双方向回路 QBC による認知機能モデルのプロトタイピング

脳型人工知能研究チーム
ミーティング資料

2016-10-27

産業技術総合研究所

一杉裕志

背景

- 大脳皮質ベイジアンネットモデルのデモを作って世の中に理解を広めたい。小規模でも動いているものがあることが大事。
- 機械学習の実装が得意な人材は世界的に不足しているので、そうでない人も容易に参加できる研究テーマを提供したい。

QBC とは

- QBC: Qualitative Bidirectional Circuit, 定性的双方向回路
- 大脳皮質ベイジアンネット仮説にもとづいた認知機能モデリングのためのプロトタイピングの道具。
  - モデルの生産性の高さ、適度な表現の自由度、モデル上の推論の効率を重視して設計。
- 飛行機の風洞実験に使う模型を作るための**粘土**に相当。

QBCは認知機能モデリングのための粘土

QBCを使う利点

- ネットワークの状態を最適化するというベイジアンネットの特性は残しつつも、記号処理に近い動作をする。
  - **大規模機械学習特有の黒魔術的困難さを回避**。
  - モデルが意図した通りに動かないときの原因の究明が容易。
- モデルの設計・記述が容易。
  - 変数の値に意味のある文字列が使えるので可読性が高い。

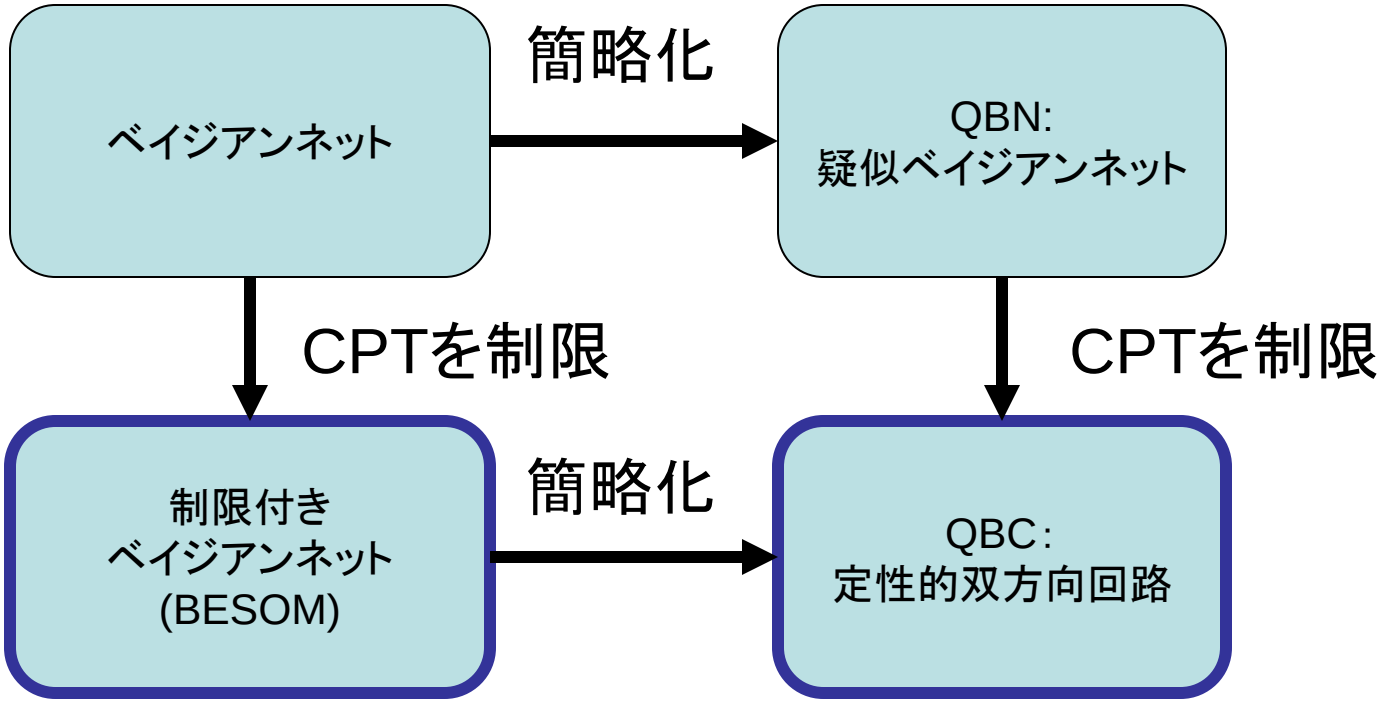
過去の認知機能モデリングの道具

- 計算のモデル
  - チューリングマシン、ラムダ計算、・・・
- ニューラルネットワークモデル
  - パターン認識のモデル：パーセプトロン
  - 連想記憶のモデル：ホップフィールドネットワーク
  - 時系列学習のモデル：エルマンネット
- 形式論理
  - 述語論理、様相論理、prolog、・・・
- プロダクションシステム
  - ACT-R 、・・・

QBNとQBC

- 現在の実装では QBC は QBN に変換される。
- QBN(Quasi Bayesian network, 疑似ベイジアンネット)
  - 0と非0の値のみを区別する簡略化したベイジアンネット。
- QBC(定性的双方向回路)
  - QBN のCPT（条件付確率表）に制約を加えたもの。
  - QBN よりも記述しやすい。
