Programowanie funkcyjne - wprowadzenie

Damian Kurpiewski

Programowanie imperatywne

Piszemy instrukcje, które zmieniają wewnętrzny stan programu

Skupiamy się na opisaniu tego, **jak działa program**

Opiera się na modelu **Maszyny Turinga**

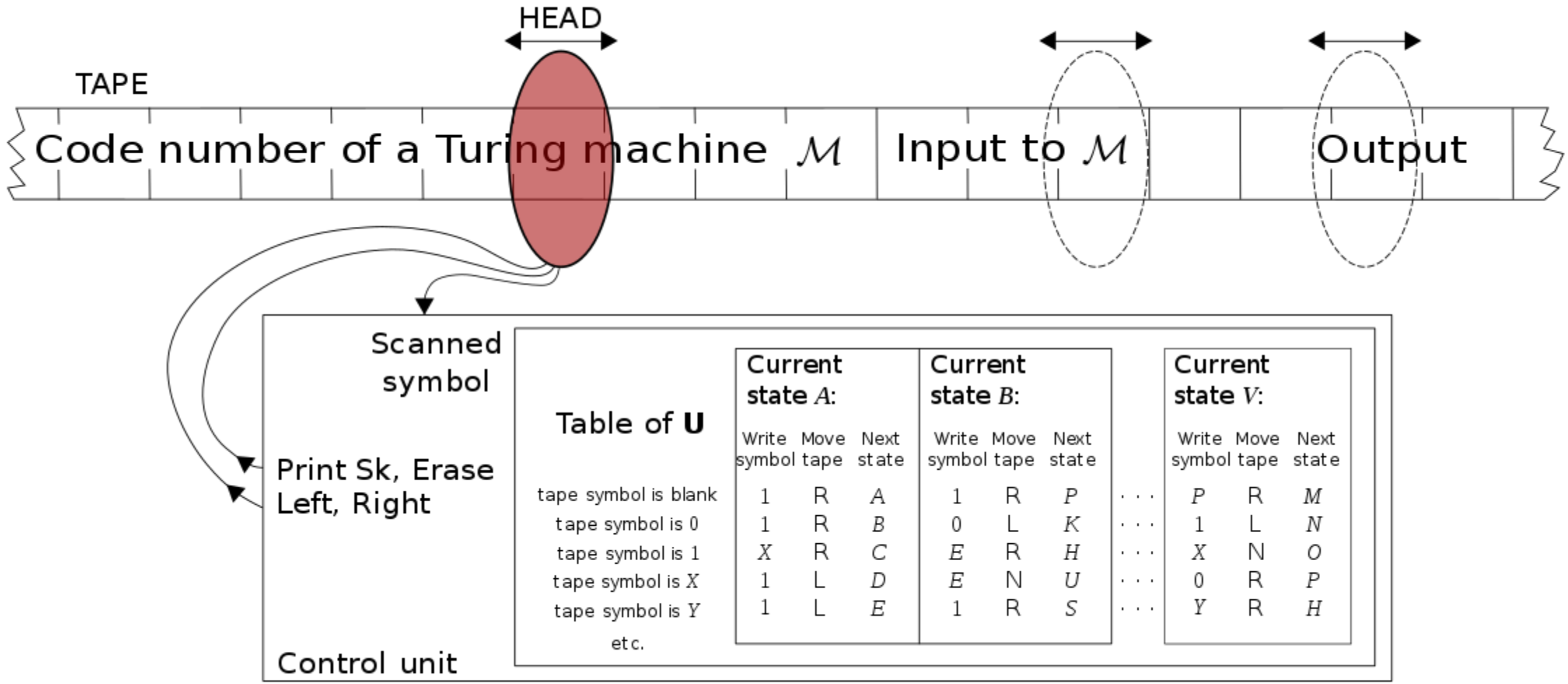
Maszyna Turinga

Stan wewnętrzny

Nieskończona taśma pamięci

Głowica poruszająca się w lewo/prawo i odczytująca/zapisująca dane

Tabela instrukcji opisująca działanie maszyny



Programowanie funkcyjne

Skupiamy się na opisaniu tego, **co robi program**, a nie jak działa

Piszemy kod **deklaratywny**, nie imperatywny

Opiera się na **Rachunku Lambda**

Rachunek Lambda

Opiera się na matematycznej idei **Funkcji Czystych**:

