PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen, J. Berthold

D-35032 Marburg
Hans Meerwein Straße
Lahnberge
22. Oktober 2002

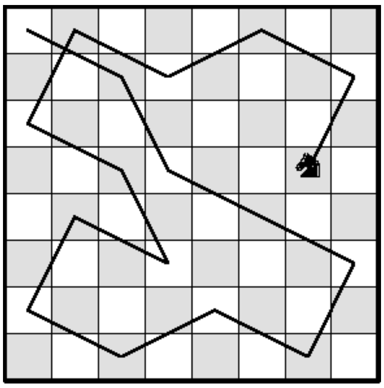
Übungen zu „Parallelität in funktionalen Sprachen“

Nr. 2, Abgabe: 5. November in der Vorlesung

Hinweise: Die Lösungen sollten grundsätzlich schriftlich, Programme zusätzlich auf Diskette oder per E-Mail an `eden@mathematik.uni-marburg.de` abgegeben werden.

Die Abgabe ist in Gruppen bis zu zwei Personen erlaubt.

Das Springerproblem (L. Euler, ca. 1759):
Ein Springer auf einem Schachbrett gegebener Größe soll, beginnend mit einem vorgegebenen Startfeld, jedes Feld des Bretts genau einmal besuchen[1].
Dieses Problem ist eine klassische Anwendung für sog. *backtracking*, d.h. Algorithmen, die eine Teillösung eines gegebenen Problems systematisch zu einer Gesamtlösung ausbauen und ihre Schritte rückgängig machen, falls in einer gewissen Situation ein weiterer Ausbau einer vorliegenden Teillösung nicht mehr möglich ist ("Sackgasse").
Der Backtracking-Algorithmus:



```
suche Weg:
   wenn neuer Zug moeglich
          suche nicht getesteten Zug aus
          wenn Brett noch nicht voll
                suche Weg
          sonst gib Gesamtweg aus und beende (fertig)
   sonst (kein neuer Zug moeglich)
      nimm letzten Zug zurueck
      suche Weg
```

Mit diesem Übungsblatt wird ein einfaches sequenzielles Haskell-Programm erstellt, das zu vorgegebener Spielfeldgröße und Startposition *einen* Weg findet. Später können mit dem Programm alle Wege gefunden werden (hoher Aufwand).
Das Spielbrett wird durch ein Feld von `Int` dargestellt und wird nach und nach mit der Nummer des aktuellen Zugs markiert. Positionen auf dem Spielbrett haben den Typ `(Int,Int)`.

```
data Board = Brd [[Int]]
empty :: Board
empty = Brd (replicate boardsize (replicate boardsize 0))
type Position = (Int, Int) -- intern: (0-..,0-..) Eingabe als (a-..,1-..)
```

Auf der Vorlesungsseite finden Sie einen Programmrumpf, der bereits diese Datentypen und die Ein- und Ausgabe enthält. Eine Startposition wird vom Benutzer erfragt und eine Lösung dazu berechnet (sofern existent).

---

[1] Mehr Information dazu gibt es z.B. unter http://www.ktn.freeuk.com/ und http://www.borderschess.org/KnightTour.htm.

1. Wir benötigen zuerst einige Hilfsfunktionen. 7 Punkte

   (a) Programmieren Sie eine Funktion, die zu gegebener Position und gegebenem Spielfeld die möglichen nächsten Züge berechnet:

   `moves :: Position -> Board -> [Position]`

   (b) Weiter muss das im Zug `i` besuchte Feld auf dem Spielbrett markiert werden.

   `markPosition :: Int -> Position -> Board -> Board`

   Programmieren Sie auch diese Funktion.

2. Programmieren Sie nun den rekursiven Backtracking-Algorithmus als Funktion. 8 Punkte

   (a) Ermitteln Sie den Typ der Funktion `tryPos` aus dem Aufruf in `main`.

   (b) Definieren Sie die rekursive Funktion mit den Hilfsfunktionen aus 1.
   Unter welchen Bedingungen wird die Rekursion abgebrochen?

3. Eine einfache Optimierung des Springerproblems ist, das Spielfeld auf Felder zu untersuchen, die nicht mehr erreichbar sind[2]. Implementieren Sie eine Suche nach solchen isolierten Feldern auf dem Spielbrett. 5 Punkte

   `isolatedFields :: Board -> Bool`

   Wo kann diese Funktion zur Effizienzsteigerung eingesetzt werden?

Tip: Erst ab einer Mindestgröße von 5 existieren Lösungen. Bei ungerader Dimension ist die Existenz der Lösung vom Startfeld abhängig. (Warum?)

[2] Andere Ansätze zielen darauf ab, möglichst schnell *einen* Weg zu finden, indem der nächste Zug "schlau" gewählt wird. Dies eignet sich natürlich nicht, um *alle* Wege zu finden.