Projekt Funkcyjny

Andrzej Tkaczyk, 309181

Czerwiec 2020

Podsumowanie ogólne

Realizacja projektu trwała od początku marca do końca maja, czyli 3 miesiące. Zostały stworzone prawie wszystkie założone elementy wersji podstawowej. Wyjątkiem są wskaźniki, które miały pomóc w realizacji structów, ale okazały się być zbędne. Punkt trzeci specyfikacji odnosił się do elementów dodatkowych, doimplementowanych po zrobienu podstawy i również został częściowo wykonany.

W następnej części raportu opiszę realizację poszczególnych elementów specyfikacji.

Składnia abstrakcyjna

Stworzona z algebraicznych typów danych OCaml, znajduje się w pliku ast/Ast.ml.
Wyróżniają się trzy główne typy:

- value – wartości przechowywane w środowisku np. typ liczbowy lub null.
- aexpr – wyrażenia arytmetyczne, np. dodawanie, logiczne, np. równość, oraz takie, które po prostu zwracają jakąś wartość, np. wywołanie procedury.
- pexpr – wyrażenia w składni języka, np. przypisanie wartości do zmiennej lub deklaracja tablicy, ale również kombinacja dwóch wyrażeń w składni języka.

Dodatkowym typem jest tree, służy do reprezentacji tablic w języku.

Parser

Generowany przez generator parserów w języku OCaml - menhir.

Definicja gramatyki znajduje się w pliku syntax/Grammar.mly.
Definicja słów kluczowych znajduje się w pliku syntax/Lexer.mll.

Opisanie sposobu tworzenia tych plików byłoby w pewnym sensie redundantym przepisaniem specyfikacji menhir-a, więc nie zrobię tego, jedynie odeślę do specyfikacji i kodu.

Pretty printer

Trywialne odwrócenie procesu parsowania. Określony typ danych oznacza określony napis, wystarczy generować tekst dokładnie tak, jak w pliku syntax/PrettyPrinter.ml.

Interpreter

Najważniejsza część projektu. Rozróżnione jest interpretowanie wyrażeń w składni języka (typu pexpr) i ewaluowanie wyrażeń zwracających typ value (aexpr). Przyjmuje jako argumenty wyrażenie oraz środowisko i zwraca zmodyfikowane środowisko (w przypadku pexpr) lub wartość wyrażenia (w przypadku aexpr).

Dokładne działanie najlepiej opisuje sam kod w pliku interpreter/Interpreter.ml

Środowisko

Jest na tyle oczywistym elementem, że nie zostało zapisane w specyfikacji.
Reprezentowane jest przez listę par klucz - wartość, gdzie klucz jest napisem, np. nazwą zmiennej, a wartość jest dowolnym tworem typu value. Udostępnia pewnego rodzaju interfejs pozwalający m. in. dodawać i usuwać elementy, lub zmieniać wartość zmiennej na inną.

Kod znajduje się w pliku interpreter/Environment.ml

System typów

Nie istnieje jako osobny twór analizujący program przed interpretacją, lecz jest "wszyty" w interpreter i żyje z nim jakby w symbiozie. Sprawdza poprawność typów w trakcie ewaluacji, np. w trakcie dodawania po ewaluacji lewego i prawego argumentu sprawdza czy są to dwie liczby, lub dwa napisy i ewentualnie zgłasza błąd.