Zwraca wynik opierający się wyłącznie na parametrach funkcji

Nie produkuje efektów ubocznych

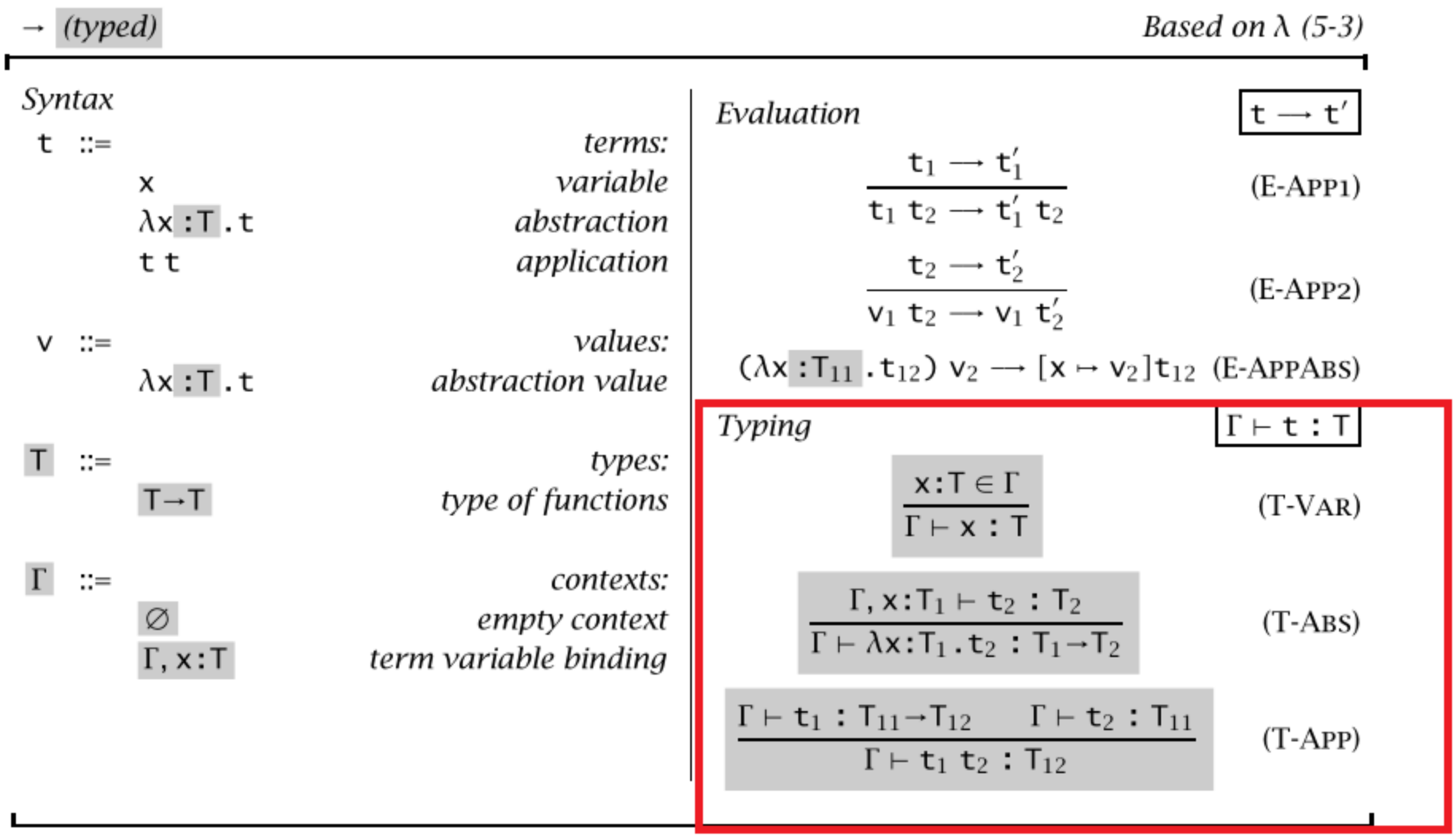


Figure 9-1: Pure simply typed lambda-calculus ($\lambda_{\rightarrow}$)

Zalety programowania funkcyjnego

Wynik zawsze zależny jedynie od wejścia

Prostszy do czytania i zrozumienia kod

Uproszczenie obliczeń równoległych – unikamy zakleszczeń

Uwaga: niektóre języki imperatywne (np. Python, Swift) zawierają elementy programowania funkcyjnego

Niezmienność

Nie zmieniamy nic, tylko obliczamy

Nie zmieniamy nic „na zewnątrz”

Dopuszczamy lokalną zmienność

Efekty uboczne – zmiana stanu, przykłady

Zmiana wartości zmiennej

Zapisanie danych na dysku

Włączanie/wyłączanie przycisku w interfejsie użytkownika

Efekt uboczny nie musi być ukryty albo niezamierzony

Funkcje czyste

Zależne wyłącznie od parametrów wejściowych

Każde wywołanie z takimi samymi wartościami parametrów daje ten sam wynik

Łatwe do testowania w testach jednostkowych

Łatwe do zrównoleglania

Przejrzystość referencyjna

- Gdy możemy zastąpić funkcję jej wynikiem
- Np. func add(a: Int, b: Int) -> Int {return a + b}
- add(a, b) = a + b
- add(10, 20) = 10 + 20

Funkcje wyższego rzędu

W językach funkcyjnych traktujemy funkcje jak inne obiekty

Funkcje mogą przyjmować inne funkcje jako parametry

Funkcje mogą zwracać funkcje jako wynik

Funkcje, które to robią, nazywamy funkcjami wyższego rzędu

Podstawowe funkcje wyższego rzędu: filter, map, reduce

filter

- Działa na zbiorze danych
- Przyjmuje inną funkcję jako parametr
- Funkcja ta przyjmuje jedną wartość ze zbioru jako wejście, sprawdza czy ta wartość pasuje i zwraca Prawda/Fałsz
- Filter wykonuje funkcję z parametru na każdym elemencie danego zbioru
- Wynikiem jest zbiór elementów, dla których zadana funkcja zwraca wartość true

map

- Działa na zbiorze danych
- Przyjmuje inną funkcję jako parametr
- Funkcja ta przyjmuje jedną wartość ze zbioru jako wejście, modyfikuje ją i zwraca jako wynik
- Map wykonuje funkcję z parametru na każdym elemencie danego zbioru
- Wynikiem jest zbiór elementów ze zmodyfikowanymi wartościami

reduce

- Działa na zbiorze danych
- Przyjmuje wartość początkową oraz inną funkcję jako parametr
- Funkcja ta przyjmuje dwa parametry: wartość z poprzedniego obliczenia i kolejną wartość ze zbioru
- Reduce wykonuje zadaną funkcję na każdym kolejnym elemencie zbioru, zbierając dotychczasowe wyniki
- Wynikiem jest jedna wartość

Rekurencja

Funkcja wywołująca samą siebie

Aby uniknąć nieskończonych wywołań, potrzebujemy warunku stopu