TDT4165 PROGRAMMING LANGUAGES

Fall 2012

# Exercise 4
# RC-kretser

## Før du starter

### Relevant lesestoff:

- Kapittel 4.3 i V&H.

### Du skal levere:

- `teori04.txt`: Svar på de teoretiske spørsmålene.
- `oving04.oz`: Løsning på programmeringsoppgavene.
- `sporsmal04.txt` (valgfritt): Mulige spørsmål du har.
- `tilbakemelding4.txt` (valgfritt): Tilbakemeldinger om faget: Hva er hovedproblemene? Hvordan kan det forbedres? Andre kommentarer?

### Du må svare på alle deloppgaver

Om du ikke klarer å løse en deloppgave må du forklare hva som er vanskelig med deloppgaven (din løsningsstrategi og hvorfor den ikke fungerer).

## RC-kretser

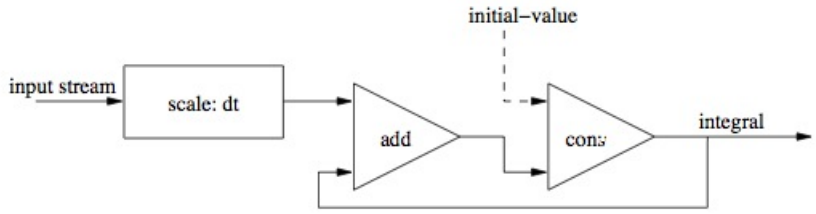
Denne øvingen bruker strømmer for å modellere enkle elektriske kretser. Du får prøvd lat utførelse, strømmer og høyere-ordens programmeringsteknikker.

## Strømmer og signaler

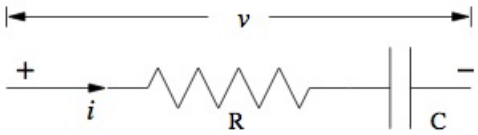
Vi bruker strømmer direkte til å modellere signaler ved å representere verdien av etterfølgende signalverdier som elementer i strømmen. For eksempel kan vi implementere en integrator eller summerer som tar en strøm som input, strøm $x = x_i$, en startverdi $C$ og en liten økning $dt$ og akkumulerer summen:

$$S_i = C + \sum_{j=1}^{i} x_j \, dt$$

og returnerer en strøm av verdier $S = (S_i)$. Figur 1 viser en modell av et signalprosesseringssystem som tilsvarer integralfunksjonen. Inputstrømmen blir skalert av $dt$ og passerer gjennom en adderer sammen med outputen.




Figur 1: Integralfunksjon sett som signalprosesseringssystem



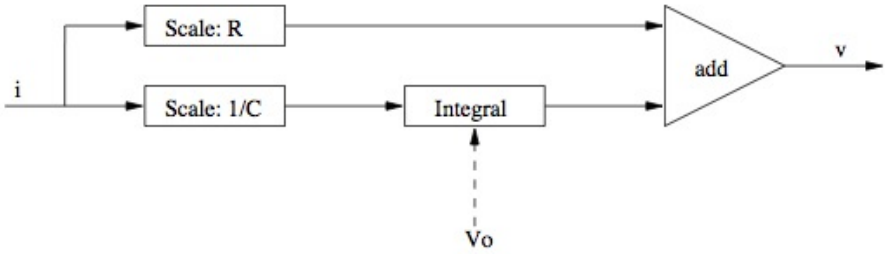

Figur 2: En RC-krets

## Modellere elektriske kretser

Vi kan modellere elektriske kretser med strømmer som representerer verdien av strøm eller spenning i en sekvens av tid. For eksempel, gitt en RC-krets bestående av en resistor med motstand $R$ og en kondensator med kapasitans $C$ i serie. Spenningen $v$ i en krets med strøm $i$ er gitt ved den følgende likningen.

$$v = v_0 + \frac{1}{C}\int_0^t i(t)\,dt + Ri$$

hvor $v_0$ er startspenningen i kondensatoren.




Figur 3: Et signalflytsdiagram for RC-kretsen

## Oppgave

For å implementere simuleringen med RC-kretsen må du lage seks funksjoner og lagre dem i filen `oving04.oz`. Heretter vil vi bruke `S`, `S1`, og `S2` til å referere til strømmmer, `F` for funksjon og `SF`, `SF1`, `SF2` for strømmer med flyttall.

Bruk funksjonene du implementerer til å implementere andre funksjoner når dette er mulig.

### Task 1 Teori

**a)** For å hente de N første elementene av en strøm S kan du kjøre {Nth S N _}. Forklar hva som skjer når denne funksjonen kjøres.

**b)** Hva er forskjellen på en strøm og en liste?

**c)** Hvordan virker unntak (exceptions) sammen med lat utføring. Hvordan må kjernespråket utvides for å støtte begge disse konseptene sammen?

