



The Curry-Howard Correspondance

WESLEY JENKINS

Proofs

- ▶ Proofs are common to all of us.
- ▶ What is a proof?
 - ▶ A “thing” that somehow shows that a conclusion is true.
- ▶ Can create a set of proofs that prove some fact:
$$P(X) = (\text{Set of proofs of } X)$$
- ▶ Not a set of all proofs, just a set of all proofs for X specifically.

Proofs and Set Theory

Proof ideas need to be rewritten into set theoretic notation

- ▶ X is true if and only if $P(X)$ is inhabited (has at least one element).
- ▶ $P(X \text{ and } Y) \cong P(X) \times P(Y)$
- ▶ $P(X \text{ or } Y) \cong P(X) \cup P(Y)$
- ▶ $P(X \text{ implies } Y) \cong P(X) \rightarrow P(Y)$
 - ▶ Given proof of X , this function will return proof of Y .
 - ▶ The function must be well-defined.
- ▶ $P(\forall (n \in N) X(n)) \cong (n : N) \rightarrow P(X(n))$
 - ▶ Confusing notation
 - ▶ Dependent product type
- ▶ $P(\exists (n \in N) X(n)) \cong (n : N) \times P(X(n))$
 - ▶ Dependent sum type
 - ▶ Confusing because it uses a product

Examples

- ▶ Proof that if 2 is even, then 4 is also even.
 - ▶ $\phi : P(2 \text{ is even}) \rightarrow P(4 \text{ is even})$
 - ▶ Given proof that 2 is even, it will return proof that 4 is even.
- ▶ Proof that all integers are real numbers.
 - ▶ $\phi : (n : \mathbb{Z}) \rightarrow P(n \text{ is real})$
 - ▶ Given a value like 3, $\phi(3) : P(3 \text{ is real})$
 - ▶ Codomain is the union of all $P(n \text{ is real})$.
 - ▶ Only true if such a function can be constructed!
- ▶ Proof that there exists an integer less than 5
 - ▶ $\phi : (n : \mathbb{Z}) \times P(n < 5)$
 - ▶ Second set depends on a value from the first.

Programming

- ▶ Everyone is also familiar with programming and programs.
- ▶ What is a program?
 - ▶ A sequence of steps to perform some computation.
- ▶ There are many abstract models of computation
 - ▶ Turing machines, lambda calculus, etc.
- ▶ And many programming languages, which are considered “turing-complete,”
 - ▶ C++, Python, Java, etc.

Language Typing

- ▶ Values in most programming languages have “types.”
 - ▶ Equivalent to sets of objects.
- ▶ Statically-typed languages make requirements on the type of a variable.
 - ▶ Equipped with a type checker, which ensures the types are correct.
 - ▶ Cannot use a value where it is unexpected, e.g. real number in place of integer

Python: Dynamic typing

```
def addValues(x, y):  
    z = x + y  
    return z
```

C++: Static typing

```
int addValues(int x, int y) {  
    int z = x + y;  
    return z;  
}
```

1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 26

- $$P(2 \text{ is even}) \rightarrow P(4 \text{ is even})$$
- ```
proof_4_is_even proveEven (proof_2_is_even proof) {
 ???
}
```

- ▶ But will it work?
- ▶ No. Why?

# Type Systems

- ▶ C++ isn't sufficient
  - ▶ Not a complex enough type system.
  - ▶ Has a lot of “outs” like exceptions and the “exit” function which ends the program.
  - ▶ Functions aren't pure (So, not even real mathematical functions).
  - ▶ Not actually strictly typed! Allows casting!
- ▶ Need a more suitable language.



# Agda

- ▶ Agda is designed for this purpose.
  - ▶ Much more complicated type system.
    - ▶ Hindley-Milner type system with support for dependent types.
  - ▶ All functions are total and pure, meaning there are no escape routes like in C++.
  - ▶ Can't create values of any type out of thin air.
- ▶ Functional language closely related to Haskell.

```
proveEven : proof_2_even -> proof_4_even
proveEven proof = ???
```

# Automated Proof Checking

- ▶ Agda programs themselves act as proofs.
- ▶ The Agda compiler checks the program.
  - ▶ If the Agda compiler accepts the program, then it must be a valid proof.
  - ▶ (Unless there is a bug in the Agda compiler)

```
data Nat : Set where
 zero : Nat
 suc : Nat -> Nat
```

```
data _==_ : Nat -> Nat -> Set where
 refl : (n : Nat) -> n == n
```

```
eqTrans : {n m l : Nat} -> n == m -> m == l -> n == l
eqTrans (refl x) (refl x) = refl x
```

# Proven Programs

- ▶ Agda programs, after being verified, can be exported to Haskell.
- ▶ These programs have guaranteed behavior, because their behavior has been proven already.
  - ▶ Can be used in critical applications where the validity of code is crucial: nuclear reactors, space probes, etc.
  - ▶ Assuming there are no hardware faults, the program cannot contain bugs.
- ▶ Can provide certain security guarantees at compile time rather than runtime.
  - ▶ Buffer overrun checks, array out of bounds checks, etc.
- ▶ So why aren't all programs proven?
  - ▶ Because it's very difficult and time-consuming to write a proven program.

# Curry-Howard Correspondence

- ▶ So, formal logic can be embedded inside of programming.
- ▶ And type checking can then be used to prove such logic is valid.
- ▶ The Curry-Howard correspondence states that proof systems and systems of computation are isomorphic to one another.
  - ▶ They describe the same set of rules in a different way.
- ▶ And so, this ties together computation and logic.
  - ▶ Every valid logical argument can be turned into a runnable program.
  - ▶ And every runnable program can be turned into a logical argument.

# And its Consequences

- ▶ There are a lot of consequences of this correspondence.
  - ▶ Unifies certain parts of mathematics and foundational computer science.
  - ▶ Gives new understandings of both proofs and programs.
  - ▶ And creates new methods of creating proofs as programs.
  - ▶ Potential philosophical interpretations on the nature of logic and computation.

Questions?