Bachelor PRD **IT University, E2010**

Skriftlig eksamen, Programmer som Data
Onsdag 5. januar 2011

Version 1.1 af 2011-01-28

Dette eksamenssæt har 7 sider. Tjek med det samme at du har alle siderne.

Eksamens varighed er 4 timer.

Der er fire opgaver. For at få fuldt point skal du besvare alle delopgaverne tilfredsstillende. Du må bruge alle bøger, forelæsningsnoter, forelæsningsplancher, opgavesæt, opgavebesvarelser, lommeregnere og så videre under eksamen, men ingen computere (heller ikke mobiltelefoner, PDA, iPod, iPad eller lignende) som kan udføre programmer i F# eller C# eller Java, eller som kan kommunikere med andre enheder.

Hvis en delopgave kræver at du definerer en bestemt funktion, så må du gerne **bruge den funktion i efterfølgende delopgaver**, også selv om du ikke selv har defineret den.

Hvis en delopgave kræver at du definerer en bestemt funktion, så må du gerne **definere alle de hjælpefunktioner du vil**, men du skal definere den ønskede funktion så den har netop den type og det resultat som delopgaven kræver.

Opgave 1 (25 %): Regulære udtryk og automater

Opgave 1.1

Lav en deterministisk endelig tilstandsmaskine (DFA) der genkender en streng hvis

- strengens længde er ulige, **og**
- strengen består udelukkende af bogstaverne b og c og d.

Strenge der skal genkendes er for eksempel b og bbb og bcc og dcb; mens strenge der skal forkastes er for eksempel bb og bc og dde.

Opgave 1.2

Skriv et regulært udtryk der genkender samme strenge som automaten i delopgave 1.1.

Opgave 1.3

HTML, eller Hypertext Markup Language, bruges til at kode websider. Et HTML-tag (uden attributter) er enten et start-tag af formen `<foo>`, eller et slut-tag af formen `</foo>`, eller et indholdsløst tag af formen `<foo/>`, der er start- og slut-tag i ét. Strengen `foo` kan generelt være en vilkårlig ikke-tom sekvens af bogstaver og cifre.

Et regulært udtryk til at beskrive HTML-tag kan derfor skrives sådan her:

```
< A+ >  |  < / A+ >  |  < A+ / >
```

hvor vi lader `A` stå for et vilkårligt bogstav eller ciffer, mens < og > og / står for sig selv. Nogle eksempler på lovlige HTML-tags er således `<h2>` og `</h2>` og `<hr/>`.

Konstruér en ikke-deterministisk endelig automat (NFA) svarende til det regulære udtryk. I automaten kan du bruge `A` til at beskrive alle bogstaver og tal i stedet for at lave en separat tilstandsovergang for hvert bogstav og tal.

Opgave 1.4

Konstruér en deterministisk endelig automat (DFA) der genkender de samme strenge som den ikke-deterministiske automat i delopgave 1.3. Du skal enten bruge en systematisk konstruktion svarende til den i Mogensens bog, eller forklare hvorfor den deterministiske automat genkender præcis de samme strenge som det regulære udtryk i opgave 1.3.

Opgave 2 (25 %): Typetjek og typeinferens

Afsnit 4.8 i *Programming Language Concepts for Software Developers* indeholder et typesystem for et simpelt funktionelt sprog.

Antag nu at dette sprog udvides med par-udtryk af formen (e_1, e_2) og tilsvarende par-typer $t * u$, sådan at (e_1, e_2) har type $t * u$ hvis e_1 har type t og e_2 har type u. Dette kan beskrives med denne typeregel:

$$\frac{\rho \vdash e_1 : t \qquad \rho \vdash e_2 : u}{\rho \vdash (e_1, e_2) : t * u}\,(pair)$$

Opgave 2.1

Lav et typeinferenstræ for udtrykket $(\texttt{11} + \texttt{10}, (\texttt{true}, 22))$.

Opgave 2.2

Antag nu at sproget også har en operation `fst(`e`)` hvis værdi er første komponent af e, og en operation `snd(`e`)` hvis værdi er anden komponent af e. I begge tilfælde skal e have en par-type.

Lav en typeregel for `fst(`e`)` og en typeregel for `snd(`e`)`.

