CS520 Lecture 2
Predicate Logic

January 12, 2020

2.1 Motivation or Objective

1. Learn four key tools in PL that will be used throughout this course

    1. Abstract Syntax
    2. Denotational Semantics
    3. Inference rule
    4. Binding

2. Learn the basics of predicate logic (or first-order logic)
3. We plan to go through some of (1)-(4) twice, first using integer expressions and then using predicate logic.

2.2 Integer Expressions

1. How to analyze integer expressions found in logic and programming languages mathematically? We will first have to define the syntax and the semantics for them.
2. Examples : $x + 3 \times y, x \div 2 + x \times x$
3. We also want sto develop mathematical tools to reason about or manipulate integer expressions

2.3 Abstract Syntax and Initial Algebra

1. Abstract Syntax
Specification of $\underline{\text{abstract phrases}}$[1] in a formal language, such as the language of integer expressions and $\underline{\text{predicte logics.}}$
2. Typically, we use $\underline{\text{abstract grammar}}$[2] to describe abstract syntax.
3. Abstract grammar for integer expressions:

$$\begin{aligned}\langle\text{intexp}\rangle ::= {} & 0|1|2|\ldots \\ & |\langle\text{var}\rangle| - \langle\text{intexp}\rangle|\langle\text{intexp}\rangle + \langle\text{intexp}\rangle \\ & |\langle\text{intexp}\rangle - \langle\text{intexp}\rangle|\langle\text{intexp}\rangle \times \langle\text{intexp}\rangle \\ & |\langle\text{intexp}\rangle \div \langle\text{intexp}\rangle|\langle\text{intexp}\rangle \text{ rem } \langle\text{intexp}\rangle\end{aligned}$$

(abstract) Integer expressions are finite derivation trees in this grammar. For instance, Note that infinite trees are not included.

[1] Vague words, but will be made rigorous when we define initial algebra
[2] Not accurate, but good approximate view.
(1) grammar without any concren on parsing or surface syntax
(2) In this case, parse trees in the grammar are abstract phrases

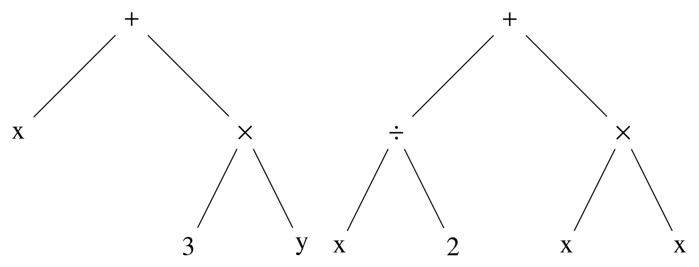


Figure 2.1: Finite derivation trees for integer expressions

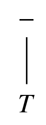
4. A more accurate views is to view abstract syntax as an initial algebra. This view will help us to see why we can define various operations on abstract phrases or integer expressions using syntax-directed definition.
5. Algebra A : Set with operations and constants.
Signature S : Type of an algebra

1. $S_{group} = (t, id : t, \circ : t \times t \to t, (-)^{-1} : t \to t)$
2. $A_0 = (\mathbb{Z}, 0 \in \mathbb{Z}, + : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}, - : \mathbb{Z} \to \mathbb{Z})$
3. $A_1 = (\mathbb{R}_{>0}, 1 \in \mathbb{R}_{>0}, \times : \mathbb{R}_{>0} \times \mathbb{R}_{>0} \to \mathbb{R}_{>0}, (-)^{-1} : \mathbb{R}_{>0} \to \mathbb{R}_{>0})$
   $A_0 : S_{group}, A_1 : S_{group}$

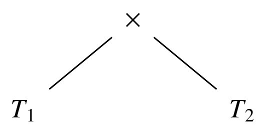
$$\begin{aligned}
S_{intexp} = (t, 0 : t, 1 : t, \ldots, x : t, y : t, \ldots, \\
- : t \to t, +, \times, -, \div : t \times t \to t) \\
A_{grammar} = (FinTrees, 0 \in FinTrees, \ldots, \\
x \in FinTrees, \ldots, \\
- \in [FinTrees \to FinTrees] \\
\times \in [FinTrees \to FinTrees])
\end{aligned}$$