### Task 2 Avbildning av strømmer

{`StreamMap S F`} bruker F på alle elementene i S og returnerer en ny strøm. Til dette brukes lat utførelse. For eksempel:

```
{StreamMap 1|2|3|4|... MultiplyWithFive}
```

returnerer `5|10|15|20|...`.

### Task 3 Slå sammen strømmer

{`StreamZip S1 S2 F`} slår sammen to strømmer på med lat utførelse. Den kjører funksjonen F på elementene på samme posisjon i S1 og S2. For eksempel:

```
local
   fun {Multiply X Y} X*Y end
in
   {StreamZip 1|2|3|... 4|5|6|... Multiply}
end
```

returnerer `4|10|18|...`.

Du kan anta at S1 og S2 er like lange

**Legg merke til:** De følgende funksjonene utgjør funksjonaliteten til diagrammet i figur 3. Bruk avbildning (mapping), folding og sammenslåing (zipping) for strømmer som definert over.

### Task 4 Skalere strømmer

{`StreamScale SF Factor`} skalerer elementene ved å multiplisere dem med en faktor (et flyttall). Dette skjer også med lat utførelse. For eksempel:

```
{StreamScale 1.0|2.0|3.0|4.0|....2.0}
```

returnerer strømmen `2.0|4.0|6.0|8.0|...`

### Task 5 Addere strømmer

{`StreamAdd SF1 SF2`} bruker lat utførelse til lage en strøm som er summen av elementene i SF1 og SF2. For eksempel:

```
{StreamAdd 1.0|1.0|1.0|... 2.0|3.0|4.0|...}
```

returnerer `3.0|4.0|5.0|...`

### Task 6 Integrere strømmer

Med lat utførelse returnerer {`StreamIntegrate SF InitValue Dt`} en strøm ved å integrere verdiene i flyttallstrømmen SF ved å bruke initialverdien InitValue og tidssteget Dt. Du kan se hvordan det fungerer i figur 1. For eksempel:

```
{StreamIntegrate 1.0|0.0|1.0|... 5.0 1.0}
```

returnerer `5.0|6.0|6.0|...`

og

```
{StreamIntegrate 5.0|6.0|7.0|... 2.0 3.0}
```

returnerer `2.0|17.0|35.0|...`

### Task 7 Simulere RC-kretser

{`MakeRC R C Dt`} tar inn motstand R, kondensatoren C og tidsteget dt som argumenter (alle er flyttall). Den returnerer en funksjon RC. Funksjonen {`RC SF V0`} tar inn flyttallstrømmen SF som representerer den elektriske strømmen sendt inn i kondensatoren og startspenningen V0 og returnerer en strøm med outputspenningen. For eksempel:

```
declare
fun lazy {MakeOnes} 1.0 | {MakeOnes} end
RC = {MakeRC 5.0 1.0 0.2}
Vs = {RC {MakeOnes} 2.0}
{Nth Vs 5 _}
{Browse Vs}
```

Viser fram `7.0|7.2|7.4|7.6|7.8|_<Future>` i Oz-browseren. Du kan se hvordan det fungerer i figur 3.

## En simulator-graf for elektriske RC-kretser

For å gjøre det lettere å teste og forstå har vi laget en simulator-graf (SG) for elektriske RC-kretser. Du må kontruere en modul som inneholder implementasjonen av den abstrakte datatypen. En modul er en record hvor feltene gir grensesnittets navn og verdiene er implementasjonen (funksjoner for oss). Konstruer den følgende modulen til oving4.oz:

```
Streams = streams(streamMap: StreamMap
                  streamZip: StreamZip
                  streamScale: StreamScale
                  streamIntegrate: StreamIntegrate
                  streamAdd: StreamAdd
                  makeRC: MakeRC)
```

For å bruke SG må du laste ned SimGraph.oz fra hjemmesiden til faget og kopiere den til den mappen du jobber fra. Kompiler den med følgende kommando i kommandovinduet når du har navigert til mappen du jobber fra:

```
ozc -c SimGraph.oz
```

Dette vil lage filen `SimGraph.ozf`. Last prosedyren Test:

```
Test={Module.link ['SimGraph.ozf']}.1.make
```

Kjør prosedyren Test med modulen som argument:

```
{Test Streams}
```

Om du ikke får noen feilmeldinger vil et vindu komme opp. Fra vinduet kan du velge typen funksjon for simulasjonen (*el-current*). Du kan gå gjennom simulasjonen ved å trykke på *step* eller *tenSteps*. Oz-browseren viser da frem de forventede verdiene og verdiene kalkulert med din RC-simulatormodul. Hvis de stemmer overens er du ferdig, ellers må du jobbe litt mer ;-)