Opgave 2.3

Antag at sproget også har en ny version af `let`-binding, der ligesom i F# kan splitte et par i dets komponenter x og y, nemlig

$$\texttt{let}\ (x, y) = e_r\ \texttt{in}\ e_b\ \texttt{end}$$

Nedenfor er fem forslag til typeregler for denne nye slags `let`-binding. Kun en af dem er korrekt. Angiv nummeret på den korrekte regel, og forklar kort hvorfor det er den rigtige regel (eller hvorfor de andre regler er forkerte).

$$\frac{\rho \vdash e_r : t * u \qquad \rho[x \mapsto t, y \mapsto u] \vdash e_b : t_b}{\rho \vdash \texttt{let}\ (x, y) = e_r\ \texttt{in}\ e_b\ \texttt{end} : t_b}\,(1)$$

$$\frac{\rho \vdash e_r : t * u \qquad \rho[x \mapsto t, y \mapsto u] \vdash e_b : t}{\rho \vdash \texttt{let}\ (x, y) = e_r\ \texttt{in}\ e_b\ \texttt{end} : \texttt{int}}\,(2)$$

$$\frac{\rho \vdash e_r : t \qquad \rho[x \mapsto t, y \mapsto t] \vdash e_b : u}{\rho \vdash \texttt{let}\ (x, y) = e_r\ \texttt{in}\ e_b\ \texttt{end} : u}\,(3)$$

$$\frac{\rho[x \mapsto t, y \mapsto u] \vdash e_b : t_b}{\rho \vdash \texttt{let}\ (x, y) = e_r\ \texttt{in}\ e_b\ \texttt{end} : t_b}\,(4)$$

$$\frac{\rho[x \mapsto t, y \mapsto u] \vdash e_r : t * u \qquad \rho \vdash e_b : t}{\rho \vdash \texttt{let}\ (x, y) = e_r\ \texttt{in}\ e_b\ \texttt{end} : t}\,(5)$$

Opgave 3 (20 %): Parsing af spørgeskemadefinitioner

Denne opgave handler om et lille sprog til at definere spørgeskemaer i. En spørgeskemadefinition består af nøgleordet `questionnaire` efterfulgt af navnet på skemaet efterfulgt af en krop (i krølleparenteser) med nul eller flere spørgsmål.

Figur 1 viser et eksempel på en spørgeskemadefinition med ialt fem spørgsmål. Figur 2 på side 5 viser hvordan det tilsvarende spørgeskema kunne se ud den person for der skal udfylde det.

```
questionnaire bicyclequestionnaire
{
   number age /What is your age/;
   singlechoice brand /What brand is your bike/ [/Cannondale/ /Jensen/
       /Den beste sykkel/ /Other/];
   multichoice uses /What do you use your bike for/ [/Exercise/
       /Going to work//Shopping//Holidays//Other/];
   freetext bestexperience /What is your best biking experience/;
   optional freetext comments /Any other comments/;
}
```

Figure 1: Eksempel på en spørgeskemadefinition.

Et spørgeskema kan indeholde nedenstående fire forskellige slags spørgsmål; figur 1 indeholder eksempler på alle fire slags:

- Spørgsmål der besvares med en vilkårlig tekst (nøgleord `freetext`); spørgsmålet skal have et navn (fx `bestexperience`) samt en spørgsmålstekst (fx `What is your best biking experience`) omgivet af skråstreger.
- Spørgsmål der besvares med indtastning af et tal (nøgleord `number`); de skal have samme bestanddele som `freetext`.
- Spørgsmål der besvares med afkrydsning af én mulighed ud af flere (nøgleord `singlechoice`); spørgsmålet skal have navn, spørgsmålstekst og en liste af én eller flere svarmuligheder, hver omgivet af skråstreger.
- Spørgsmål der besvares med afkrydsning af én eller flere muligheder (nøgleord `multichoice`); de skal have samme bestanddele som `singlechoice`.

Desuden kan et spørgsmål være frivilligt, hvilket angives med nøgleordet `optional` foran spørgsmålsdefinitionen. Det gælder fx sidste spørgsmål i spørgeskemadefinitionen i figur 1.

