4.2 Universality of Logic Programming

Freitag, 2. Juni 2017 13:00

We now know the Syntax + semantics of (pure) logic programming.

In this section, we show that LP is a universal (= Turing-complete) programming language.

In other words, it is as expressive as Java, C, Haskell, ...

Thus: every computable function can also be computed by a logic program.

What are the computable functions?

$\left.\begin{array}{l}\text{Turing: Turing Machines}\\ \text{Church: Lambda Terms}\\ \text{Kleene: } \mu\text{-recursive Functions}\end{array}\right\}$ Yield the same set of functions

Church's Thesis: every sensible formalism to define computability yields the same set of computable functions

Here: we show that every μ-recursive function can be computed by a logic program

We only regard functions on $\mathbb{N}$.

μ-recursive functions are defined by 6 rules to characterize all computable functions.

Rules 1-3 define basic functions.
Rules 4-6 allow us to construct new computable functions from existing ones.

<u>Def 4.2.1</u> (μ-recursive Functions) Slide 23

The set of <u>μ-recursive functions</u> is the smallest set of functions with:

1. For all $n \in \mathbb{N}$, the function $null_n : \mathbb{N}^n \to \mathbb{N}$ with $null_n(k_1, \dots, k_n) = 0$ is μ-recursive.
2. The successor function $succ : \mathbb{N} \to \mathbb{N}$ with $succ(k) = k+1$ is μ-recursive.
3. For all $n \geq 1$ and all $1 \leq i \leq n$, the projection function $proj_{n,i}(k_1, \dots, k_n) = k_i$ is μ-recursive.
4. The μ-recursive functions are closed under composition: For all $m \geq 1$ and all $n \geq 0$ we have: If $f : \mathbb{N}^m \to \mathbb{N}$, $f_1, \dots, f_m : \mathbb{N}^n \to \mathbb{N}$ are μ-recursive, then the following function $g : \mathbb{N}^n \to \mathbb{N}$ is also μ-recursive:
$$g(k_1, \dots, k_n) = f(f_1(k_1, \dots, k_n), \dots, f_m(k_1, \dots, k_n))$$
5. The μ-recursive functions are closed under <u>primitive recursion</u>: For all $n \geq 0$ we have:

<u>primitive recursion</u>: For all $n \geq 0$ we have:

If $f: \mathbb{N}^n \to \mathbb{N}$ and $g: \mathbb{N}^{n+2} \to \mathbb{N}$ are μ-recursive, then the following fct. $h: \mathbb{N}^{n+1} \to \mathbb{N}$ is also μ-recursive:

$$h(k_1, \ldots, k_n, 0) = f(k_1, \ldots, k_n)$$

$$h(k_1, \ldots, k_n, k+1) = g(k_1, \ldots, k_n, k, h(k_1, \ldots, k_n, k))$$

6. The μ-recursive functions are closed under (unbounded) minimalization:

   If $f: \mathbb{N}^{n+1} \to \mathbb{N}$ is μ-recursive, then the following fct. $g: \mathbb{N}^n \to \mathbb{N}$ is also μ-recursive:

   $g(k_1, \ldots, k_n) = k$ iff $f(k_1, \ldots, k_n, k) = 0$ and for all $0 \leq k' < k$, $f(k_1, \ldots, k_n, k')$ is defined and $f(k_1, \ldots, k_n, k') > 0$.

   If there is no such k, then $g(k_1, \ldots, k_n)$ is undefined.

   ↑ In an imperative language, g is easy to implement: initialize k with 0 and increase it repeatedly until $f(k_1, \ldots, k_n, k) = 0$.

The class of functions that can be constructed with the principles 1–5 are the <u>primitive recursive</u>

the principles 1–5 are the <u>primitive recursive functions.</u>

There are computable functions that are not primitive recursive:

- partial functions
- total functions like the Ackermann function

Ex. 4.22

Slide 24

The addition function plus : $\mathbb{N}^2 \to \mathbb{N}$ is primitive recursive:

$f(x,y,z) = succ(proj_{3,3}(x,y,z))$	$f(x,y,z) = z+1$
$plus(x,0) = proj_{1,1}(x)$	$plus(x,0) = x$
$plus(x,y+1) = f(x,y,plus(x,y))$	$plus(x,y+1) = plus(x,y)+1$

The multiplication function times : $\mathbb{N}^2 \to \mathbb{N}$ is also primitive recursive:

$g(x,y,z) = plus(proj_{3,1}(x,y,z), proj_{3,3}(x,y,z))$	$g(x,y,z) = x + z$
$times(x,0) = null_1(x)$	$times(x,0) = 0$
$times(x,y+1) = g(x,y,times(x,y))$	$times(x,y+1) = x + times(x,y)$

The predecessor function $p : \mathbb{N} \to \mathbb{N}$ is prim. rec., where $p(0) = 0$ and $p(x+1) = x$.