6. Algebra Homomorphism

$T \mapsto$



$(T_1, T_2) \mapsto$



)

Map between algebras that preserves constants and operations.

$$\begin{aligned}
S &= (t, c_1 : t, \ldots, c_n : t, \ldots, \\
&\quad op_1 : t \times \ldots \times t \to t, \ldots, op_m : t \times \ldots \times t \to t) \\
A_0 &= (\mu_0 (= |A_0| \text{ (notation)}), c_1^0 \in \mu_0, \ldots, c_n^0 \in \mu_0, \\
&\quad op_1^0 \in \mu_0 \times \ldots \times \mu_0 \to \mu_0, \ldots, op_m^0 \in \mu_0, \times \ldots, \times \mu_0 \to \mu_0) \\
A_1 &= (\mu_1, c_1^1 \in \mu_1, \ldots, c_n^1 \in \mu_1, \\
&\quad op_1^1 \in \mu_1 \times \ldots \times \mu_1 \to \mu_1, \ldots, op_m^1 \in \mu_1, \times \ldots, \times \mu_1 \to \mu_1)
\end{aligned}$$

$f \in \mu_0 \to \mu_1$ is a homomorphism if

$$\begin{aligned}
&(1) \forall j. f(c_j^0) = c_j^1 \\
&(2) \forall i. f(op_j^0(x_1, \ldots, x_k)) = op_j^1(f(x_1, \ldots, f(x_k)))
\end{aligned}$$

7. Initial Algebra of a signature S

- An algebra A of the signature S s.t.
  for all algebras A' of the same signature, there is a $\underline{\text{unique}}$ homomorphism f from A to A'.
- $A_{grammar}$ is initial.
- Formally, an abstract syntax fixes a signature, and it denotes an initial algebra of the signature. An abstract phrase is an element of that algebra.

Exercise : Prove that $A_{grammar}$ is indeed an initial algebra.
Exercise : Let A_0, A_1 be initial algebras of the same signatures S.
Then, there are homomorphisms $f \in |A_0| \to |A_1|$ and $g \in |A_1| \to |A_0|$ s.t. $f \circ g = id, g \circ f = id$.
This means that all initial algebras of S are essentially the same, i.e. isomorphic. Prove this fact.

2.4 Syntax-directed definition and denotational semantics

1. Defenition of a map on integer expressions using a form of induction and case analysis
2. $FV(e) = V$ (so, $FV : \langle\text{intexp}\rangle \to 2^{\langle\text{var}\rangle}$)
FV : free variables, e : integer expression, V : set of free variables in e.

$$\begin{aligned}
FV(c) &= \emptyset \text{ c is a constant} FV(x) = \{x\} \text{ x is a variable} \\
FV(-e) &= FV(e) FV(e_1 + e_2) = FV(e_1) \cup FV(e_2)
\end{aligned}$$

3. Two features : case analysis, recursive calls on subphrases 4. $[\![-]\!] \in \langle\text{intexp}\rangle \to \Sigma \to \mathbb{Z}$
where $\Sigma = \langle\text{var}\rangle \to \mathbb{Z}$, a set of states σ.

$$
\begin{aligned}
[\![c]\!]\sigma &= c\\
[\![x]\!]\sigma &= \sigma(x)\\
[\![-e]\!]\sigma &= -[\![e]\!]\sigma\\
[\![e_1 + e_2]\!]\sigma &= [\![e_1]\!]\sigma + [\![e_2]\!]\sigma
\end{aligned}
$$

Intuitively, $[\![-]\!]$ maps trees to mathematical functions in a syntax directed (also called compositional) way. Such a compositional mapping from syntactic entities to mathematical entities is called denotational semantics.
5. In both cases, we are using the initiality of ¡intexp¿. What are the target algebras in those cases?
(1)

$$
\begin{aligned}
&|A_1| = 2^{\langle\text{var}\rangle}\\
&c^1 = \emptyset\\
&x^1 = \{x\}\\
&-^1\,(X) = X\\
&+^1\,(X, Y) = X \cup Y \qquad\qquad \times^2, \div^2,\ \text{rem}^{\,2}\ \text{defined similarly}
\end{aligned}
$$

