ネットワークプログラミング

Jacques Garrigue, 2004 年 12 月 21 日

3　同期のモデル ペトリ・ネット

ネットワーク・プログラムを正しく書くには，通信と実行の関係を理解しなければならない．単純なプロトコルでも，相手が期待している答えを送ってくれないと動きません．

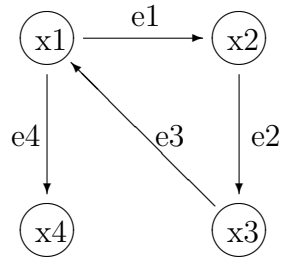
グラフ理論から来ているペトリネットがその関係を分かりやすく表す．

3.1　グラフとネット

定義 1 グラフ $G = (V, E, \gamma)$ は頂点の集合 V，辺の集合 E，そして辺の端を与える写像 $\gamma : E \to V \times V$ から定義される．

辺に方向を認めるかどうかによって，有向グラフと無向グラフが区別される．以下は有向グラフの例

- $V = x_1, x_2, x_3, x_4$
- $E = e_1, e_2, e_3, e_4$
- $\gamma(e_1) = (x_1, x_2) \quad \gamma(e_2) = (x_2, x_3) \quad \gamma(e_3) = (x_3, x_1) \quad \gamma(e_4) = (x_1, x_4)$

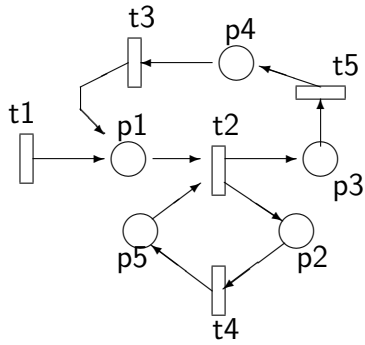


二種類の頂点を持つグラフを二部グラフという．二部グラフでは辺は異なる種類の頂点の間にしかない．

定義 2 ペトリネット $PN = (P, T, I, O)$ は以下の集合と関数から定義される二部グラフである.

- $P = \{p_1, \ldots, p_n\}$ は場所の有限集合
- $T = \{t_1, \ldots, t_m\}$ は遷移の有限集合
- $I : P \times T \to \{0, 1\}$ は各場所から各遷移への辺の有無を表す
- $O : T \times P \to \{0, 1\}$ は各遷移から各場所への辺の有無を表す

グラフを描くとき，場所は円として，遷移は棒として表す.



- $P = \{p_1, p_2, p_3, p_4, p_5\}$
- $T = \{t_1, t_2, t_3, t_4, t_5\}$
- $I(t_1) = \emptyset,\ I(t_2) = \{p_1, p_5\},\ I(t_3) = \{p_4\},\ I(t_4) = \{p_2\},\ I(t_5) = \{p_3\}$
- $O(t_1) = \{p_1\},\ O(t_2) = \{p_2, p_3\},\ O(t_3) = \{p_1\},\ O(t_4) = \{p_5\},$
  $O(t_5) = \{p_4\}$

純粋なペトリネットは自己ループを含まない．自己ループとは，直接に同じ場所に戻る遷移のことをいう．即ち，純粋なペトリネットでは $I(p, t) = 1$ かつ $O(t, p) = 1$ となるような p と t は存在しない.

純粋なペトリネットだったら，全ての辺を以下の接続行列 C によって定義できる．C は $n \times m$ の整数行列で

$$C_{ij} = O(t_j, p_i) - I(p_i, t_j).$$

逆に，I と O は以下のように定義できる．

$$I(p_i, t_j) = \mathsf{max}(-C_{ij}, 0) \qquad O(t_j, p_i) = \mathsf{max}(C_{ij}, 0)$$

3.2 ペトリネットのマーク付けと実行

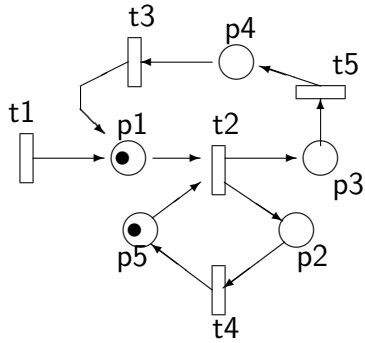
ペトリネットで計算をモデル化するために，動的に変化する**マーク付け**を導入する．

あるペトリネット PN のマーク付け $M : P \to \mathbf{N}$ は各場所に対して，そこに置かれるマークの数を表している．描くときには，場所の中の黒い丸として表現される．マーク付け全体を自然数のベクターとして表現することもできる．

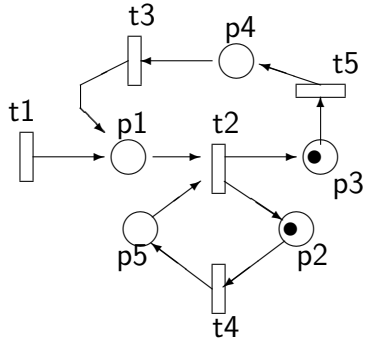
前の例の中にマークはなかったので，$M_0 = [0\ 0\ 0\ 0\ 0]$.

