AANVULLENDE TOETS Grammatica's en Ontleden, 25 februari 2004, 9–12 uur
De drie opgaven van dit tentamen tellen mee voor 20%, 40% en 40% van het totaal.

1. (a) Als L en M talen zijn, welke taal wordt er dan aangeduid met $L\ M$?

   (b) Als L en M contextvrije talen zijn, is $L\ M$ dan ook een contextvrije taal?
   Zo ja: geef een bewijs; zo nee: geef een tegenvoorbeeld.

   (c) Als L en M reguliere talen zijn, is $L\ M$ dan ook een reguliere taal?
   Zo ja: geef een bewijs; zo nee: geef een tegenvoorbeeld.

2. In deze opgave bekijken we de taal van een eenvoudig soort predicaten (logische uitdrukkingen met variabelen). De basispredicaten worden gevormd door de operator $\leq$ te gebruiken tussen twee rekenkundige uitdrukkingen, zoals bijvoorbeeld $x \leq 10$ en $3 \leq x + y$. De rekenkundige uitdrukkingen op hun beurt zijn een natuurlijk getal, een éénletterige variabele, of een optelling van twee andere uitdrukkingen.

   Twee predicaten kunnen gecombineerd worden met de and-operator $\wedge$, en je kunt de not-operator $\neg$ voor een predicaat zetten. (In dit taaltje is er geen or-operator, implicatie, of ander fraais, en ook geen andere relaties dan $\leq$). Een voorbeeld van een predicaat in de taal is $\neg x \leq 5 \wedge 1 + 2 \leq y$.

   Een laatste manier om een predicaat te maken is door het gebruik van de universele quantificatie $\forall$. In deze taal moet bij zo'n quantificatie een eindige deelverzameling van de natuurlijke getallen worden aangegeven waarover de quantificatie zich uitstrekt. Het volgende voorbeeld geeft de precieze syntax aan:

   $$\forall x \in [y..5] : y \leq x \wedge x \leq 5$$

   Het symbool $\in$, de vierkante haakjes, de twee punten en de dubbele punt zijn scheidingstekens. In de onder- en bovengrens kunnen ook weer variabelen voorkomen.

   (a) Geef twee Haskell-datatypen `Expr` en `Pred`, die de abstracte syntax van deze taal beschrijven.

   (b) De prioriteiten van de operatoren zijn zodanig dat, in het voorbeeld $1 + 2 \leq y$, de expressie $1 + 2$ bij elkaar hoort, en niet $2 \leq y$. De prioriteit van de operator $\neg$ is zo, dat de $\neg$ uit het voorbeeld $\neg x \leq 5 \wedge 1 + 2 \leq y$ alleen slaat op $x \leq 5$, en dus niet op $1+2 \leq y$. Een quantificatie daarentegen gaat wel zo ver mogelijk door; in het voorbeeld hierboven wordt dus ook de x in $x \leq 5$ door de quantor gebonden.

   De auteur van een predicaat kan echter een andere prioriteit afdwingen door ronde haakjes te gebruiken, bijvoorbeeld $\neg((\forall x \in [1..2] : x \leq 3) \wedge 2 \leq 1)$.

   Schrijf met behulp van de parser-combinatorlibrary een parser voor expressies en een parser voor predicaten, met als onleedresultaat een waarde van de respectievelijke datatypes uit (a). Ook de parser *natural*, die een natuurlijk getal herkent, zit in de library).

   Je mag aannemen dat de speciale symbolen uit deze taal ($\wedge$, $\leq$, $\in$ enzovoort) gewone characters zijn, en ze dus gewoon tussen kwootjes opschrijven. Je hoeft geen rekening te houden met spaties.

   (c) Definieer een Haskell-type `PredAlgebra`, die het tupel beschrijft dat aan de functie `foldPred` meegegeven kan worden om een betekenis van een predicaat te bepalen.

   (Omdat de abstracte syntax uit twee types bestaat, kent deze algebra twee carrier-sets!). Je hoeft de functie `foldPred` niet zelf te schrijven.

   (d) Definieer een PredAlgebra `free`, zo dat `foldPred free :: Pred -> [Char]` de lijst van vrije variabelen geeft die in een predicaat voorkomen.

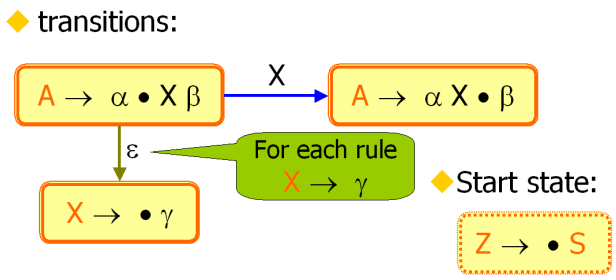
   (e) Definieer een PredAlgebra waarmee de waarheid van een predicaat kan worden bepaald. Je mag hierbij gebruikmaken van een

   ```
   type Env = [ (Char,Int) ]
   ```

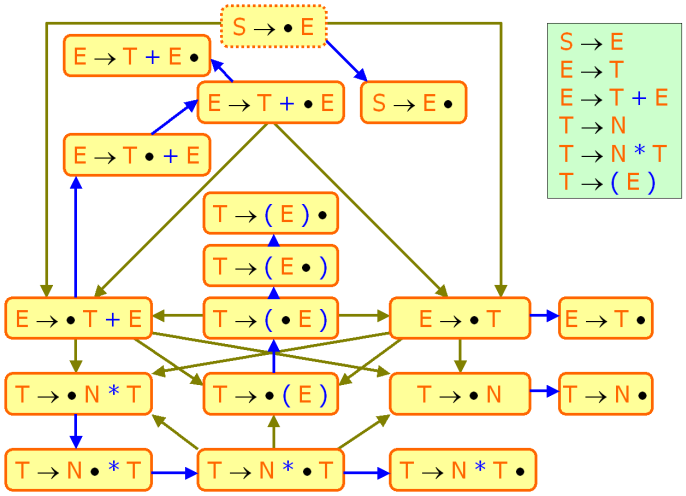
   en een bijbehorende opzoek-operator `(?)` .

3. *In deze opgave komt LR-ontleden ter sprake. Als je even niet meer weet wat dat is, probeer dan toch onderdeeel a en b van de opgave te maken, want die gaan eigenlijk alleen maar over toestandsautomaten!*

   Bij LR-ontleden wordt gebruik gemaakt van een toestands-automaat, waarbij de toestanden zijn gelabeld met regels uit de grammatica met in die regel een 'cursorpositie'. Er zijn twee soorten toestandsovergangen:



   Hieronder is een voorbeeldgrammatica met zes regels gegeven, en de toestandsautomaat die daar bij geconstrueerd is (de bijschriften bij de pijlen zijn weggelaten).



   (a) Hoe hangt het aantal toestanden af van de grammatica?

   (b) Is deze toestandsautomaat *deterministisch* of *nondeterministisch*? Waar blijkt dat uit?

   (c) De LR-ontleedfunctie maakt gebruik van een stack. Bij elke stap is er de keus uit een *shift* of een *reduce*. Een *shift* betekent dat er een symbool van de input wordt gelezen en op de stack gezet. Maar wat is een *reduce* ?

   (d) Om de keuze tussen *shift* en *reduce* te maken speelt de toestandsautomaat een rol. Op welke manier? (In woorden beschrijven, hoeft niet in Haskell).

   (e) Op welk moment termineert de LR-ontleedfunctie?
   Op welke manier is dit duaal aan LL-ontleden?