4.2 Universality of Logic Programming

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: \( \mu \)-recursive functions

\[ \Rightarrow \text{Church's thesis:} \]

No prog. language can compute more functions than those expressible by Turing machines, \( \lambda \)-calculus, \( \mu \)-recursion.

Thus: to prove that LP is Turing-complete, show that for every \( \mu \)-recursive function, there is a LP computing it.
All algebraic data structures (lists, trees, ...) can be encoded as natural numbers if we regard algorithms on numbers.

\textbf{Def 4.2.1. (\(\mu\)-recursive functions)}

The set of \(\mu\)-recursive functions is the smallest set of functions such that:

1. For every \(n \in \mathbb{N}\), the function \(\text{null}_n : \mathbb{N}^n \rightarrow \mathbb{N}\) with \(\text{null}_n(k_1, ..., k_n) = 0\) is \(\mu\)-recursive.

2. The successor function \(\text{succ} : \mathbb{N} \rightarrow \mathbb{N}\) with \(\text{succ}(k) = k + 1\) is \(\mu\)-recursive.

3. For every \(n \geq 1\) and every \(1 \leq i \leq n\), the projection function \(\text{proj}_{n, i} : \mathbb{N}^n \rightarrow \mathbb{N}\) with \(\text{proj}_{n, i}(k_1, ..., k_n) = k_i\) is \(\mu\)-recursive.

4. \(\mu\)-recursive functions are closed under composition: For all \(n \geq 1\) and \(n \geq 0\) we have:

if \(f : \mathbb{N}^m \rightarrow \mathbb{N}\) and \(f_1, ..., f_m : \mathbb{N}^n \rightarrow \mathbb{N}\) are \(\mu\)-recursive, then the following function \(g : \mathbb{N}^n \rightarrow \mathbb{N}\) is also \(\mu\)-recursive:

\[ g(k_1, ..., k_n) = f\left(f_1(k_1, ..., k_n), ..., f_m(k_1, ..., k_n)\right) \]
5. The $\mu$-recursive functions are closed under primitive recursion. For all $n \geq 0$ we have:
if $f : \mathbb{N}^n \rightarrow \mathbb{N}$ and $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ are $\mu$-recursive, then the following function $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ is also $\mu$-recursive:

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

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

Functions that can be expressed with principles 1–5 are called primitive recursive.

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. $\mu$-recursive functions are closed under unbounded minimization: For all $n \geq 0$ we have:
if $f : \mathbb{N}^n \rightarrow \mathbb{N}$ is $\mu$-recursive, then the following function $g : \mathbb{N}^n \rightarrow \mathbb{N}$ is also $\mu$-recursive:
\[ g(k_1, \ldots, k_n) = k \text{ iff } f(k_1, k_2, k) = 0, \]
and for all \( 0 \leq k' < k, \)
\[ f(k_1, \ldots, k_n, k') \text{ is defined and } f(k_1, \ldots, k_n, k') > 0. \]

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

Now we will show that every \( \mu \)-recursive function can be computed by a LP.

**Ex 4.22** Consider some well-known computable facts on \( \mathbb{N} \) and show that they are \( \mu \)-recursive.

- **plus:** \( \mathbb{N}^2 \rightarrow \mathbb{N} \) is \( \mu \)-recursive, even primitive recursive
  \[
  \begin{align*}
  \text{plus}(x, 0) &= \text{proj}_{1,1}(x) \\
  \text{plus}(x, y + 1) &= f(x, y, \text{plus}(x, y)) + 1 \\
  f(x, y, z) &= \text{succ}(\text{proj}_{3,3}(x, y, z))
  \end{align*}
  \]
- **times:** \( \mathbb{N}^2 \rightarrow \mathbb{N} \) is also primitive recursive
  \[
  \begin{align*}
  \text{times}(x, 0) &= \text{null}_1(x) \\
  \text{times}(x, y + 1) &= g(x, y, \text{times}(x, y)) + x \\
  g(x, y, z) &= \text{times}(x, y, z)
  \end{align*}
  \]
\( g(x, y, z) = \text{plus}(\text{proj}_{3,1}(x, y, z), \text{proj}_{3,3}(x, y, z)) \)

- The predecessor function is also primitive recursive: \( p : \mathbb{N} \to \mathbb{N} \) with \( p(0) = 0 \), \( p(x+1) = x \)
  \[
  p(0) = \text{null}_0 \\
  p(x+1) = \text{proj}_{2,1}(x, p(x))
  \]

- The function \( \text{minus} : \mathbb{N}^2 \to \mathbb{N} \) is also primitive recursive, where \( \text{minus}(x, y) = 0 \) if \( x \leq y \) and \( \text{minus}(x, y) = x - y \) otherwise.
  \[
  \text{minus}(x, 0) = \text{proj}_{1,1}(x) \\
  \text{minus}(x, y+1) = \underbrace{g(x, y, \text{minus}(x, y))}_{p(\text{minus}(x, y))}
  \]

- \( h(x, y, z) = p(\text{proj}_{3,3}(x, y, z)) \)

- \( \text{div} : \mathbb{N}^2 \to \mathbb{N} \) is also \( \mu \)-recursive, where
  \[
  \text{div}(x, y) = \left\lfloor \frac{x}{y} \right\rfloor \text{ if } y \neq 0 \\
  \text{div}(0, 0) = 0 \\
  \text{div}(x, 0) \text{ is undefined if } x \neq 0
  \]