$p(0) = null_0$

$p(x+1) = proj_{2,1}(x, p(x))$

The subtraction fct. minus: $\mathbb{N}^2 \to \mathbb{N}$ is prim. rec., where minus$(x,y) = 0$ if $x \leq y$ and

minus$(x,y) = x - y$ otherwise.

More precisely: $\text{minus}(x,y) = \underbrace{p(p(\ldots p}_{y \text{ times}}(x)))$

$h(x,y,z) = p(proj_{3,3}(x,y,z))$

$\text{minus}(x,0) = proj_{1,1}(x)$ $\leftarrow x$

$\text{minus}(x,y+1) = h(x,y,\text{minus}(x,y))$ $\leftarrow p(\text{minus}(x,y))$

Finally, we show that div: $\mathbb{N}^2 \to \mathbb{N}$ is μ-recursive, where

$\text{div}(x,y) = \lceil \frac{x}{y} \rceil$ if $y \neq 0$

$\text{div}(0,0) = 0$

$\text{div}(x,0)$ undefined if $x > 0$

Idea: $\frac{x}{y} = z$ iff $x = y \cdot z$

iff $\underbrace{x - y \cdot z}_{i(x,y,z)} = 0$

We search for the smallest z where $i(x,y,z)$ is 0.

$\text{div}(x,y) = z$ iff $i(x,y,z) = 0$ and for all $0 \le z' < z$, $i(x,y,z')$ is defined and $i(x,y,z') > 0$

Here, i is primitive recursive:

$i(x,y,z) = minus(proj_{3,1}(\underline{x},y,z), j(x,y,z)) \Leftarrow x - y \cdot z$

$j(x,y,z) = times(proj_{3,2}(x,\underline{y},z), proj_{3,3}(x,y,\underline{z})) \Leftarrow y \cdot z$

The μ-recursive functions are exactly the computable functions. To show that logic programming is Turing-complete, we have to show that every μ-rec. fct. can be computed by a logic program.

We first have to make clear when a log. prog. "computes" a function on nat. numbers.

Problems:

ⓐ • Logic prog. operate on terms, not on numbers.

ⓑ • Logic prog. implement relations/predicates, not functions.

Solution for ⓐ:

Represent numbers by terms over $\mathcal{O} \in \Sigma_0$ and $s \in \Sigma_1$.

$0 \;\hat{=}\; \mathcal{O}$

$$1 \mathrel{\hat{=}} s(0)$$
$$2 \mathrel{\hat{=}} s(s(0))$$
$$\vdots$$

Solution for (b):

We compute a function $f : \mathbb{N}^n \to \mathbb{N}$ by a predicate $\underline{f}$ of arity $n+1$.

<u>Def 4.2.3</u> (Computing arithmetic functions by Logic Programs)

- Every number $k \in \mathbb{N}$ is represented by the term $\underline{k} \in \mathcal{T}(\Sigma, \mathcal{V})$ with $\underline{k} = \underbrace{s(\ldots s(}_{k \text{ times}}0)\ldots)$, where $0 \in \Sigma_0$ and $s \in \Sigma_1$.

$\underline{0} = 0$
$\underline{1} = s(0)$
$\underline{2} = s(s(0))$
$\vdots$

- A logic prog $\mathcal{P}$ over (Σ, Δ) <u>computes</u> a function $f : \mathbb{N}^n \to \mathbb{N}$ iff there is a predicate symbol $\underline{f} \in \Delta_{n+1}$ such that

$$f(k_1, \ldots, k_n) = k \quad \text{iff} \quad \mathcal{P} \models \underline{f}(\underline{k_1}, \ldots, \underline{k_n}, \underline{k})$$

Then one can use the LP $\mathcal{P}$ to compute f's result by Scitas56 queries:

$$?- \underline{f}(\underline{k_1}, \dots, \underline{k_n}, X).$$

$$X = \underline{k}$$

<u>Ex. 4.2.4</u> The following logic program implements the functions from Ex. 4.2.2.

