

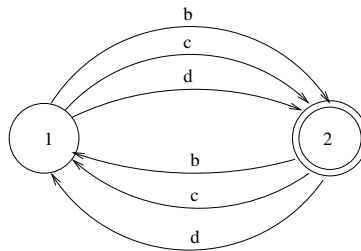
Løsningsforslag
Skriftlig eksamen

5. januar 2011

Version 3, 2011-01-28

Spørgsmål 1

Spørgsmål 1.1

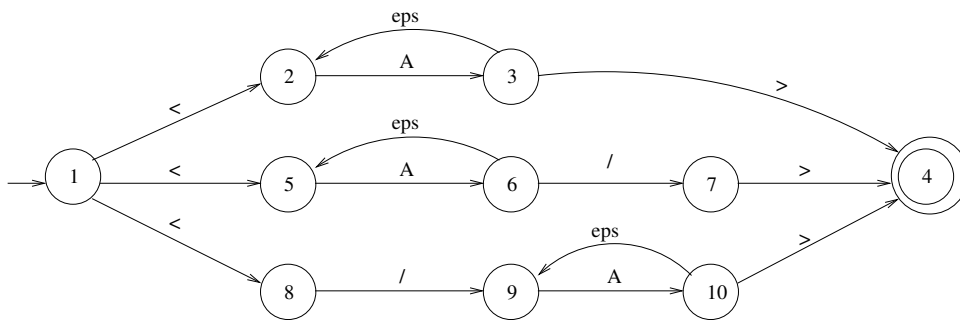


Spørgsmål 1.2

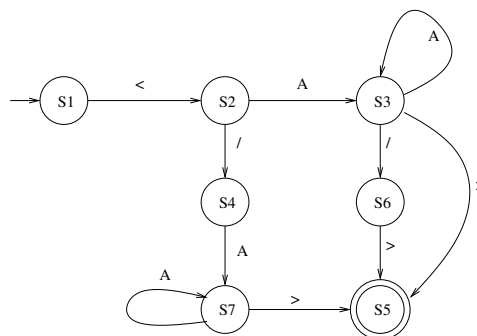
Det regulære udtryk kunne være: $(b|c|d)((b|c|d)(b|c|d))^*$

Spørgsmål 1.3

Her er en let reduceret version af den NFA man får fra Mogensens opskrift, idet der benyttes den oplagte konstruktion for én-eller-flere gentagelsen A^+ :



Spørgsmål 1.4



Spørgsmål 2**Spørgsmål 2.1**

$$\frac{\frac{\frac{}{\rho \vdash 11 : \text{int}} (1) \quad \frac{}{\rho \vdash 10 : \text{int}} (1)}{\rho \vdash 11 + 10 : \text{int}} (4) \quad \frac{\frac{}{\rho \vdash \text{true} : \text{bool}} (2) \quad \frac{}{\rho \vdash 22 : \text{int}} (1)}{\rho \vdash (\text{true}, 22) : \text{bool} * \text{int}} (\text{pair})}{\rho \vdash (11 + 10, (\text{true}, 22)) : \text{int} * (\text{bool} * \text{int})} (\text{pair})$$

Spørgsmål 2.2

$$\frac{\rho \vdash e : t * u}{\rho \vdash \text{fst}(e) : t} (\text{fst}) \qquad \frac{\rho \vdash e : t * u}{\rho \vdash \text{snd}(e) : u} (\text{snd})$$

Spørgsmål 2.3

Regel (1) er korrekt. Den siger at udtrykket e_r skal have par-type $t * u$, og let-kroppen e_b skal typetjekkes i et environment hvor x har type t og y har type u . Dette typetjek skal give en type t_b som så er typen for hele let-udtrykket.

Regel (2) er forkert fordi e_b kunne være false , der jo umuligt kan have type int .

Regel (3) er forkert fordi den ikke kræver at e_r har en par-type.

Regel (4) er forkert af samme grund; faktisk kræver ikke en gang at e_r er type-korrekt.

Regel (5) er forkert af to grunde: for det første er det urimeligt at kræve at e_b har samme type t som første komponent af e_r ; for det andet tillader den at x og y optræder i e_r hvilket er meget ejendommeligt — det ville tillade udtryk som dette: $\text{let } (x, y) = (x, y) \text{ in } x \text{ end}$ som det er svært at tillægge nogen værdi.

