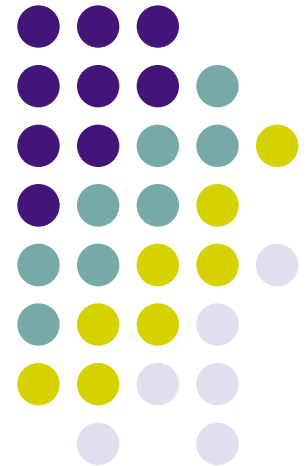
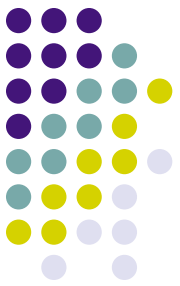


CSC501

Semester Review





String Rewriting Systems

Definition: [String Rewriting System] A *string rewriting system* is a tuple (Γ, \rightarrow) where,

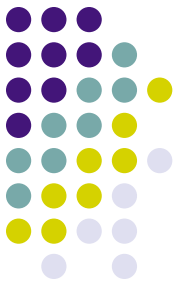
- Γ is an *alphabet*.
- \rightarrow is a binary relation in Γ^* , i.e., $\rightarrow \subseteq \Gamma^* \times \Gamma^*$. Each element $(u, v) \in \rightarrow$ is called a (*rewriting*) *rule* and is usually written as $u \rightarrow v$.

An *inference step* in this formal system is: given a string $u \in \Gamma^*$ and a rule $u \rightarrow v$ then the string u can be *rewritten* as the string $v \in \Gamma^*$. We write,

$$u \Rightarrow v.$$

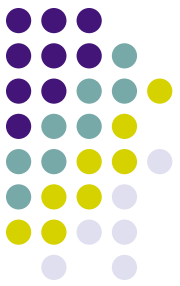
Note: Rule definitions, $u \rightarrow v$, and rule applications or inference steps, $u \Rightarrow v$, are two separate things and we use different symbols.

Grammars



Definition: [Grammar] A *grammar* is a triple $(\Gamma, \rightarrow, \gamma)$ such that,

- $\Gamma = T \cup N$ with $T \cap N = \emptyset$, where T is a set of symbols called the *terminals* and N is a set of symbols called the *non-terminals*,¹
- \rightarrow is a set of rules of the form $u \rightarrow v$ with $u, v \in \Gamma^*$,
- γ is called the *start symbol* and $\gamma \in N$.



Natural Semantis

Arithmetic
Expressions:

$$\frac{}{(n, \sigma) \mapsto eval(n)} \quad \text{for } n \in \mathbb{I}$$

$$\frac{}{(x, \sigma) \mapsto \sigma(x)} \quad \text{for } x \in \mathbf{Loc}$$

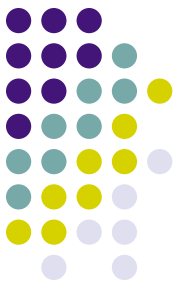
$$\frac{(a_0, \sigma) \mapsto k_0 \quad (a_1, \sigma) \mapsto k_1}{(a_0 + a_1, \sigma) \mapsto k} \quad \text{where } k = k_0 + k_1$$

$$\frac{(a_0, \sigma) \mapsto k_0 \quad (a_1, \sigma) \mapsto k_1}{(a_0 - a_1, \sigma) \mapsto k} \quad \text{where } k = k_0 - k_1$$

$$\frac{(a_0, \sigma) \mapsto k_0 \quad (a_1, \sigma) \mapsto k_1}{(a_0 * a_1, \sigma) \mapsto k} \quad \text{where } k = k_0 \times k_1$$

$$\frac{(a, \sigma) \mapsto k}{((a), \sigma) \mapsto k}$$

with $k, k_0, k_1 \in \mathbb{I}$, $a, a_0, a_1 \in \mathbf{Aexp}$, and $\sigma \in \Sigma$.