$\underline{plus}(X, 0, X).$

$\underline{plus}(X, s(Y), s(Z)) :- \underline{plus}(X, Y, Z).$

$\underline{times}(X, 0, 0).$

$\underline{times}(X, s(Y), Z) :- \underline{times}(X, Y, U), \underline{plus}(U, X, Z).$

$\underline{p}(0, 0).$

$\underline{p}(s(X), X).$

$\underline{minus}(X, 0, X).$

$\underline{minus}(X, s(Y), Z) :- \underline{minus}(X, Y, U), \underline{p}(U, Z).$

$\underline{div}(0, Y, 0).$

$\underline{div}(s(X), s(Y), s(Z)) :- \underline{minus}(X, Y, U), \underline{div}(U, s(Y), Z).$

This computes a partial function div, because $?- \underline{div}(1, 0, Z).$

fails. So we can implement partiality by failure or by non-termination.

Thm 4.2.5 (Universality of LP)

Every μ-recursive fct. can be computed by a logic program.

Proof: We prove the thm. by induction on the construction of the class of μ-rec. fcts.

1. The function $null_n$ can be computed by the following LP:

   $\underline{null_n}(X_1, \dots, X_n, 0).$

2. The fct. succ is implemented by

   $\underline{succ}(X, s(X)).$

3. The fct. $proj_{n,i}$ is implemented by

   $\underline{proj_{n,i}}(X_1, \dots, X_n, X_i).$

4. Composition can also be realized by a LP.
   By the ind. hyp. there is a LP with $\underline{f} \in \Delta_{m+1}$, $\underline{f_1}, \dots, \underline{f_m} \in \Delta_{n+1}$ that computes $f, f_1, \dots, f_m$.
   We extend this LP by the following rule:

$$\underline{g}(X_1, \dots, X_n, Z) :- \underline{f_1}(X_1, \dots, X_n, Y_1), \dots, \underline{f_m}(X_1, \dots, X_n, Y_m), \underline{f}(Y_1, \dots, Y_m, Z).$$

5. Prim. Recursion
   By the ind. hyp there is a LP with $\underline{f} \in \Delta_{n+1}$, $\underline{g} \in \Delta_{n+3}$ that computes f and g. We extend it by the following

two clauses:

$$\underline{h}(X_1,\dots,X_n,0,Z) :- \underline{f}(X_1,\dots,X_n,Z).$$

$$\underline{h}(X_1,\dots,X_n,s(X),Z) :- \underline{h}(X_1,\dots,X_n,X,Y), \quad \underline{g}(X_1,\dots,X_n,X,Y,Z).$$

6. Unbounded Minimalization

By the ind. hyp, there is a LP with $f \in \Delta_{n+2}$ which computes f. We extend it by the following rules.

Here $\underline{f}'(X_1,\dots,X_n,Y,Z)$ holds iff

$f(X_1,\dots,X_n,Z)=0$ and

for all $Y \le Z' < Z$ we have $f(X_1,\dots,X_n,Z')>0$

$$\underline{g}(X_1,\dots,X_n,Z) :- \underline{f}'(X_1,\dots,X_n,0,Z).$$

$$\underline{f}'(X_1,\dots,X_n,Y,Y) :- \underline{f}(X_1,\dots,X_n,Y,0).$$

$$\underline{f}'(X_1,\dots,X_n,Y,Z) :- \underline{f}(X_1,\dots,X_n,Y,s(U)), \quad \underline{f}'(X_1,\dots,X_n,s(Y),Z).$$

$\boxed{4}$

Ex 4.2.6 If one uses the construction of the above proof, then one would obtain the following LP for plus from the μ-recursive plus-function in Ex 4.2.2:

$proj_{3,3}(X, Y, Z, Z).$

$succ(X, s(X)).$

$f(X, Y, Z, V) :- proj_{3,3}(X, Y, Z, U), succ(U, V).$

$proj_{1,1}(X, X).$

$plus(X, 0, U) :- proj_{1,1}(X, U).$

$plus(X, s(Y), U) :- plus(X, Y, Z), f(X, Y, Z, U).$

If one applies the construction from the proof to the μ-recursive fct. div, then one gets:

$div(X1, X2, Z) :- i'(X_1, X_2, 0, Z).$

$i'(X_1, X_2, Y, Y) :- i(X_1, X_2, Y, 0).$

$i'(X_1, X_2, Y, Z) :- i(X_1, X_2, Y, s(U)),$
$\quad i'(X_1, X_2, s(Y), Z).$

$\vdots$ clauses for i