(2)

$$
\begin{aligned}
&|A_2| = \Sigma \to \mathbb{Z}\\
&c^2(\sigma) = c\\
&x^2(\sigma) = \sigma(x)\\
&-^2\,(f)(\sigma) = -(f)(\sigma)\\
&+^2\,(f, g)(\sigma) = f(\sigma) + g(\sigma) \qquad\qquad \times^2, \div^2,\ \text{rem}^{\,2}\ \text{defined similarly}
\end{aligned}
$$

6. Exercise
$\delta \in \langle\text{var}\rangle \to \langle\text{intexp}\rangle$ is substitution.
Define ${}^{e}/_{\delta}$, the application of substitution δ to e.

2.5 Structural induction

1. We want to show that some property ϕ holds for all integer expressions. What should we do?
2. Use induction on the structure of expressions.
That is, for each expression e, prove the property assuming that the property holds for the subexpressions of e.

Lemma 1 (Coincidence)
For every expression e and states σ, σ'*,*
if $\forall x \in FV(e)\sigma(x) = \sigma'(x)$
then $[\![e]\!]\sigma = [\![e]\!]\sigma$

Proof. By structural induction,

- $e \equiv c : [\![c]\!]\sigma = c = [\![c]\!]\sigma'$
- $e \equiv x : [\![x]\!]\sigma = \sigma(x) = \sigma'(x) = [\![x]\!]\sigma'$
  because $x \in FV(x)$
- $e \equiv -e' : FV(e) = FV(e')$
  By induction hypothesis, $[\![e']\!]\sigma = [\![e']\!]\sigma'$
  $[\![e]\!]\sigma = -[\![e']\!]\sigma = -[\![e']\!]\sigma' = [\![e]\!]\sigma'$
- $e_1 \times e_2, e_1 \div e_2, e_1$ rem e_2

□

Lemma 2 (Substitution)
If $\forall x.\sigma(x) = [\![\sigma(x)]\!]\sigma'$
$[\![{}^e\!/\!_\delta]\!]\sigma' = [\![e]\!]\delta$

Proof. By structural induction. □

Notation : (1) :$x_1 \to e_1, \ldots, x_n \to e_n$
means the substitutuion that maps x_i to e_i and $y \neq x_i$ to y.

$$(2) :[\delta|x : v](y) = \begin{cases} \delta(y) & \text{if } y \neq x \\ v & \text{if } x = y \end{cases}$$

Corollary 1
$[\![{}^e\!/\!_{x_1 \to e_1, \ldots, x_n \to e_n}]\!]\sigma = [\![e]\!][\delta|x_1 : [\![e_1]\!]\delta, | \ldots |x_n : [\![e_n]\!]\delta]$

This intuitively says the correspondence between syntactic and semantic substitutions.
3. Structural induction holds because of the initiality of ¡intexp¿. Can you explain why it is the case?

2.6 Predicate logic (first-order logic) informally

1. Language for expressing (boolean) properties (also called assertions).
2. Extensions of boolean expressions in programming languages with universal and existential quantifications.
3. Examples

- $\forall x.\forall y.\exists m.\exists n.x \times m + y \times n = 1$

- $\forall x.\exists y.y > x$

4. Quantifiers are over integers, reals, and other first-order entities (i.e. not over sets of integers, and functions, etc). The "first-order" in first-order logic refers to this restriction.
We will consider a version of predicate logic or first-order logic where quantifiers range over integers and all variables are integer variables.