  Idea: \( \text{div}(x, y) = z \) if \( \frac{x}{y} = z \) if \( x = y \cdot z \)
\[ \text{iff } x - y \cdot z = 0 \]

\[ \Rightarrow \text{ use a function } i(x, y, z) = x - y \cdot z \]

and search for the smallest \( z \) where
\[ i(x, y, z) = 0. \]

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

where \( i(x, y, z) \) computes \( x - y \cdot z \). This function \( i \)

is primitive recursive:
\[ i(x, y, z) = \text{minus} \left( \text{proj}_{3,1}(x, y, z), j(x, y, z) \right) \]
\[ j(x, y, z) = \text{times} \left( \text{proj}_{3,2}(x, y, z), \text{proj}_{3,3}(x, y, z) \right) \]

---

How can a LP "compute" an arithmetic function?

- A LP only "evaluates" predicate symbols, not function symbols.

Solution: to compute a function \( f : \mathbb{N}^n \rightarrow \mathbb{N} \),

use a predicate symbol \( \overline{f} \) of arity \( n+1 \)

where \( \overline{f}(\overline{x}_1, \ldots, \overline{x}_n, \overline{y}) \) is true \text{ iff }
\[ f(x_1, \ldots, x_n) = y. \]

- 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 $0$,

\[
\begin{align*}
S(0) & \quad 1 \\
S(S(0)) & \quad 2 \\
& \quad \vdots
\end{align*}
\]

Def 4.23 (Computing arithmetic functions with logic programs)

- Every $k \in \mathbb{N}$ is represented by the term $\overline{k} \in \mathcal{F}(\Sigma, \Delta)$ where $\overline{k} = S^k(0)$, where $0 \in \Sigma_0$, $s \in \Sigma_1$.
- A LP $P$ over $(\Sigma, \Delta)$ computes an arithmetic function $\overline{f} : \mathbb{N}^n \rightarrow \mathbb{N}$ iff there is a pred. symbol $\overline{f} \in \Delta_{n+1}$ such that

\[
f(k_1, \ldots, k_n) = k \quad \text{iff} \quad S \vdash \overline{f}(\overline{k_1}, \ldots, \overline{k_n}, \overline{k}).
\]

Reason: To compute $f(k_1, \ldots, k_n)$, one can then ask the query \[\neg \overline{f}(\overline{k_1}, \ldots, \overline{k_n}, X)\].

Ex. 4.24 The example functions in Ex. 4.22 can all be computed by a LP.
\textbf{Thm 425 (Universality of \textit{CP})}

Every \textit{\mu}-recursive \textit{fct.} can be computed by a \textit{CP}.

\textbf{Proof:} Induction according to the construction principle for \textit{\mu}-recursive \textit{fcts}.

1. \underline{\text{null}}_n (X_1, \ldots, X_n, O).

2. \underline{\text{succ}} (X, s(X)).

3. \underline{\text{proj}}_n,i (X_1, \ldots, X_n, X_i).

4. By ind. hypothesis, there are predicates \underline{f}, \underline{f}_1, \ldots, \underline{f}_m
   that compute \underline{f}, \underline{f}_1, \ldots, \underline{f}_m.

   \[ g (X_1, \ldots, X_n, 2) :\!
   \begin{aligned}
   &\underline{f}_1 (X_1, \ldots, X_n, Y_1), \ldots, \underline{f}_m (X_1, \ldots, X_n, Y_m), \\
   &\underline{f} (Y_1, \ldots, Y_m, 2).
   \end{aligned} \]

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

   \[ h (X_1, \ldots, X_n, 0, 2) :\!
   \begin{aligned}
   &\underline{f} (X_1, \ldots, X_n, 2).
   \end{aligned} \]
\[ h(X_1, \ldots, X_n, s(X), z) \equiv g(X_1, \ldots, X_n, X, Y), \]
\[ g(X_1, \ldots, X_n, X, Y, Z). \]

6. By induction hypothesis, there is a pred \( f \).

We introduce an additional predicate \( f' \) such that
\[ f'(X_1, \ldots, X_n, Y, Z) \text{ is true iff} \]
\[ f(X_1, \ldots, X_n, Z) = 0 \text{ and} \]
\[ f(X_1, \ldots, X_n, X) > 0 \text{ for all } X \text{ with } Y \leq X \leq Z. \]

\[ g(X_1, \ldots, X_n, Z) \equiv f'(X_1, \ldots, X_n, 0, Z). \]
\[ f'(X_1, \ldots, X_n, Y, Y) \equiv f(X_1, \ldots, X_n, Y, 0). \]
\[ f'(X_1, \ldots, X_n, Y, Z) \equiv f'(X_1, \ldots, X_n, Y, s(U)). \]
\[ f'(X_1, \ldots, X_n, s(Y), Z). \]

\textbf{Ex 426} The construction principle from the proof of Thm 425 could be directly used to convert \( \mu \)-recursive functions to LPS.

\[ \text{plus}(X, 0, U) \equiv \text{proj}_1(\text{plus}(X, U)). \]
\textit{plus} (X, s(Y), U) : - \textit{plus} (X, Y, Z), f (X, Y, Z, U).
\textit{succ} (X, s(X)).
\textit{proj}_{1,1} (X, X).
\textit{proj}_{3,3} (X, Y, Z, U).