Den abstrakte syntaks for spørgeskemadefinitioner i F# er vist nedenfor. Den Boolske værdi i et spørgsmål er sand hvis spørgsmålet er frivilligt; den første string er spørgsmålets navn; den anden er spørgsmålets tekst; og en `string list` er en liste af svarmuligheder:

```
module Absyn
type question =
     | Freetext of bool * string * string
     | Number of bool * string * string
     | Singlechoice of bool * string * string * string list
     | Multichoice of bool * string * string * string list
type questionnaire = string * question list
```

Et eksempel på denne abstrakte syntaks ses i opgave 3.3.

Opgave 3.1

Skriv en uformel grammatik for spørgeskemadefinitioner. Læg mærke til hvor der kan være nul eller flere, henholdvis én eller flere, forekomster af fx spørgsmål og svarmuligheder og lav grammatikken så den afspejler dette. Bemærk at der skal være et afsluttende semikolon (`;`) efter hver spørgsmålsdefinition. En tekst omgivet af skråstreger kan du tænke på som en enkelt token kaldet `TEXT`.

Opgave 3.2

Skriv regeldelen af parserspecifikationen for spørgeskemadefinitioner. Du behøver ikke skrive de semantiske aktioner i dette spørgsmål.

Du kan antage at der findes tokens (`QUESTIONNAIRE`, `FREETEXT`, ...) svarende til de forskellige nøgleord, at der findes tokens `LBRACE`, `RBRACE`, `LBRACKET` og `RBRACKET` svarende til de fire slags parenteser `{}[]`, og `SEMI` svarende til semikolon. Desuden findes et token `NAME` svarende til et enkelt navn, og et token `TEXT` svarende til en tekst omsluttet af skråstreger, fx `/Jensen/`; begge har en tilknyttet `string` værdi.

Tokenspecifikationen og regeldelen kunne således have denne form; du skal færdiggøre regeldelen:

```
%token <string> NAME TEXT
%token FREETEXT MULTICHOICE NUMBER OPTIONAL QUESTIONNAIRE SINGLECHOICE
%token LBRACE RBRACE LBRACK RBRACK SEMI

%start Questionnaire
%type <Absyn.questionnaire> Questionnaire

%%

Questionnaire:
    QUESTIONNAIRE NAME LBRACE ...
;
...
```

Opgave 3.3

Udvid parserspecifikationen fra delopgave 3.2 med semantiske aktioner indeholdt i `{ ... }`, sådan at parseren konstruerer abstrakt syntaks af type `questionnaire` svarende til den konkrete syntaks.

For spørgeskemadefinitionen i figur 1 skal der produceres denne abstrakte syntaks:

```
("bicyclequestionnaire",
 [Number (false,"age","What is your age");
  Singlechoice
    (false,"brand","What brand is your bike",
     ["Cannondale"; "Jensen"; "Den beste sykkel"; "Other"]);
  Multichoice
    (false,"uses","What do you use your bike for",
     ["Exercise"; "Going to work"; "Shopping"; "Holidays"; "Other"]);
  Freetext (false,"bestexperience","What is your best biking experience");
  Freetext (true,"comments","Any other comments")])
```

Opgave 4 (30 %): F#-funktioner til at generere spørgeskema

Et spørgeskema kan vises på mange forskellige måder, for eksempel i en browser, som i figur 2:

