3` - interpretation predicate,
first argument: AST; second
argument: context; third
argument: semantics 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
```

TWO: Natural Semantics

As before, we will write a natural semantics to capture the basic proof rules

We will again use the symbol \rightarrow for this relation, though it is a different relation...

We will write $\langle E, C \rangle \rightarrow v$ to mean that the value of the AST E in context C is v

TWO: Natural Semantics

$$\frac{\langle E_1, C \rangle \rightarrow v_1 \quad \langle E_2, C \rangle \rightarrow v_2}{\langle \mathbf{plus}(E_1, E_2), C \rangle \rightarrow v_1 + v_2} \quad \langle \mathbf{var}(v), C \rangle \rightarrow \mathit{lookup}(C, v)$$

$$\frac{\langle E_1, C \rangle \rightarrow v_1 \quad \langle E_2, C \rangle \rightarrow v_2}{\langle \mathbf{times}(E_1, E_2), C \rangle \rightarrow v_1 \times v_2} \quad \langle \mathbf{const}(n), C \rangle \rightarrow \mathit{eval}(n)$$

$$\frac{\langle E_1, C \rangle \rightarrow v_1 \quad \langle E_2, \mathit{bind}(x, v_1) :: C \rangle \rightarrow v_2}{\langle \mathbf{let}(x, E_1, E_2), C \rangle \rightarrow v_2}$$

This still needs definitions for $+$, \times and eval , as well as bind , lookup , $::$, and the nil environment