2.7 Abstract Syntax of predicate logic

1. Described in terms of the following abstract grammar :

$$
\begin{aligned}
\langle \text{intexp} \rangle &::= 0|1|2|\ldots \\
&\quad |\langle \text{var} \rangle | -\langle \text{intexp} \rangle | \langle \text{intexp} \rangle \begin{matrix} + \\ - \\ \times \\ \div \\ \text{rem} \end{matrix} \langle \text{intexp} \rangle \\
\langle \text{assert} \rangle &::= \mathit{true} | \mathit{false} | \langle \text{intexp} \rangle \begin{matrix} = \\ \neq \\ < \\ \leq \\ > \\ \geq \end{matrix} \langle \text{intexp} \rangle \\
&\quad |\neg \langle \text{assert} \rangle | \langle \text{assert} \rangle \begin{matrix} \vee \\ \wedge \\ \Rightarrow \\ \Leftarrow \end{matrix} \langle \text{assert} \rangle \\
&\quad | \begin{matrix} \forall \\ \exists \end{matrix} \langle \text{var} \rangle . \langle \text{assert} \rangle
\end{aligned}
$$

To simplify presentation, we will consider only $+, \times, \wedge, \forall, =, >$.
2. What do we mean by abstract grammar and abstract syntax here?
If you got confused about initial-algebra stuff, just think that our abstract syntax is the set of all finite derivation or parse trees.
If not, you can view the abstract syntax as the initial algebra of the signature induced by the grammar.

Signature

$$
\begin{aligned}
S_{PL} = (&t^{\swarrow \text{integer exps}}, u^{\swarrow \text{assertions}}), 0 : t, 1 : t, \dots,\\
&x : t, y : t, \dots,\\
&- : t \to t, + : t \times t \to t, \times : t \times t \to t\\
&true : u, false : u, =: t \times t \to u, <: t \times t \to u\\
&\neg : u \to u, \wedge : u \times u \to u,\\
&\forall : \langle \text{var} \rangle \times u \to u
\end{aligned}
$$

Algebra

$$
\begin{aligned}
A_0 = (&\mu^0, \nu^0, 0^0 \in \mu^0, 1 \in \mu^0, \dots,\\
&x^0 \in \mu^0, y^0 \in \mu^0, \dots,\\
&-^0 \in [\mu^0 \to \mu^0], \begin{matrix} +^0 \\ \times^0 \end{matrix} \in [\mu^0 \times \mu^0 \to \mu^9],\\
&true^0 \in \nu^0, false^0 \in \nu^0, \begin{matrix} =^0 \\ <^0 \end{matrix} \in [\mu^0 \times \mu^0 \to \nu^0],\\
&\neg^0 \in [\nu^0 \to \nu^0], \wedge^0 \in [\nu^0 \times \nu^0 \to \nu^0],\\
&\forall^0 \in [\langle \text{var} \rangle \times \nu^0 \to \nu^0])\\
A_1 = (&\mu^1, \nu^1, \dots, =^1 \in [\mu^1 \times \mu^1 \to \nu^1],\\
&\forall^1 \in [\langle \text{var} \rangle \times \nu^1 \to \nu^1])
\end{aligned}
$$

An algebra homomorhpism from A_0 to A_1 is a pair $(h : \mu^0 \to \mu^1, k : \nu^0 \to \nu^1)$ that preserves all constants and operations. For instance,

$$
k(=^0 (a, b)) = =^1 (h(a), h(b)) \text{ for all } a, b \in \mu^0
$$
$$
\text{for any } x \in \langle \text{var} \rangle, a \in \mu^0, k(\forall^0(x, a)) = \forall^1(x, k(a))
$$

As before, the abstract syntax is the initial algebra of th signature S_{PL} [3]. because of initiality, we can define a function on assertions in a syntax-directed way. Also, we can use structural induction to prove properties about assertions.