  - 学習機能はない。CPTを手で与える。
  - 将来、時系列処理等の拡張を加える予定。

QBC の位置づけ

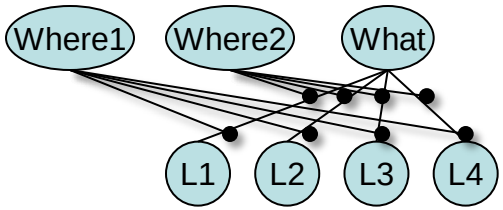


学習可
高速な近似推論
状態の可視化は困難

学習不可、手でCPTを記述
指数時間で厳密推論
状態の解釈が容易

QBCの記述例とQBNへの変換例

```
List<TableNode> tableNodeList = new ArrayList<>();
List<GateMatrix> gateMatrixes = new ArrayList<>();
tableNodeList.add(tableNode(What, table(
        row(V1,A,A,A,A),
        row(V2,B,B,B,B),
        row(V3,C,C,C,C)
        )));
tableNodeList.add(tableNode(Where1, table(
        row(V1,I,O,I,O),
        row(V2,O,I,O,I)
        )));
tableNodeList.add(tableNode(Where2, table(
        row(V1,I,I,O,O),
        row(V2,O,O,I,I)
        )));
tableNodeList.add(tableNode(L1, table(
        row(A),row(B),row(C))));
tableNodeList.add(tableNode(L2, table(
        row(A),row(B),row(C))));
tableNodeList.add(tableNode(L3, table(
        row(A),row(B),row(C))));
tableNodeList.add(tableNode(L4, table(
        row(A),row(B),row(C))));
gateMatrixes.add(new GateMatrix(
        list(Where1,Where2), list(What), list(L1,L2,L3,L4)
        ));
```