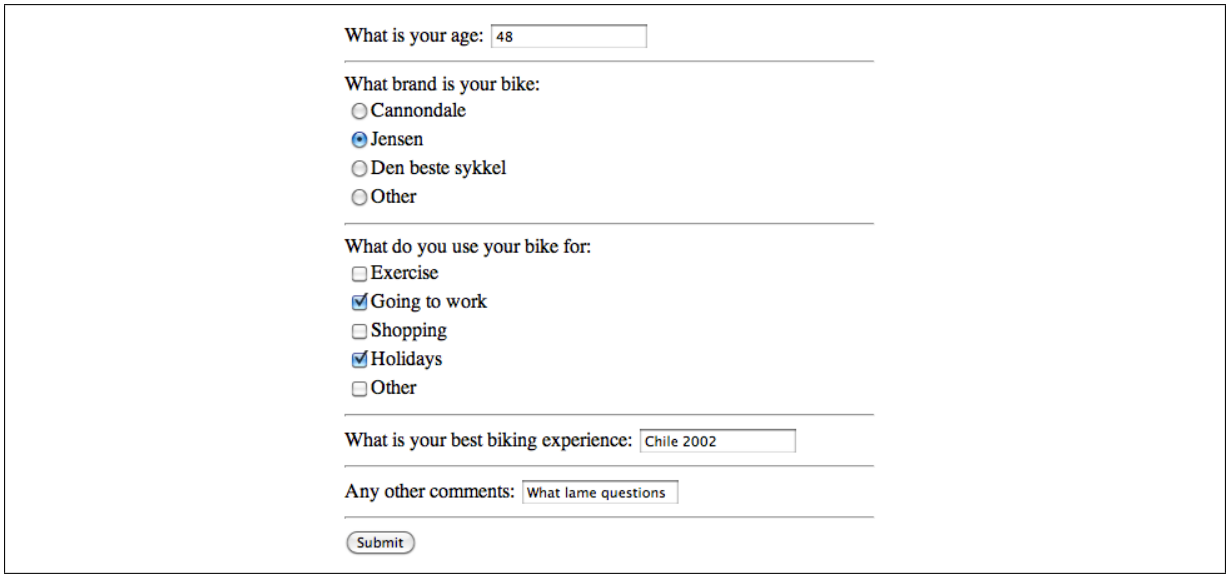

Figure 2: En browsers visning af et spørgeskema (delvis besvaret) svarende til definitionen i figur 1.

For at vise et spørgeskema i en browser skal spørgeskemadefinitionen oversættes til HTML-koder som browseren kan fortolke. Figur 3 nedenfor indeholder HTML-koder der bliver vist som spørgeskemaet i figur 2:

```
What is your age: <input type="text" name="age"/>
<hr/>
What brand is your bike:
<table>
<tr><td><input type="radio" name="brand" value="Cannondale">Cannondale</input></td></tr>
<tr><td><input type="radio" name="brand" value="Jensen">Jensen</input></td></tr>
<tr><td><input type="radio" name="brand" value="Den beste sykkel">Den beste sykkel</input></td></tr>
<tr><td><input type="radio" name="brand" value="Other">Other</input></td></tr>
</table>
<hr/>
What do you use your bike for:
<table>
<tr><td><input type="checkbox" name="uses" value="Exercise">Exercise</input></td></tr>
<tr><td><input type="checkbox" name="uses" value="Going to work">Going to work</input></td></tr>
<tr><td><input type="checkbox" name="uses" value="Shopping">Shopping</input></td></tr>
<tr><td><input type="checkbox" name="uses" value="Holidays">Holidays</input></td></tr>
<tr><td><input type="checkbox" name="uses" value="Other">Other</input></td></tr>
</table>
<hr/>
What is your best biking experience: <input type="text" name="bestexperience"/>
<hr/>
Any other comments: <input type="text" name="comments"/>
<hr/>
<input type="submit"/>
```

Figure 3: HTML-koder der danner spørgeskemaet vist i figur 2.

Resten af denne opgave går ud på at skrive F#-funktioner der genererer HTML-koder som tekst (type `string`) ud fra den abstrakte syntaks for spørgeskemadefinitioner. Opgaven kræver ikke forudgående kendskab til HTML.

En HTML-tekst består af *tags*. Et tag kan enten være indholdsløst, såsom `<hr/>` eller `<input ...>`; eller have et indhold, såsom `<tr>...</tr>` hvor `<tr>` er start-tag, `</tr>` er det tilhørende slut-tag, og `...` er taggets indhold. Begge slags tags kan indeholde attributter, som er en liste af bindinger adskilt med mellemrum, fx `type="text" name="age"`. Se eksemplerne i figur 3.

Tagget `<hr/>` betyder "horizontal rule" og vises som en vandret linje; det bruges her til at adskille spørgsmål. Tagget `<input .../>` bruges til at generere tekstbokse, trykknapper mv. Tagget `<table>...</table>`

bruges til at layoute valgmulighederne så de kommer pænt under hinanden; tagget `<tr>...</tr>` giver en ny tabelrække ("table row"), og `<td>...</td>` indeholder et element i en tabelrække ("table data").