2.8 Denotational Semantics

1. We define two functions :

$$
[\![-]\!]_{intexp}{}^{4} \in [\langle \text{intexp} \rangle \to \overbrace{\Sigma}^{\langle \text{var} \rangle \to \mathbb{Z}} \to \mathbb{Z}]
$$
$$
[\![-]\!]_{assert}{}^{5} \in [\langle \text{assert} \rangle \to \Sigma \to \underbrace{\mathbb{B}}_{\{tt, ff\}}]
$$

[3]Isomorphic to the algebra built with derivation trees

$$\frac{p_0 \quad p_0 \Rightarrow p_1}{p_1} \qquad \frac{p}{\forall x.p}$$

$$\frac{}{e_0 = e_1 \Rightarrow e_1 = e_0} \qquad \frac{p_0 \quad p_1 \quad \dots \quad p_n}{p}$$

2. The definition of $[\![-]\!]_{intexp}$ is the same as before. Recall that it is syntax-directed.
3. The definition of $[\![-]\!]_{assert}$ is as follows:

$$[\![true]\!]\sigma = tt \qquad [\![false]\!]\sigma = ff$$

$$[\![e_0 \overset{=}{\underset{<}{}} e_1]\!]\sigma = ([\![e_0]\!]_{intexp}\sigma \overset{=}{\underset{<}{}} [\![e_1]\!]_{intexp}\sigma)$$

$$[\![\neg p]\!]\sigma = (\neg[\![p]\!]\sigma)$$

$$[\![p_1 \wedge p_2]\!]\sigma = ([\![p_1]\!]\sigma \wedge [\![p_2]\!]\sigma)$$

$$[\![\forall x.p]\!]\sigma = (\forall n \in \mathbb{Z}.[\![p]\!][\sigma|x : n])$$

4. Don't forget that what appears inside $[\![-]\!]$ is a tree, while $\forall$ and $\wedge$ etc on the RHS of = are the usual mathematical notations.
5. As we discussed already, here we are really defining an algebra A of S_{PL} s.t.

$$A = (\Sigma \to \mathbb{Z}, \Sigma \to \mathbb{B}, \dots)$$

Then, we are using the initiality of the abstract syntax to get a map from it to A.

2.9 Inference Rules

1. Rules for deriving always-true (In other words, valid[6]) assertions.
2. Expressed using the inference-rule notation.

- general form
- expressions that if all of $p_0, \dots, p_n$ are valid, then p is valid.
- doesn't say that for all $\sigma \in \Sigma$, if $[\![p_0]\!]\sigma = \dots = [\![p_n]\!]\sigma = tt$ then $[\![p]\!]\sigma = tt$.

Exercise : Prove why the above three rules are correct[7]
3. A big part of research on or study about predicate logic is to study these rules.
In this course, however, we will not do this.

[5]We will omit subscripts most of the time.
[6]p is valid if $[\![p]\!]\sigma = tt$ for all $\sigma \in \Sigma$
[7]also called sound

2.10 Binding and substitution

1. $\forall v, \exists v$

- examples of binders.
- They have the scope of binding.
- Informally, they introduce a new variable, give a name v to it, and make it available within its scope[8]. Morally, renaming v to w should not change the meaning of the assertion.

2. An occurence of a variable x is bounded in p if x is within the scope of a binder for x in p.
3. An occurence of x is free in p if it is not bound in p.
4. A variable x is free in p if it has a free occurence in p.
5. We can define functions FV_{assert} and FV_{intexp} that compute the set of free variables of assertions and integer expressions in a syntax-directed way.

$$\begin{aligned}
&FV_{intexp} : \text{ defined as before}\\
&FV_{assert}(true) = FV_{assert}(false) = \emptyset\\
&FV_{assert}(e_0 \underset{<}{=} e_1) = FV_{intexp}(e_0) \cup FV_{intexp}(e_1)\\
&FV_{assert}(\neg p) = FV_{assert}(p)\\
&FV_{assert}(p_1 \wedge p_2) = FV_{assert}(p_1) \cup FV_{assert} p_2\\
&FV_{assert}(\forall v.p) = FV_{assert}(p) \setminus \{v\}
\end{aligned}$$