Spørgsmål 3

Spørgsmål 3.1

Ligesom der er et terminalsymbol (eller token) TEXT for tekster omgivet af skråstreger, antager jeg at der er et terminalsymbol LABEL for navne på spørgeskemadefinitioner og spørgsmål.

```

Questionnaire ::=
    "questionnaire" LABEL "{" Questions "}"
Questions ::=
    /* empty */
    | Question Questions
Optional ::=
    /* empty */
    | "optional"
BaseQuestion ::=
    Optional "freetext" LABEL TEXT ";"
    | Optional "number" LABEL TEXT ";"
    | Optional "singlechoice" LABEL TEXT Textlist ";"
    | Optional "multichoice" LABEL TEXT Textlist ";"
Texts ::=
    TEXT
    | TEXT Texts
Textlist ::=
    "[" Texts "]"

```

Spørgsmål 3.2 og 3.3

```

Main:
    QUESTIONNAIRE NAME LBRACE Questions RBRACE { ($2, $4) }
;

Questions:
    /* empty */                { [] }
    | Question Questions        { $1 :: $2 }
;

Optional:
    /* empty */                { false }
    | OPTIONAL                  { true }
;

Question:
    Optional FREETEXT NAME TEXT SEMI           { Freetext($1, $3, $4) }
    | Optional NUMBER NAME TEXT SEMI          { Number($1, $3, $4) }
    | Optional SINGLECHOICE NAME TEXT Textlist SEMI { Singlechoice($1, $3, $4, $5) }
    | Optional MULTICHOICE NAME TEXT Textlist SEMI { Multichoice($1, $3, $4, $5) }
;

Texts:
    TEXT                { [$1] }
    | TEXT Texts        { $1 :: $2 }
;

Textlist:
    LBRACK Texts RBRACK          { $2 }
;

```

Spørgsmål 4

Spørgsmål 4.1

Lidt fremsyn viser at det er nyttigt at definere en hjælpefunktion `mapAndConcat f xs` af type `('a -> string) -> 'a list -> string` som anvender funktion `f` på alle elementer af listen `xs` og konkatenerer de resulterende strenge. (Funktionerne `String.concat` og `List.map` findes i PLCSD appendix A, for eksempel):

```
let mapAndConcat f xs = String.concat "" (List.map f xs)

let makeAttributes attributes =
  mapAndConcat (fun (a, v) -> " " + a + "=" + enquote v) attributes
```

Man kan selvfølgelig også definere `makeAttributes` direkte, rekursivt:

```
let rec makeAttributes attributes =
  match attributes with
  | [] -> ""
  | (a, v) :: rest -> " " + a + "=" + enquote v + makeAttributes rest
```

Spørgsmål 4.2

Her skal man selvfølgelig benytte sig af den netop definerede `makeAttributes` funktion:

```
let tag0 tag attributes =
  "<" + tag + makeAttributes attributes + ">";;

let tag1 tag attributes contents =
  "<" + tag + makeAttributes attributes + ">" + contents + "</" + tag + ">";;
```

Spørgsmål 4.3

```
let makeFreetext label ask =
  ask + ": " + tag0 "input" [("type", "text"); ("name", label)];;
```

Spørgsmål 4.4

Her bruges hjælpefunktionen fra 4.1. For at få hver tabellinje pænt på sin egen tekstlinje tilføjes et linjeskift `"\n"` men der står udtrykkeligt i opgaven at dette skal man ikke bekymre sig om:

```
let makeTable lines =
  tag1 "table" []
    (mapAndConcat (fun line -> tag1 "tr" [] (tag1 "td" [] line) + "\n") lines)
```

Løsningen kunne nok være blevet en anelse klarere ved at indføre nye hjælpefunktioner `tr` og `td`:

```
let tr contents = tag1 "tr" [] contents
let td contents = tag1 "td" [] contents
let makeTable lines =
  tag1 "table" [] (mapAndConcat (fun line -> tr (td line) + "\n") lines)
```

Spørgsmål 4.5

Også her bliver svaret klarere ved at definere en hjælpefunktion `makeSinglechoice1` som genererer HTML-kode svarende til en enkelt valgmulighed.

```
let makeSinglechoice1 label choice =
  tag1 "input" [("type", "radio"); ("name", label); ("value", choice)] choice

let makeSinglechoice label ask choices =
  ask + ":\n" + makeTable (List.map (makeSinglechoice1 label) choices)
```

Bemærk at hjælpefunktionen `makeSinglechoice1` kaldes med kun ét argument `label`; resultatet er en partielt anvendt funktion af type `string -> string` som er den `List.map` anvendes på.

Alternativt kan man anvende en anonym funktion:

```
let makeSinglechoice label ask choices =
  ask + "\n"
  + makeTable (List.map (choice -> tag1 "input" [{"type", "radio"}; {"name", label};
                                               {"value", choice}]))
              choices)
```

Eller man kan definere en rekursiv hjælpefunktion hvis man ikke kan huske at `List.map` findes eller hvordan den virker — men det skulle man gerne kunne.

Spørgsmål 4.6

```
let makeQuestion question =
  match question with
  | Freetext(isOptional, label, ask)          -> makeFreetext label ask
  | Number(isOptional, label, ask)          -> makeNumber label ask
  | Singlechoice(isOptional, label, ask, choices) -> makeSinglechoice label ask choices
  | Multichoice(isOptional, label, ask, choices) -> makeMultichoice label ask choices
```