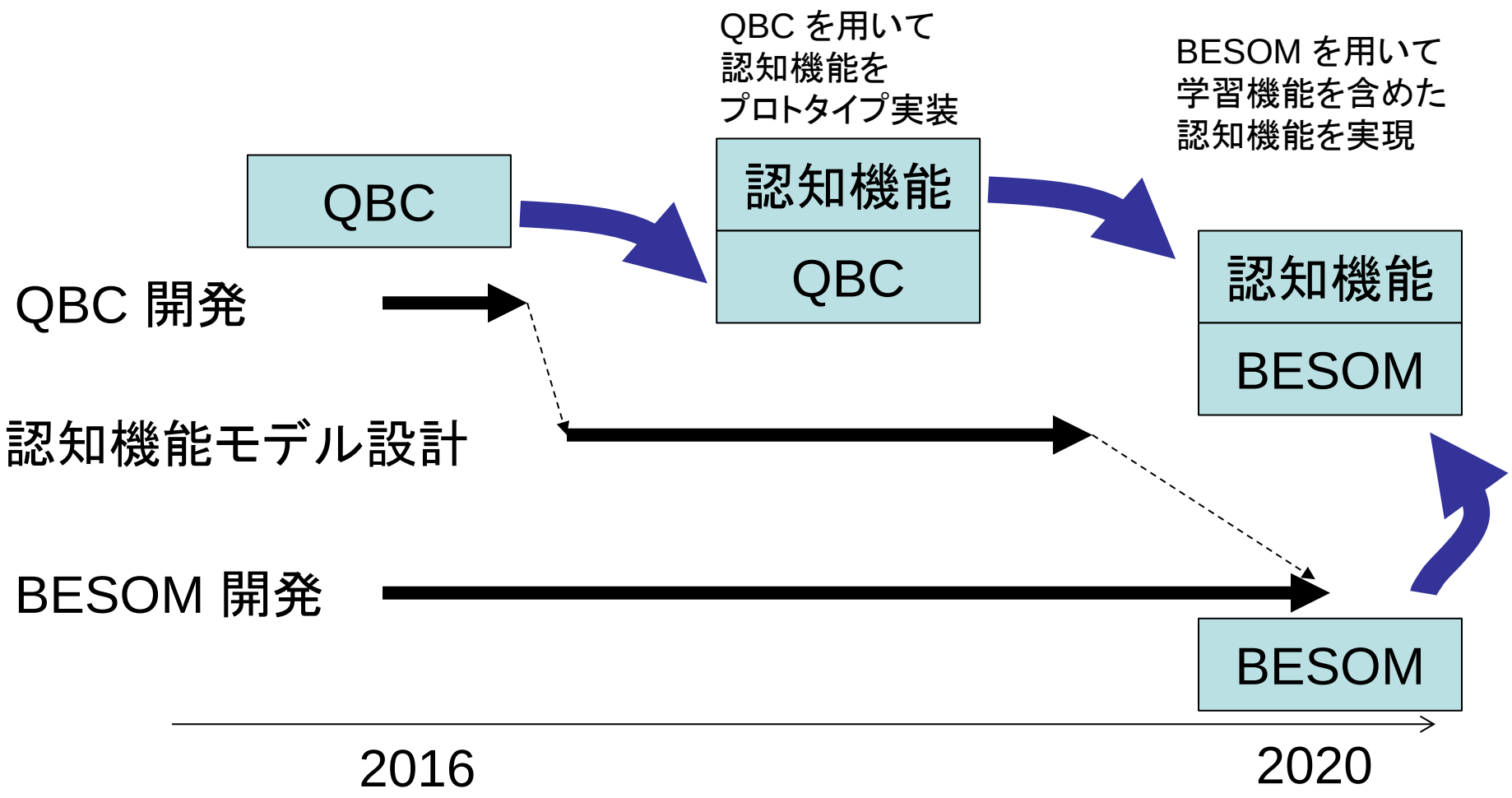


変換

```
netData = { {
    {L4, What->L4, },
    {A, V1, },
    {B, V2, },
    {C, V3, },
    {0, 0, },
  }, {
    {What, },
    {V1, },
    {V2, },
    {V3, },
    {0, },
  }, {
    {L1, What->L1, },
    {A, V1, },
    {B, V2, },
    {C, V3, },
    {0, 0, },
  }, {
    {L3, What->L3, },
    {A, V1, },
    {B, V2, },
    {C, V3, },
    {0, 0, },
  }, {
    {Where2, },
    {V1, },
    {V2, },
    {0, },
  }, {
    {Where1, },
    {V1, },
    {V2, },
    {0, },
  }, {
```

```
{L2, What->L2, },
    {A, V1, },
    {B, V2, },
    {C, V3, },
    {0, 0, },
  }, {
    {What->L1, Where1, Where2, What, },
    {0, V1, __, __, },
    {0, __, V1, __, },
    {V1, V2, V2, V1, },
    {V2, V2, V2, V2, },
    {V3, V2, V2, V3, },
  }, {
    {What->L2, Where1, Where2, What, },
    {0, V2, __, __, },
    {0, __, V1, __, },
    {V1, V1, V2, V1, },
    {V2, V1, V2, V2, },
    {V3, V1, V2, V3, },
  }, {
    {What->L3, Where1, Where2, What, },
    {0, V1, __, __, },
    {0, __, V2, __, },
    {V1, V2, V1, V1, },
    {V2, V2, V1, V2, },
    {V3, V2, V1, V3, },
  }, {
    {What->L4, Where1, Where2, What, },
    {0, V2, __, __, },
    {0, __, V2, __, },
    {V1, V1, V1, V1, },
    {V2, V1, V1, V2, },
    {V3, V1, V1, V3, },
  }, };
```

