



مستندات فاز 1 پروژه PL

علی احمدوند 401110318
محمد داودآبادی فراهانی 401110331
سید مبین رضوی 401110267



1- نمای کلی :

در این پروژه ، یک مفسر برای زبانی مدنظر در فاز 0 طراحی شده است. مفسر در زبان Racket نوشته شده است و از پارسر ای که در فاز قبل طراحی شده استفاده می کند. فایل اصلی آن interpreter.rkt می باشد. ساختار طراحی شباهت زیادی به طراحی های کتاب EOPL دارد و برگرفته از آن می باشد. در اینجا از یک store-based environment برای نگهداری variable bindings استفاده می شود. داده ساختاری به نام Expval وجود دارد که همان expressed values می باشد و خروجی مورد انتظار توابع value-of می باشد و دارای انواعی مانند int-val ، string-val ، array-val ، proc-val و ... است. این پروژه شامل بخش های مختلف و feature های متعددی مانند error-handling ، type-checking ، lazy-evaluation ، short-circuit-design و ... است که در پایین به آنها می پردازیم. همچنین توجه کنید که تست های این مفسر در فایل interpreterTest.rkt نوشته شده است.

2- Environment

Environment در اینجا یک تناظر از نام متغیرها به reference هایشان می باشد که در واقع مکان ذخیره ی آنها در store را نشان میدهد. Store نیز یک بردار mutable با در واقع تناظری از مکان ها به value ها می باشد. Value ها در store با داده ساختار expval ذخیره شده اند. این پیاده سازی ویژگی هایی مانند Lexical scoping و variable shadowing را برای ما فراهم میکند.

3- Value-of

توابع اصلی جهت evaluate کردن برنامه ، توابع value-of می باشند. برای هر یک از node های موجود در درخت AST ، تابعی مخصوص آن نوشته شده است که به عنوان مثال می توان به توابع value-of- ، value-of-declaration ، value-of-program statement و ... اشاره کرد. در نهایت نیز برای محاسبه ی expression ها تابع value-of وجود دارد که بعد از حالت بندی روی نوع expression ، هریک را به شیوه ی درست evaluate میکند. قابل ذکر است که در این مفسر خاصیت های کار با string نیز در دل همین توابع پیاده سازی شده اند ، به عنوان مثال می توان یک داده ی string را به دید آرایه ای از char ها بررسی کرد و به char های آن در جایگاه های مختلف دسترسی داشته و حتی آنها را ادیت کرد.

4- Lazy Evaluation

با کمک داده ساختارهای thunk ، طراحی call-by-need در اینجا پیاده سازی شده است. هنگام پاس دادن آرگومان ها به توابع ، ساختار thunk از عبارت آنها و environment فعلی شان ساخته شده و رفرنس های جدیدی برای نگهداری شان ساخته شده و به تابع موردنظر پاس داده می شوند. اکنون اولین لحظه ای که ورودی های تابع مورد استفاده قرار بگیرند از فرمت thunk خارج شده و evaluate می شوند و حاصل آن در رفرنس موردنظر جایگزین می شوند. این اتفاق در تابع value-of روی case های var-exp و array-ref-exp اتفاق می افتد که با چک کردن اینکه آیا رفرنس مورد نظر شامل thunk می باشد یا خیر ، در صورت نیاز مقدار را پس از evaluate شدن در جایگاه متناظر ذخیره میکند. در زیر می توانید نمونه ای از کارکرد آن را

ببینید:

```
467
468 (run (do-parse lazy-eval-call))
469 (newline)
470
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
11
(int-val 1)
```

```

348 (define lazy-eval-call "
349 int loop (int n) {
350 |     return loop (n + 1);
351 | }
352
353 int f(int n) {
354 |     return 11;
355 | }
356
357 int main () {
358 |     print ("\n~x\" f(loop(2)));
359 |     return 1;
360 | }
361 ")

```

همچنین توجه کنید که این مفسر دارای ویژگی short-circuit می باشد به این معنی که در محاسبه ی عباراتی مانند $x * y$ یا $x || y$ ، اگر عبارت سمت چپ مساوی صفر در مثال اول یا True در مثال دوم محاسبه شود ، به محاسبه ی عبارت سمت راست نخواهیم پرداخت و حاصل کل برابر صفر یا True قرار داده می شود. در زیر نمونه ای از کارکرد آن را می بینید :

```

371 (define short-circuit "
372 int loop (int n) {
373 |     return loop (n + 1);
374 | }
375 int main () {
376 |     return False && (2 == loop(2));
377 | }
378 ")

```

```

473
474 (run (do-parse short-circuit))
475 (newline)
476

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

(bool-val #f)

Procedures -5

این مفسر به طور کامل از توابع عادی و بازگشتی پشتیبانی می کند. در call-exp از تابع value-of ، همانطور که بالاتر توضیح داده شد ، thunk هایی از آرگومان ها ساخته می شود و بعد از استخراج شدن داده ساختار proc (که شامل بدنه ی تابع ، نام آن ، environment آن ، لیست آرگومان هایش و لیست تایپ های آرگومان هایش می باشد) همگی آن را به تابع apply-procedure می دهد. در آنجا رفرنس های جدیدی از آرگومان ها (thunk ها در واقع) ساخته و به environment تابع اضافه کرده و value-of-statement را روی بدنه و environment جدید صدا می کند. توجه کنید که چک کردن درستی تایپ آرگومان های داده شده به شیوه ی خاصی انجام می شود که نیاز به evaluate شدن thunk ها موقع پاس دادن شان نباشد. در عوض ، اولین لحظه ای که این آرگومان ها در بدنه ی تابع مورد استفاده قرار گیرند و از thunk خارج شده و evaluate بشوند ، برابری تایپ حاصل شان با تایپ مورد انتظار تابع بررسی شده و در صورت عدم سازگاری ارور مناسب داده می شود. برای پیاده سازی این منظور ، به داده ساختار thunk ، ویژگی expected-type نیز اضافه شده است که موقع ساخته شدن آنها در یک function call ، برابر تایپ مورد توقع تابع قرار میگیرد.

در رابطه با توابع بازگشتی نیز توابع مورد نیاز برای چک کردن بازگشتی بودن و ... پیاده سازی شده اند و بقیه ی فرایند تفسیر شدنشان به طور کلی مشابه توابع عادی است.

Type Checking -6

Error Handling -7