Elementy języka z pierwszego punktu specyfikacji

- zmienne liczbowe – Wyewoluowały do zmiennych przechowujących różne typy danych. Każda zmienna jest przechowywana w środowisku i musi mieć nazwę złożoną z liter alfabetu angielskiego (takie ograniczenie pomaga w parsowaniu). Można dodać zmienną do środowiska, przykryć zmienną, inną zmienną o tej samej nazwie, zmienić wartość zmiennej. Zmienne zadeklarowane lokalnie, są widoczne tylko w bloku w którym zostały zadeklarowane, dlatego po interpretacji programu w środowisku znajdują się tylko zmienne zadeklarowane globalnie. Obliczenie wartości zmiennej, gdy jest to zmienna liczbowa, boolowska, lub napis, jest szczególnym przypadkiem wyrażenia arytmetycznego.
- wyrażenia z arytmetyką – Dostępne są podstawowe operacje arytmetyczne i porównania. Argumentami mogą być stałe, zmienne, wartości wyliczane procedurami. Wyniki obliczenia mogą być zapisane w zmiennej, lub użyte jako warunek, np. w pętli.
- konstrukcje IF i WHILE – Polegają na interpretacji odpowiedniej części programu, lub zwrócenie niezmienionego środowiska, w zależności od wartości warunku.
- procedury – Mogą zostać zadeklarowane, a następnie wykorzystywane do obliczenia jakiejś wartości. Wymaga podania nazwy procedury, listy nazw argumentów, bloku kodu który ma wykonać oraz wyrażenia którego wartość zwróci po wykonaniu bloku kodu. Przechowywana w środowisku jako zmienna z podanymi informacjami w deklaracji oraz domknięciem. Wywołanie wymaga podania wartości argumentów (możliwe również w postaci wyrażeń), które zostaną dodane do domknięcia.
- wskaźniki – Odrzucone.
- coś jak ubogi struct w C – Deklaracja structa wymaga podania nazwy oraz listy nazw pól. Szablon tak podanego structa jest przechowywany w środowisku jako zmienna z listą nazw pól. Utworzenie nowego structa wymaga podania początkowych wartości pól (możliwe również w postaci wyrażeń). Structy są przechowywane w środowisku jako małe środowiska, do których nie można nic dodać ani odjąć, są tam jedynie zmienne o nazwach pól structa i można jedynie zmieniać ich wartości. Takie rozwiązanie może ułatwić późniejsze rozwinięcie structów do klas i obiektowości, ponieważ np. dziedziczenie wymaga jedynie wywołania envlookup na nadklasie, jeśli nie znajdziemy w naszym środowisku, lub tworzenie pól publicznych i prywatnych wymaga zmiany reprezentacji z jednego małego środowiska, do dwóch, jednego widocznego z zewnątrz, drugiego nie.

Elementy języka z drugiego punktu specyfikacji

- rekurencja w procedurach – domknięcie w procedurze jest przechwywane jako środowisko z momentu deklaracji, więc wystarczyło zmienić to pole na modyfikowalne i po dodaniu procedury do środowiska zrobić cykl.
- tablice i napisy – tablice są reprezentowane jako słowniki z kluczami i wartościami, w których klucze są ustalone, od 0 do rozmiar-1 i można tylko odczytywać i modyfikować wartości o odpowiednim kluczu. słowniki są reprezentowane jako kopce binarne. Deklaracja tablicy wymaga podania nazwy, rozmiaru i początkowej wartości, która pojawi się w każdej komórce. Dodanie modyfikowalnych napisów jako osobny element języka uznałem za zbędny, po tym, jak zostały dodane napisy jako dobra wartość zmiennej, można po prostu utworzyć tablicę i wypełnić ją takimi napisami żeby mieć modyfikowalne napisy.
- system typów – już opisany, jako większa część projektu.

Elementy języka z trzeciego punktu specyfikacji

- rozbudowa structów do czegoś przypominającego klasy – odrzucone, chociaż jak zaznaczyłem w opisie samych structów, nie jest to ogromna zmiana, więc zapewne dorobię to w przyszłości dla sportu.
- obsługa wejścia i wyjścia – odziedziczone z OCaml-a. Oprócz składni do zwykłego czytania i pisania wymagało jeszcze dodania specjalnych funkcji do wypisania spacji i końca linii, co wynika z tego, że przydatne czasem jest wypisanie takich znaków, a nie występują jako poprawny element napisu w języku, co z kolei jest spowodowane ułatwieniem w parsowaniu.
- dodanie kolejnych konstrukcji do języka – czyszczenie pamięci jako przykład "funkcji wbudowanych". Jest to pomysł wynikający z tego, że nie ma typowych funkcji zwracających nic, jeśli chcemy zwrócić nic, to musimy przynajmniej zwrócić null, a co za tym idzie musimy wynik takiego obliczenie przypisać do jakiejś zmiennej, lub użyć w wyrażeniu. Zatem można utworzyć takie funkcje będące elementem języka wywoływane jak np. funkcje typu void w C i użytkownik języka nie będzie miał możliwości utworzyć takich fuunkcji, ani ich przykryć. Takim przykładem są właśnie funkcja czyszcząca całe środowisko i funkcja usuwająca konkretną zmienną ze środowiska, lub funkcje wypisujące spację i koniec linii.
  Drugim elementem dodanym na końcu jest pętla for. wymaga podania nowej zmiennej z początkową wartością (w zamyśle ma pełnić rolę iteratora), warunku wejścia w pętlę oraz polecenia wykonywanego po każdym obrocie pętli. Dodanie do składni podtypu LoopExpr jest zbędne, bo da się to samo wyrazić za pomocą WhileExpr, to również zostanie zmienione w niedalekiej przyszłości.