4.2 Universality of Logic Programming

Freitag, 22. Mai 2015 08:30

Goal: Show that LP is a Turing-complete language

for every computable function, there is a LP that computes it

⇝ LP is as powerful as C, Java, Haskell, ...

Defining computable functions (1930s):

- Turing: Turing machines
- Church: Lambda Calculus
- Kleene: μ-recursive functions

the set of computable functions is always the same

⇒ Church's thesis:

no prog. language can compute more functions than those expressible by Turing machines, λ-calculus, μ-recursion

Thus: to prove that LP is Turing-complete, show that for every μ-recursive function, there is a LP computing it.

All algebraic data structures (lists, trees, ...) can be encoded as natural numbers ⇒ only regard algorithms on natural numbers.

Def 4.2.1. (μ-recursive functions)

The set of μ-recursive functions is the smallest set of functions such that:

1. For every $n \in \mathbb{N}$, 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 every $n \geq 1$ and every $1 \leq i \leq n$, the projection function $proj_{n,i}(k_1, \dots, k_n) = k_i$ is μ-recursive.
4. μ-recursive functions are closed under composition: For all $m \geq 1$ and $n \geq 0$ we have: if $f : \mathbb{N}^m \to \mathbb{N}$ and $f_1, \dots, f_m : \mathbb{N}^n \to \mathbb{N}$ are μ-recursive, then the following fct. $g : \mathbb{N}^n \to \mathbb{N}$ is also μ-recursive:

$$g(k_1, \dots, k_n) = f\Big(f_1(k_1, \dots, k_n), \dots, f_m(k_1, \dots, k_n)\Big)$$

5. The μ-recursive functions are closed under <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:

$$h(k_1, .., k_n, 0) = f(k_1, .., k_n)$$

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

Functions that can be expressed with principles 1-5 are called <u>primitive recursive</u>.

There exist computable functions that are not primitive recursive:

- partial functions (implemented by programs that do not always terminate)
- certain total functions (e.g., the Ackermann function), but almost all total computable functions used in practice are primitive recursive.

6. μ-recursive functions are closed under unbounded minimization: For all $n \geq 0$ we have:

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:

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

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

Now we will show that every μ-recursive fct can be computed by a LP.

Ex 422 Consider some well-known computable fcts on $\mathbb{N}$ and show that they are μ-recursive.

- plus: $\mathbb{N}^2 \to \mathbb{N}$ is μ-recursive, even primitive recursive

$$plus(x,0) = proj_{1,1}(x)$$
$$plus(x,y+1) = \underbrace{f(x,y,plus(x,y))}_{plus(x,y)+1}$$

$$f(x,y,z) = succ(proj_{3,3}(x,y,z))$$

- times: $\mathbb{N}^2 \to \mathbb{N}$ is also primitive recursive

$$times(x,0) = null_1(x)$$

$$times(x,y+1) = \underbrace{g(x,y,times(x,y))}_{times(x,y)+x}$$

$$g(x,y,z) = plus\left(proj_{3,1}(x,y,z),\ proj_{3,3}(x,y,z)\right)$$

- The predecessor function is also primitive recursive:
  $p: \mathbb{N} \to \mathbb{N}$ with $p(0)=0$, $p(x+1)=x$

$$p(0) = null_0$$
$$p(x+1) = proj_{2,1}(x, p(x))$$

- The funct. minus: $\mathbb{N}^2 \to \mathbb{N}$ is also prim. recursive, where $minus(x,y) = 0$ if $x \le y$ and $minus(x,y) = x-y$ otherwise.

$$minus(x,0) = proj_{1,1}(x)$$
$$minus(x,y+1) = \underbrace{h(x,y,minus(x,y))}_{p(minus(x,y))}$$

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

- div: $\mathbb{N}^2 \to \mathbb{N}$ is also μ-recursive, where

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

Idea: $div(x,y) = z$ iff $\frac{x}{y} = z$

iff $x = y \cdot z$

iff $\quad X - Y \cdot Z = 0$

$\Rightarrow$ use a function $i(X,Y,Z) = X - Y \cdot Z$
and search for the smallest Z where
$i(X,Y,Z) = 0$.

