Interpretation of Computer Programs II
Academiejaar 2012-2013

Project: Lazy Tables

Joeri De Koster
Vrije Universiteit Brussel
jdekoste@vub.ac.be

Het oefeningen onderdeel van het examen bestaat uit het uitwerken van het project beschreven in dit document. Dit poject moet **individueel** gemaakt worden. Het doel van dit project is het uitbreiden van de Meta en de C implementatie van Pico met een aantal nieuwe taalconstructies ten einde het ondersteunen van lazy tabellen. Het mondeling examen zal bestaan uit twee delen. En deel is het theoretisch examen en het andere deel is een **mondelinge verdediging** van jouw uitwerking van het project.

Projectopgave

De bedoeling van dit project is streams (cfr. streams in Scheme) mogelijk te maken binnen Pico. In Scheme zijn streams gebaseerd op stream-cons-cellen (dwz. cons-cellen waarvan het cdr-deel een promise bevat; een promise wordt aangemaakt via een delay special form). In Pico zullen we streams moeten enten op tabellen[1].

We zullen "lazy" tabellen definiren als volgt:

```
LazyTabel[: Uitdrukking1, Uitdrukking2, ..., UitdrukkingN  :]:
  LazyUitdrukking(i)
```

waarbij `[:` en `:]` nieuwe tokens zijn in een uitgebreide grammatica. Het resultaat van de evaluatie van deze uitdrukking is een lazy tabel waarvan de

[1] In tegenstelling tot Scheme zullen we *geen* expliciete tegenhanger van de delay aanmaken.

eerste `n` velden ingevuld zijn met concrete waarden. Deze waarden worden opgebouwd door het evalueren van `Uitdrukking1` tot en met `UitdrukkingN`. Voor de velden waarvan de index groter is dan n wordt bij het aanmaken van de tabel geen waarde gespecificeerd. In plaats daarvan wordt er een `LazyUitdrukking(i)` meegegeven. Deze uitdrukking stelt de andere waarden van de tabel voor gegeven de index `i`.

Analoog aan gewone tabellen kan er een lazy tabel ook aangemaakt worden door gebruik te maken van de volgende syntactische suiker:

```
[: Uitdrukking1, Uitdrukking2, ..., UitdrukkingN, LazyUitdrukking(i) :]
```

De slots van een lazy tabel kunnen altijd overschreven worden als volgt:

```
LazyTabel[: index :] := Uitdrukking
```

met dezelfde semantiek als die voor gewone tabellen. Vanaf dan is de lazy tabel op deze indexplaats niet meer te onderscheiden van een gewone tabel.

Indien voorgaande niet gebeurd is voor een gegeven index (groter dan het aantal concrete waarden), dan zal een referentie naar:

```
LazyTabel[: index :]
```

de waarde van de evaluatie van `LazyUitdrukking` geparametriseerd met de `index` opleveren (LazyUitdrukking, is dus een functionele parameter van de lazy tabel met op zijn beurt n parameter, namelijk de index, `i`). **Let op:** de LazyUitdrukking is gescoped. Dit wil zeggen dat bij het evalueren van deze expressie deze toegang heeft tot zijn lexicale scope (inclusief zichzelf), uitgebreid met de binding van de parameter `i` aan de index.

Het volgende voorbeeld illustreert dit:

```
> f(x): t[::]: x+i
<function f>

> lazy_table: f(42)
[: ... :]

> lazy_table[:2:]
44

> lazy_table
[: <lazy>, 44, ... :]
```

In de implementatie verwachten we dat lazy waarden maar 1 keer berekend worden (en dus ergens bewaard worden). Indien de tabel met concrete waarden te klein is om de nieuwe waarde te stockeren zal er dus een dynamische herlocatie moeten gebeuren van de hele tabel. Waarden die nog niet gevalueerd zijn worden als `<lazy>` weergegeven. Het volgende voorbeeld illustreert dit:

```
> t[::]: { display("Hello "); "World!" }
[: ... :]

> t[:2:]
Hello World!

> t[:2:]
World!

> t
[: <lazy>, World!, ... :]
```

Opmerkingen

- Het aantal concrete uitdrukkingen kan ook nul zijn. In dat geval zijn alle waarden van de tabel lazy.
- In Lazy Pico is `[:uitdrukking:]` analoog aan `(delay uitdrukking)` in Scheme. Het equivalent van `(force lazy_waarde)` zou dan in Lazy Pico `lazy_waarde[:1:]` zijn.
- Het equivalent van `(stream_cons a b)` in Scheme zou dan in Lazy Pico [:a, b:] zijn.

Nog meer voorbeeldcode

```
> integers[::]: i
[: ... :]

> integers[:2:]
2

> integers
[: <lazy>, 2, ... :]
```

```
> sum_till(t): sum[:t[:1:]:]: t[:i:] + sum[:i-1:]
<function sum_till>

> s: sum_till(integers)
[: 1, ... :]

> s[: 3 :]
6

> s
[: 1, 3, 6, ... :]

> fibs[: 1, 1 :]: fibs[:i-1:]+fibs[:i-2:]
[: 1, 1, ... :]

> lazy_map(t, f): u[::]: f(t[:i:])
<function lazy_map>

> lazy_zip(t, u, f): v[::]: f(t[:i:], u[:i:])
<function lazy_zip>
```

Bonus

Als bonus kan je ervoor zorgen dat het concreet stuk van de lazy tabel voorgesteld wordt door een zoekboom (bv. binary search tree) zodat de plaats die de tabel van gekende concrete waarde inneemt proportioneel groeit met het aantal gekende waarden. De printer hoeft dan ook enkel de gekende waarden uit te printen. Het volgende voorbeeld illustreert dit:

```
> { integers[::]: i ; integers[:1000:] ; integers }
[: ..., 1000, ... :]
```

Conclusie

Gevraagd wordt Pico uit te breiden met lazy tabellen, volgens de semantiek hierboven beschreven. Het zal nodig zijn om de Pico grammatica uit te breiden met tokens en regels om lazy tabellen te kunnen gebruiken binnen Pico programma's. Vervolgens moet de abstracte grammatica uitgebreid worden met de representatie van definitie/gebuik/wijziging van lazy tabel, met lazy tabel waarden, met promises etc. Ten slotte moeten de nodige evaluatiefuncties gedefinieerd en uitgewerkt worden. In de geest van de Scheme

stream aanpak kan overwogen worden om stream-versies van de primitieve iteratiefuncties aan te maken.

De implementatie dient voorbereid worden via een metacirculaire versie. Deze moet dienen als specificatie voor de definitieve implementatie in C.