ペトリネットの実行は遷移の**発火**によって行われる．ある遷移 t の入力場所 $I(t)$ の全てに一個以上のマークがあると，その遷移が発火可能になる．発火されると，入力場所のマークを一個ずつ減らし，出力場所 $O(t)$ のマークを一個ずつ増やす．

例えば，前の例にマーク付け $M_1 = [1\ 0\ 0\ 0\ 1]$ を与える．そうすると遷移 t_2 が発火可能になる．



t_2 を発火すると，新しいマーク付け $M_2 = [0\ 1\ 1\ 0\ 0]$ になる．



複数の遷移を特定の順番で発火する発火列を以下のように書く．

$$s = t_{i_1} \bullet t_{i_2} \bullet \ldots \bullet t_{i_s}$$

その中の遷移の発火回数を分けて数えると，長さ m のベクター N_s ができる．新しいマーク付けは以下の式によって得ることができる．

$$M' = M + C \cdot N_s$$

あるマーク付けから連続発火でたどりつけるマーク付けの集合を**到達可能マーク付け**という．

3.3 ペトリネットの性質

あるマーク付けの到達可能集合が有限ならば，そのマーク付けは**有界**である．あるペトリネットの任意のマーク付けが有界ならば，そのペトリネットは**構造的に有界**という．

あるペトリネット PN がある重みベクター (w_i) を**保存**するというのは，遷移がマーク付けの重さを変えないことである．

$$\sum_{p_i \in P} w_i M(p_i) = \sum_{p_i \in P} w_i M_0(p_i)$$

ここでは各 w_i は正の有理数である．保存性から有界性は導ける．行列表現を使い，重みベクターが以下の方程式を満すものである．

$$\forall N_S \; w^T \cdot M = w^T \cdot M' = w^T \cdot (M + C \cdot N_S) \Leftrightarrow C^T \cdot w = 0$$

即ち w は接続行列の転置行列の核に入っている．

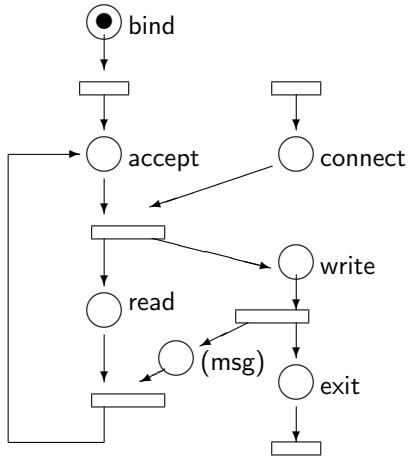
あるマーク付けから到達でききる全てのマーク付けから，任意の遷移 t を含む発火列が存在すれば，そのマーク付けが **live**(生きている) だという．同様に，ペトリネットは構造的に live だという．live でなければ，deadlock している遷移がある．どんな発火列でもその遷移が発火されない．

ある場所から複数の遷移に行く辺があり，それらが自己ループでなければ，**競合**を起こす可能性がある．二つの遷移が競合するというのは，あるマーク付けでは両方とも発火可能だあるが，どちらかを発火させると，もう一方が発火できなくなる．

競合が起きていなくても，発火列によって競合状態になれるとき，そのペトリネットに**曖昧性**がある．

特に live 性以降の諸問題は並列・分散システムに付いて考えなければならないものである．

例　TCP のクライアントとサーバ



実習

先週のネットじゃんけんを完成する．できれば，DNS を利用できるようにする．

- `struct hostent* gethostbyname(const char *name);`
  DNS(ネームサービス) を利用してホスト名に対する IP アドレスを調べる．

```
#include <netdb.h>

/* ... */

  struct hostent *host;

  host = gethostbyname("name");
  memcpy((char*)&sin.sin_addr, host->h_addr, host->h_length)
  sin.sin_family = host.h_addrtype;
```