Som det ses begynder hvert spørgsmål med en ledetekst og fortsætter med nogle `<input .../>` tags, eventuelt indpakket i et `<table>...</table>` tag af hensyn til layout.

HTML er ligeglad med linjeskift og ekstra mellemrum, så dem skal du ikke bekymre dig om i denne opgave.

Du kan antage at der findes en F# funktion `enquote : string -> string` der sætter dobbelte anførselstegn om den givne tekst:

```
let enquote str = "\"" + str + "\""
```

Opgave 4.1

Definér en F# funktion `makeAttributes : (string * string) list -> string` der tager en liste af par af attributnavne og attributværdier, og producerer en streng af bindinger som kan bruges i et HTML-tag. Attributværdierne skal sættes i dobbelte anførselstegn.

For eksempel skal `makeAttributes [("type", "text"); ("name", "brand")]` give strengen `type="text" name="brand"`.

Opgave 4.2

Definér en F# funktion `tag0 : string -> (string * string) list -> string` der tager et tagnavn og en attributliste og producerer en streng svarende til et indholdsløst tag med de givne attributter.

For eksempel skal `tag0 "input" [("type", "text"); ("name", "brand")]` give strengen `<input type="text" name="brand"/>`.

Definér en F# funktion `tag1 : string -> (string * string) list -> string -> string` der tager et tagnavn og en attributliste og et indhold og producerer en streng svarende til et HTML-tag med de givne attributter og med den tredje streng som indhold.

For eksempel skal `tag1 "table" [] "indhold"` give strengen `<table>indhold</table>`.

Opgave 4.3

Definér en F# funktion `makeFreetext : string -> string -> string` der tager et spørgmålsnavn og et spørgsmål og producerer HTML-koderne svarende til et fritekstspørgsmål.

Fx skal `makeFreetext "comments" "Any other comments"` give denne streng:
`Any other comments: <input type="text" name="comments"/>`, som vist nederst i figur 3.

Opgave 4.4

Definér en F# funktion `makeTable : string list -> string` der tager en liste af strenge og producerer en streng som repræsenterer en HTML-tabel hvor hver streng fra strenglisten kommer på en tabellinje for sig selv.

For eksempel skal `makeTable ["linje et"; "linje to"]` give strengen:

```
<table>
<tr><td>linje et</td></tr>
<tr><td>linje to</td></tr>
</table>
```

Opgave 4.5

Definér en F# funktion `makeSinglechoice : string -> string -> string list -> string` der tager et spørgsmålsnavn og et spørgsmål og en liste af svarmuligheder, og producerer en streng svarende til et valgspørgsmål der skal gives netop ét svar på.

For eksempel skal

```
makeSinglechoice "brand" "What brand is your bike"
                 ["Cannondale"; "Jensen"; "Den beste sykkel"; "Other"]
```

give denne streng (også vist i figur 3):

```
What brand is your bike:
<table>
<tr><td><input type="radio" name="brand" value="Cannondale">Cannondale</input></td></tr>
<tr><td><input type="radio" name="brand" value="Jensen">Jensen</input></td></tr>
<tr><td><input type="radio" name="brand" value="Den beste sykkel">Den beste sykkel</input></td></tr>
<tr><td><input type="radio" name="brand" value="Other">Other</input></td></tr>
</table>
```

Opgave 4.6

Antag nu at der er defineret passende F#-funktioner `makeFreetext`, `makeNumber`, `makeSinglechoice` og `makeMultichoice` på samme måde som i delopgave 4.3 og 4.5 ovenfor.

Definér en F# funktion `makeQuestion : question -> string` der tager et spørgsmål som argument og producerer en tilsvarende HTML-tekst. Benyt funktionerne `makeFreetext` osv. i besvarelsen.

For eksempel skal

```
makeQuestion (Number(false, "age", "What is your age"))
```

give strengen

```
What is your age: <input type="text" name="age"/>
```

(Den Boolske `optional`-værdi i `question`-typen spiller ikke nogen rolle for generering af HTML-koden. Værdien kunne bruges til at tjekke at spørgeskemaer er korrekt udfyldt, men det falder helt uden for denne opgave).