# Haskellからのグラフ描画 – Graphvizの使い方 –

溝口佳寛

2006/11/21

## 1 準備

Graphvizとはグラフ描画のためのソフトウェアです. ホームページ

http://www.graphviz.org/

からダウンロード出来ます.

## 2 HaskellのGraphクラス

### 2.1 必要なモジュール

```
import System
import Data.Graph.Inductive
import Data.Graph.Inductive.Example
import Data.Graph.Inductive.Graphviz
import Data.Graph.Inductive.Internal.Heap
```

### 2.2 dottyによる描画

```
dotty_program = "dotty "
-- dotty_program = "lneato "

dotDisplay ::(Show a,Show b)=>FilePath -> (Gr a b)-> IO ExitCode
dotDisplay base g = let dotfile = base ++ ".dot" in
  do writeFile dotfile $ gx g
     system (dotty_program ++ base ++ ".dot")
        where gx g = graphviz g base (0,0) (1,1) Portrait

--        where gx g = graphviz g base (8.5,11) (1,1) Landscape

dotDisplay' ::(Show a,Show b)=>Gr a b -> IO ExitCode
dotDisplay' g = dotDisplay "temp" g
```

### 2.3 グラフの例

```
g01::Gr String Char
g01=mkGraph [(1,"x")] [(1,1,'b')]
```

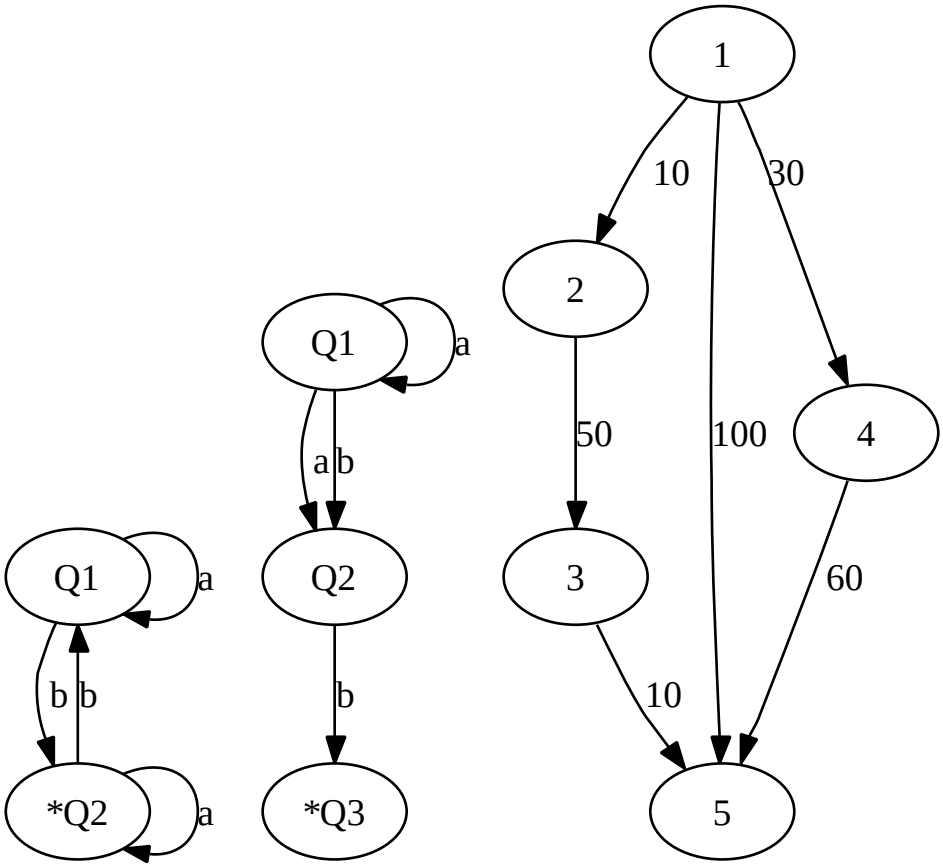


図 1: m1, m2, d01

```
m01::Gr String Char
m01=mkGraph [(1,"Q1"),(2,"*Q2")] [(1,1,'a'),(1,2,'b'),(2,1,'b'),(2,2,'a')]

m02::Gr String String
m02=mkGraph [(1,"Q1"),(2,"Q2"),(3,"*Q3")] [(1,1,"a"),(1,2,"a"),(1,2,"b"),(2,3,"b")]

d01::Gr () Integer
d01=mkGraph [(i,())|i<-[1..5]] [(1,2,10),(1,5,100),(1,4,30),(2,3,50),(3,5,10),(4,5,60)]

d01sp = sp 1 5 d01
d01splen = spLength 1 5 d01
```

**例 1** `D:\> dotDisplay "m01" m01`

`D:\> dotDisplay "m02" m02`

`D:\> dotDisplay "d01" d01`

### 2.4 K グラフ

```
kgraph::Int->Gr () ()
kgraph n =mkUGraph [1..n] [(((i-1) `mod` n) + 1, (i `mod` n)+1)|i<-[1..n]]

kngraph::Int->Int->Gr () ()
kngraph n m = mkUGraph [1..n] $ concat [ e j | j<-[1..m]]
                  where e j = [(((i-j) `mod` n)+1, (i `mod` n)+1)|i<-[1..n]]
```

### 2.5 ダイクストラのアルゴリズム

```
h01::Heap Integer (LPath Integer)
h01=Data.Graph.Inductive.Internal.Heap.unit 0 (LP [(0,0)])
```

未完成

### 2.6 gmap の利用例

```
c5::Gr () ()
c5=ucycle 5

ct::Context () () -> Context Int ()
ct (a0,n,_,a1) = (a0,n,100+n,a1)

c5'= gmap ct c5
```

## 3 量子回路のグラフ

未完成

```
t0=[([1,2],"cnot"),([2,3],"cnot"),([1,3],"cnot")]

-- circuit0::[Int]->[Int]->Int->Int->[Int]
-- circuit0 a [] x p = a
-- circuit0 [] _ x p = []
-- circuit0 (ax:at) b@(bx:bt) x p | p  == bx  = (x:(circuit0 at bt x))
--                                | otherwise = (ax:(circuit0 at b x))

-- t1=circuit0 [1,2,3,4,5] [2,3] 6
-- t2=circuit1 [1,2,3,4,5] [2,3] 6

-- circuit1::[Int]->[Int]->Int->[(Int,Int)]
-- circuit1 a b x = filter f s
--                  where  f (x,y) = not (x == y)
--                         s  = zip a a'
--                         a' = circuit0 a b x
```