Opis implementacji języka Matal

Aleksander Matusiak

# Cechy języka

* **Typy**: bool, int, void (do możliwości stworzenia procedur)
* **Arytmetyka i przypisania**: +, -, \*, /, ( ), zmienna++, zmienna--, =, +=, -=, \*=, /=, porównywanie wartości
* **Pętle:** while i for
* **Instrukcje warunkowe:** if, if … else (niezbędne nawiasy klamrowe)
* **Funkcje**: rekurencja, dowolne zagnieżdżanie (możliwości deklaracji funkcji wewnątrz funkcji), statyczne wiązanie identyfikatorów, przekazywanie parametrów przez wartość, możliwość zwrócenia typu void (de facto – procedury)
* **Przesłanianie identyfikatorów ze statycznym ich wiązaniem**
* **Statyczne typowanie**
* **Jawnie obsłużone błędy,** np.: dzielenia przez 0, wyjście poza rozmiar tablicy
* **Wypisanie wartości:** instrukcja print – może wypisywać pojedyncze zmienne, dodaje znak nowej linii na końcu
* **Tablice:** indeksowane int, każda tablica przed użyciem musi mieć przydzielony rozmiar przy pomocy polecenia *init*
* **Mapy:** indeksowane dowolnymi porównywalnymi wartościami
* **Struktury:** możliwość deklarowania pól z nazwami, a później odczytywania i zapisywania ich

# Opis rozwiązania

Rozwiązanie składa się z dwóch modułów, które zapewniają większość logiki interpretera. Są to:

* moduł Interpreter, który umożliwia wykonywanie instrukcji, ewaluowanie wyrażeń i deklaracji.
* moduł Checker, który umożliwia dokonanie statycznej kontroli typów, m.in.:
  + czy wartości zmiennych zgadzają się z ich zadeklarowanymi typami
  + czy typy wewnątrz wyrażeń się zgadzają
  + czy zmienne i funkcje są widoczne w danym miejscu programu
  + czy wywołujemy funkcję z odpowiedną liczbą argumentów i mają one odpowiednie typy
  + czy funkcje zwracają odpowiedni typ
  + czy pola struktur istnieją i mają odpowiednie typy

## Kompilacja

Rozwiązanie zostało napisane tak, żeby kompilowało się przy użyci ghc w wersjach 7.10.\*. Na maszynie students konieczne jest zatem wykonanie polecenia *make* w odpowiednim środowisku z właściwą wersją ghc.

## Zastosowane konstrukcje haskelowe

Zasadniczy kod programu (interpreter) opiera się na następujących typach:

*type Result = ExceptT String IO*

*type Interpreter a = StateT Store (ReaderT Env Result) a*

*Result* jest typem, który zwraca funkcja ewaluacyjna dla całego programu – może to być informacja o błędzie i/lub wypisanie pewnych informacji na wyjście. Z kolei przy denotacji instrukcji, wyrażeń i deklaracji posługujemy się jeszcze monadami stanu i środowiska, co umożliwia eleganckie zaimplementowanie wymaganych funkcjonalności.

## Uwagi o składni

Składnia nakłada pewne ograniczenia, związane z decyzjami projektowymi poczynionymi w rozwiązaniu:

* polecenie *return* musi być zawsze umieszczone na końcu każdej funkcji (też procedury),
* deklaracje zmiennych muszą znajdować się na początku ciała funkcji/bloku,
* wewnętrzne funkcje muszą być zdefiniowane w funkcji zaraz po deklaracji zmiennych lokalnych.

## Obsługa L-wartości

Rozwiązanie dostarcza podstawową obsługę L-wartości (co jest sprawdzane podczas statycznej analizy typów). Po lewej stronie przypisania mogą być:

* zmienne,
* pola struktur,
* elementy tablic,
* elementy map,

przy czym nie można łączyć powyższych wyrażeń, tzn. nie można np. przypisywać na pole struktury w tablicy struktur. Powyższe ograniczenia związane z tablicami dotyczą również polecenia *init*, tzn. można inicjować tylko jeden „poziom” tablicy na raz.