$div(X,Y) = Z$ iff $i(X,Y,Z) = 0$ and
forall $0 \leq Z' < Z$, $i(X,Y,Z')$ is defined
and $i(X,Y,Z') > 0$

where $i(X,Y,Z)$ computes $X - Y \cdot Z$. This function i
is primitive recursive:

$$i(X,Y,Z) = minus(proj_{3,1}(X,Y,Z), j(X,Y,Z))$$
$$j(X,Y,Z) = times(proj_{3,2}(X,Y,Z), proj_{3,3}(X,Y,Z))$$

How can a LP "compute" an arithmetic function?

- A LP only "evaluates" predicate symbols, not function symbols.
  Solution: to compute a function $f : \mathbb{N}^n \to \mathbb{N}$,
  use a predicate symbol $\underline{f}$ of arity $n+1$
  where $\underline{f}(K_1, \dots, K_n, K)$ is true iff
  $f(K_1, \dots, K_n) = K$.
- LPs operate on terms, not on natural numbers.

Solution: represent natural numbers by terms using $0 \in \Sigma_0$ and $s \in \Sigma_1$.

Then the term 0 represents the number 0,

$s(0)$ — 1

$s(s(0))$ — 2

$\vdots$

Def 423 (Computing arithmetic functions with logic programs)

- Every $k \in \mathbb{N}$ is represented by the term $\underline{k} \in \mathcal{T}(\Sigma, \mathcal{V})$ where $\underline{k} = s^k(0)$, where $0 \in \Sigma_0$, $s \in \Sigma_1$
- A LP $\mathcal{S}$ over (Σ, Δ) computes an arithmetic fct $f : \mathbb{N}^n \to \mathbb{N}$ iff there is a pred. symbol $\underline{f} \in \Delta_{n+1}$ such that

$$f(k_1, \dots, k_n) = k \quad \text{iff} \quad \mathcal{S} \models \underline{f}(\underline{k_1}, \dots, \underline{k_n}, \underline{k}).$$

Reason: To compute $f(k_1, \dots, k_n)$, one can then ask the query $?- \underline{f}(\underline{k_1}, \dots, \underline{k_n}, X)$.

Ex. 424 The example functions in Ex. 422 can all be computed by a LP:

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

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

$\vdots$

Thm 425 (Universality of LP)

Every μ-recursive fct. can be computed by a LP.

Proof: Induction according to the construction principle for μ-recursive fcts.

1. $\underline{null}_n(X_1, \dots, X_n, 0).$
2. $\underline{succ}(X, s(X)).$
3. $\underline{proj}_{n,i}(X_1, \dots, X_n, X_i).$
4. By ind. hypothesis, there are predicates $\underline{f}, \underline{f_1}, \dots, \underline{f_m}$ that compute $f, f_1, \dots, f_m$.

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

5. By ind. hyp., there are predicates $\underline{f}$ and $\underline{g}$:

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

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

$\qquad\qquad\qquad\qquad\qquad \underline{g}(X_1,\dots,X_n,X,Y,Z).$

6\. By ind. hyp, there is a pred $\underline{f}$.

We introduce an additional predicate $\underline{f}'$ such that

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

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

$f(X_1,\dots,X_n,X) > 0$ for all X with $Y \leq X < Z$

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

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

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

$$\qquad\qquad\qquad \underline{f}'(X_1,\dots,X_n,s(Y),Z).$$

$\boxed{\text{\_}}$

Ex 426 The construction principle from the proof of Thm 425 could be directly used to convert μ-recursive functions to LPs.

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

```
plus(X, s(Y), U) :- plus(X, Y, Z), f(X, Y, Z, U).
f(X, Y, Z, V) :- proj3,3(X, Y, Z, U), succ(U, V).
succ(X, s(X)).
proj1,1(X, X).
proj3,3(X, Y, Z, Z).
```