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 درstore را نشان میدهد. Store نیز یک بردار mutable با در واقع تناظری از مکان ها به value ها می باشد. Valueها 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-program ، value-of-declaration ، value-of-statement و ... اشاره کرد. درنهایت 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)
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

11
(int-val 1)
```

```
(define lazy-eval-call "
int loop (int n) {
    return loop (n + 1);
}

int f(int n) {
    return 11;
}

int main () {
    print (\"~x\" f(loop(2)));
    return 1;
}
")
```

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 :

```
(define short-circuit "
int loop (int n) {
    return loop (n + 1);
}
int main () {
    return False && (2 == loop(2));
}
")
```

```
(run (do-parse short-circuit))
(newline)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
(bool-val #f)
```

5- Procedures

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 میدهد. 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.

6- Type Checking

در زبان ما تایپ‌های int، float، Boolean و string موجودند. هر متغیری که تعریف شود باید نوع داده‌ای که قرار است در خود ذخیره کند را تعیین کند.

مثالی از تعریف متغیر به صورت int x; می‌باشد. همچنین توابع نیز باید نوع خروجی خود را در تعریف خود مشخص کنند و مثالی از این تعریف به شکل int test(int a, int b){return a + b;} می‌باشد. چک کردن تایپ در دو زمان رخ می‌دهد: زمانیکه assignment ای رخ می‌دهد و زمانی که آرگومان‌هایی را به یک تابع می‌دهیم. حالت دوم در بخش قبل توضیح داده شد که با استفاده از thunk، تنها زمانی تطابق تایپ ورودی چک می‌شود که در بدنه‌ی تابع حداقل یکبار استفاده شده باشد که باعث evaluate شدن thunk و چک کردن تایپش می‌شود. حالت اول نیز در value-of، پس از بدست آوردن مقدار سمت راست تساوی، چک می‌شود که آیا تایپش مطابق تایپ پیش فرض سمت چپ هست یا خیر. سمت چپ ممکن است یک متغیر یا جایگاهی از آرایه باشد. برای متغیرهای عادی، تابعی داریم (get-type) که از اینکه متغیر چه کلاسی از expval است به ما تایپش را به صورت استرینگ بر می‌گرداند. اما برای آرایه‌ها که خود یک کلاس از expval می‌باشند (array-val)، نوع آنها را در فیلدی از خود array-val به نام type-name نگه داشته می‌شود. در نتیجه کافی‌ست تساوی تایپ سمت راست و چپ را با تابع equal? بدست آوریم و در صورت مطابقت نداشتن، TypeError بدهیم.

7- Error Handling

انواع مختلفی از ارورها در مفسر ما هندل شده است. این ارورها عبارتند از:

- ReferenceError: هنگامی که رفرنس به جایی خارج از store اشاره می‌کند.
- NameError: متغیری تعریف نشده باشد و از آن استفاده کنیم.
- TypeError: ارورهای مربوط به تایپ در این دسته هستند که در بخش قبل توضیحشان دادیم.

- Evaluation Error: زمانیکه متغیری در بدنه‌ی تابع به نقطه‌ای از store اشاره کند که شامل thunk یا expval نباشد.
- IndexError: اگر بخواهیم به ایندکسی بیشتر از سایز یک آرایه دسترسی پیدا کنیم.
- ArityError: تعداد ورودی‌های یک تابع با تعداد پارامترهای تعریف شده‌ی او یکی نباشند.
- DivideByZero: در یک عملیات تقسیم خارج قسمت 0 باشد.
- OperatorError: زمانیکه که یک علامتی تحت عنوان unary operator استفاده شده باشد و هیچکدام از – و not نباشند.
- SyntaxError: کد مطابق گرامر مدنظر نباشد.

در کد هر زمانیکه که نیز به ارور دادن باشد، تابع raise-runtime-error در اینترپرتر صدا می‌شود. اروری که raise می‌شود، توسط handlerهایی که تعبیه شده است گرفته می‌شوند. این هندلرها هنگام ران شدن برنامه و اپلای کردن یک تابع استفاده شده اند.