Meta Interpreter

Oscar Melin

2016-11-05

Tips angående rapportskrivning

TA BORT DETTA STYCKE INNAN DU LÄMNAR IN RAPPORTEN.

Målgrupp för rapporten är personer som har precis samma kunskaper som författaren, bortsett från att de inte vet något alls om det specifika program rapporten beskriver.

Tänk på följande:

- Är stavning och grammatik korrekt? Undviks talspråk? Är rapporten kort och koncis eller onödigt lång och ordrik?
- Är lösningen tydligt förklarad? Kommer läsaren att förstå programmet? Tänk på vad du själv skulle vilja veta om du läste om programmet.
- Bevisar rapporten att programmet i fråga fungerar? Detta kan göras genom att presentera prestandamätningar eller andra testresultat.
- Finns det en analys av testresultaten? Visar de programmets viktiga egenskaper? Borde det göras ytterligare utvärderingar?
- Framgår det vilka problem som uppstod under utvecklingen och hur de lösts?
- Finns det en utvärdering av arbetet? Framgår det vad författaren har lärt sig av uppgiften?
- Är texten illustrerad med kodsnuttar och/eller bilder och/eller tabeller? Undvik långa kodlistningar, ta bara med just den kod som är relevant och som förklaras i den löpande texten.
- Har rapporten en bra struktur med rubriker, underrubriker och stycken?

1 Uppgiften

Den här rapporten behandlar skapandet av en meta interpreter för ett minimalt funktionellt språk väldigt likt Erlang.

En interpreter är ett datorprogram som tar ett ett annat program som input och sedan tolkar och utför instruktionerna av detta program utan att först kompilera programmet till maskinkod. Vidare kallas en interpreter som är skriven i sammma eller ett liknande språk till språket som tolkas för en meta interpreter.

Uppgiften till fråga går ut på att implementera en meta interpreter kapabel att hantera variabeldeklarering, simpel mönstermatchning samt så ska den kunna tolka case of statements, likt de som finns i Erlang.

ADD: kortfattat hur uppgiften ska ösas

2 Ansats

2.1 Teori

Vårat nya språk som ska tolkas av meta interpretern kommer att bestå av två enheter vid tolkningsfasen, miljön och sekvensen av kod som ännu ej har tolkats. Miljön är en kartläggning mellan variabler och deras tilldelade värde. Exempelvis så representerar vi miljön {X/qwer, Y/asdf} som {{'X', qwer}, {'Y', asdf}}.

```
X = foo, Y = nil, {Z,_} = {X,asdf}, X.
```

Figur: En simpel sekvens.

En sekvens kan beskrivas i följande **BNF:**

⟨*sequence*⟩ ::= ⟨*expression*⟩
| ⟨*case expression*⟩
| ⟨*match*⟩ ',' ⟨*sequence*⟩

⟨*match*⟩ ::= ⟨*pattern*⟩ '=' ⟨*expression*⟩

⟨*expression*⟩ ::= ⟨*atom*⟩
| ⟨*variable*⟩
| ⟨*cons*⟩

Hur <case expression> definieras förklaras under praktik tillsammans med en förklaring av implementationen.

2.2 Praktik

Den praktiska implementeringen består av två huvudsakliga delar, delade upp i två moduler; **env.erl** och **eager.erl**.

2.2.1 env.erl

env.erl innehåller de operationer som krävs för att interagera med miljön såsom möjligheten att lägga till variabler, se ifall en variabel existerar och om så är fallet, returnera variabelns värde.

```
-module(env).
-export([new/0, add/3, lookup/2]).

new() ->
    [].

add(Id, Str, Env) ->
    Env ++ [{Id, Str}].

lookup(_, []) ->
    false;
lookup(Id, [{Id, Str} | _]) ->
    {Id, Str};
lookup(Id, [_ | R]) ->
    lookup(Id, R).
```

Exempelvis, följande funktionsanrop ska returnera {foo, 42}:

```
env:lookup(foo, env:add(foo, 42, env:new())).
```

2.2.2 eager.erl

eager.erl tolkar uttryck, mönstermatchar, utvärderar case expressions och räknar ut resultatet av en sekvens av dessa. Eftersom detta är en meta interpreter så representeras en atom och en cons-cell i detta språk av en Erlang atom och en Erlang cons-cell. Modulen består av tre funktioner.

eval_expr/2 tar ett uttryck och en miljö och returnerar antingen **{ok, S}** där S är en datastruktur, eller **error**. Vi har valt att uttrycka ett uttryck enligt nedan och således behöver **eval_expr/2** kunna skilja på dessa.

⟨*expression*⟩ ::= ⟨*atom*⟩
| ⟨*variable*⟩
| ⟨*cons*⟩

```
eval_expr({atm, Id}, _) ->
    {ok, Id};
eval_expr({var, Id}, Env) ->
    case env:lookup(Id, Env) of
        false ->
            error;
```

```
        {Id, Str} ->
            {ok, Str}
    end;
eval_expr({cons, {var, H}, {atm, T}}, Env) ->
    case eval_expr({var, H}, Env) of
        error ->
            error;
        {ok, V} ->
            {ok, {V, T}}
    end.
```

eval_match/3
eval_seq/2

3 Utvärdering

Uppgifterna i denna kurs kanske inte kräver en utvärdering men man kan här visa resultat från testkörningar mm. Om man vill lägga in resultat från testkörningar kan man sammanfatta dessa i en tabell. Ett exempel kan ses i tabell 1.

kärnor	tid	uppsnabbning
1	400ms	1
2	240ms	1.7
4	140ms	2.8

Tabell 1: Ha alltid en rad som förklarar vad tabellen handlar om.

Om ni vill ha med en graf så rekommenderas ni att använda gnuplot. Om ni lär er använda det för att göra enkla diagram så kommer ni ha mycket nytta av det i framtiden.

4 Sammanfattning

Vad gick bra och vad gick mindre bra? Vad var de största problemen och hur löstes de? Skriv en kort sammanfattning.