Exercise. Define an algebra for S_{PL} such that the algebra homomorphism for the abstract syntax to this algebra is $(FV_{intexp}, FV_{assert})$
6. Now, how should we deal with binders during substitution? It is not entirely obvious. Several textbooks had wrong definitions in old days.
Mistakes : $({}^{(\forall y. y > x)}/_{x \mapsto y+1}) = (\forall y. y > y + 1)$
Correct definition

$${}^{true}/_{\delta} = true \qquad\qquad {}^{false}/_{\delta} = false$$

$${}^{e_0 \underset{<}{=} e_1}/_{\delta} = {}^{e_0}/_{\delta} \underset{<}{=} {}^{e_1}/_{\delta} \qquad\qquad {}^{\neg p}/_{\delta} = \neg {}^{p}/_{\delta}$$

$${}^{p_1 \wedge p_2}/_{\delta} = {}^{p_1}/_{\delta} \wedge {}^{p_2}/_{\delta}$$

$${}^{\forall v.p}/_{\delta} = \forall v_{new}.({}^{p}/_{[\delta | v : v_{new}]})^{9}$$

where $v_{new} \notin \cup_{w \in FV(p)\setminus\{v\}} FV(\delta(w))$

If v is not in the set, then $v_{new} = v$. Otherwise, v_{new} is the first variable not in the set.

Proposition 1
Coincidence p : assertion or integer expression
δ, δ' : states s.t. $\delta\omega = \delta'\omega$ for all $\omega \in FV(p)$
$\Rightarrow$
$[\![p]\!]\delta = [\![p]\!]\delta'$

[8]binding, which happens frequently in a programming language.

Proof. By structural induction.
We can use it because the abstract syntax for predicate logic is an initial algebra.

- true or false : obvious
- $p \equiv (e_1 \overset{=}{\underset{<}{}} e_2)$:

$$\begin{aligned}&FV(e_i) \subseteq FV(p)\\ &\forall \omega \in FV(e_i).\delta\omega = \delta'\omega\\ &\text{We can use induction and get } [\![e_i]\!]\delta = [\![e_i]\!]\delta'\end{aligned}$$

$$\begin{aligned}[\![e_1 < e_2]\!]\delta \qquad &= [\![e_1]\!]\delta < [\![e_2]\!]\delta\\ &= ([\![e_1]\!]\delta' < [\![e_1]\!]\delta')\\ &= ([\![e_1 < e_2]\!]\delta')\end{aligned}$$

- $p \equiv \neg p'$ or $p \equiv p_1 \wedge p_2$: similar proof.
- $p \equiv \forall x.p'$: For all $n \in \mathbb{Z}$

$$\sigma_1 := [\sigma|x : n] \text{ and } \sigma'_1 = [\sigma'|x : n]$$

Then, $\forall \omega \in FV(p')$

$$\sigma_1(\omega) = \sigma'_1(\omega)$$

By induction hypothesis, $[\![p']\!]\sigma_1 = [\![p']\!]\sigma'_1$

$$\begin{aligned}[\![\forall x.p']\!]\sigma &= (\forall n \in \mathbb{Z}.[\![p']\!][\sigma|x : n])\\ &= \forall n \in \mathbb{Z}, [\![p']\!][\sigma'|x : n] = [\![\forall x.p']\!]\sigma'\end{aligned}$$

□

Proposition 2 (Substitution)

$$\sigma\omega = [\![\delta\omega]\!]\sigma' \Rightarrow [\![p]\!]\sigma = [\![^{p}/_{\delta}]\!]\sigma'$$

Proof. By structural induction. □

Proposition 3 (Finite substitution theorem)

$$[\![^{p}/_{v_0 \to e_0, \ldots, v_{n-1} \to e_{n-1}}]\!]\sigma' = [\![p]\!][\sigma'|v_0 : [\![e_0]\!]\sigma'| \ldots |v_{n-1} : [\![e_{n-1}]\!]\sigma']$$

Proof. An easy consequence of proposition 1.3. □

Correspondence between syntactic and semantic substitutions.

Proposition 4 (Renaming)
If $\omega \notin FV_{assert}(q) \setminus \{v\}$
then $[\![\forall \omega.(^{q}/_{v \to \omega})]\!] = [\![\forall v.q]\!]$

Renaming doesn't change the meaning of an assertion.