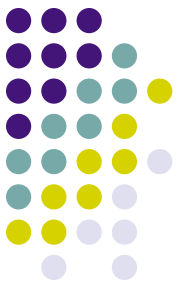


Induction

Proposition: (Mathematical Induction) Let P be a predicate over the natural numbers \mathbb{N} , then

$$\forall n \in \mathbb{N}.P(n) \text{ iff } P(0) \wedge \forall n \in \mathbb{N}.P(n) \Rightarrow P(n + 1).$$

Here, $P(0)$ is called the *basis*, $P(n)$ is the *induction hypothesis*, and $P(n) \Rightarrow P(n + 1)$ is called the *inductive step*.



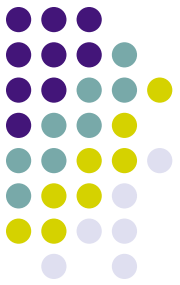
Structural Induction

Given the ordering of the terms we can now state our *structural induction principle* to show that some predicate P holds for all arithmetic expressions:

$$\begin{aligned} \forall a \in \mathbf{Aexp}. P(a) \quad \text{iff} \quad & (\forall n \in \mathbf{I}. P(n)) \wedge \\ & (\forall x \in \mathbf{Loc}. P(x)) \wedge \\ & (\forall a_0, a_1 \in \mathbf{Aexp}. P(a_0) \wedge P(a_1) \Rightarrow P(a_0 + a_1)) \wedge \\ & (\forall a_0, a_1 \in \mathbf{Aexp}. P(a_0) \wedge P(a_1) \Rightarrow P(a_0 - a_1)) \wedge \\ & (\forall a_0, a_1 \in \mathbf{Aexp}. P(a_0) \wedge P(a_1) \Rightarrow P(a_0 * a_1)) \wedge \\ & (\forall a \in \mathbf{Aexp}. P(a) \Rightarrow P(a)) \end{aligned}$$

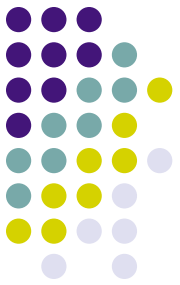
As expected, here we also take advantage of the precise ordering of terms and their sub terms and therefore the domino effect also works here.

Prolog Semantics



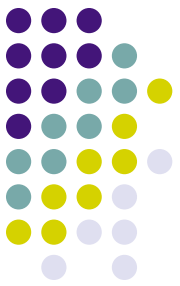
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
% semantics of arithmetic expressions  
  
(C,_) -->> C :-                % constants  
    int(C),!.  
  
(X,State) -->> Val :-          % variables  
    atom(X),  
    lookup(X,State,Val),!.  
  
(add(A,B),State) -->> Val :-   % addition  
    (A,State) -->> ValA,  
    (B,State) -->> ValB,  
    Val xis ValA + ValB,!.  
  
(sub(A,B),State) -->> Val :-   % subtraction  
    (A,State) -->> ValA,  
    (B,State) -->> ValB,  
    Val xis ValA - ValB,!.  
  
(mult(A,B),State) -->> Val :-  % multiplication  
    (A,State) -->> ValA,  
    (B,State) -->> ValB,  
    Val xis ValA * ValB,!.  

```



Prolog Semantics

- Executable Specs/Prolog Specs:
 - state, arithmetic expressions
 - boolean expressions, commands
 - declarations, type systems
 - I/O, block structured languages
 - functions
 - program correctness
 - pre- and postconditions
 - program correctness and iteration
 - loop invariants
 - program correctness and recursive functions
 - translational semantics
 - translation, source and target semantics
 - compiler correctness



Elements of Model Theory

- In terms of programming language semantics, let P be a description of a programming language model, let M be the intended model, then because of soundness and completeness, any characteristic c about our programming language that can be deduced from P will also be true in the intended model,

$$P \vdash c \Rightarrow M \models c$$

and any characteristic c that is true in M can be proven,

$$M \models c \Rightarrow P \vdash c$$

- That means, we are justified to use Prolog as a theorem prover to prove characteristics about our programming language models.