QBC と BESOM のロードマップ



BESOM 開発と並行して認知機能モデルを設計、その後統合

QBC で記述する予定の認知機能

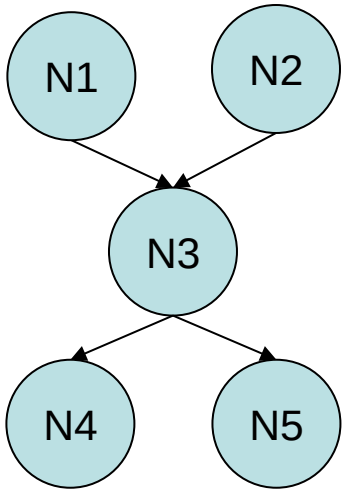
- 視覚野
  - 腹側経路と背側経路、グリッド細胞
  - オクルージョン
- 言語野
  - CYKパーザ
  - ユニフィケーションの機構と記号推論
  - CCG、単語列から深層格表現への変換
- 前頭前野
  - 視覚バインディング（オブジェクトファイル）
  - 階層的時系列処理

疑似ベイジアンネット(QBN)

疑似ベイジアンネット(QBN)

ベイジアンネットとは、確率変数の同時確率を
条件付確率の積に分解したもの。
例:
P(N1,N2,N3,N4,N5)
=P(N4|N3)P(N5|N3)P(N3|N1,N2)P(N1)P(N2)

・QBNは、ベイジアンネットの確率値の
0と非0のみを区別するように簡略化したもの。
・同時確率を分解した条件付確率が
すべて非0のときのみ、同時確率が非0になる。

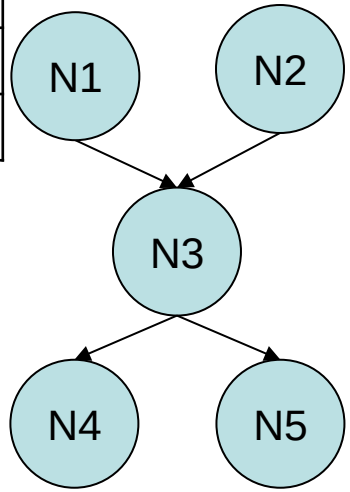


疑似ベイジアンネットの定義例

```
netData = { {
   {N1, },
   {False, },
   {True, },
 }, {
   {N2, },
   {False, },
   {True, },
 }, {
   {N3, N1, N2, },
   {False, False, False, },
   {True, True, False, },
   {True, False, True, },
   {True, True, True, },
 }, {
   {N4, N3, },
   {False, False, },
   {True, True, },
 }, {
   {N5, N3, },
   {True, True, },
   {False, False, },
 }, };
```

{子ノード名、親ノード名１、親ノード名２、・・・} と、
条件付確率が非0になる親子の値の組を列挙

N3	N1	N2	P(N3|N1,N2)
False	False	False	non-zero
True	True	False	non-zero
True	False	True	non-zero
True	True	True	non-zero
	これら以外		zero



疑似ベイジアンネットの解

```
netData = { {
   {N1, },
   {False, },
   {True, },
 }, {
   {N2, },
   {False, },
   {True, },
 }, {
   {N3, N1, N2, },
   {False, False, False, },
   {True, True, False, },
   {True, False, True, },
   {True, True, True, },
 }, {
   {N4, N3, },
   {False, False, },
   {True, True, },
 }, {
   {N5, N3, },
   {True, True, },
   {False, False, },
 }, };
```

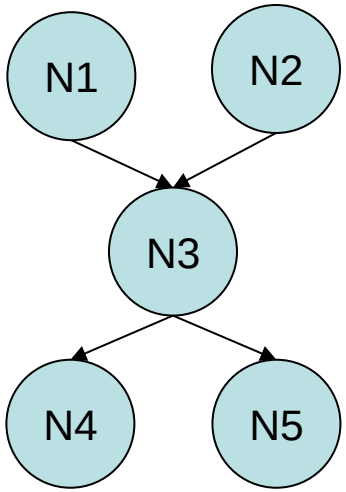
すべての解
（同時確率が
非0になる値の
組）

```
result 1
N5 : False
N4 : False
N3 : False
N2 : False
N1 : False

result 2
N5 : True
N4 : True
N3 : True
N2 : True
N1 : False

result 3
N5 : True
N4 : True
N3 : True
N2 : False
N1 : True

result 4
N5 : True
N4 : True
N3 : True
N2 : True
N1 : True
```

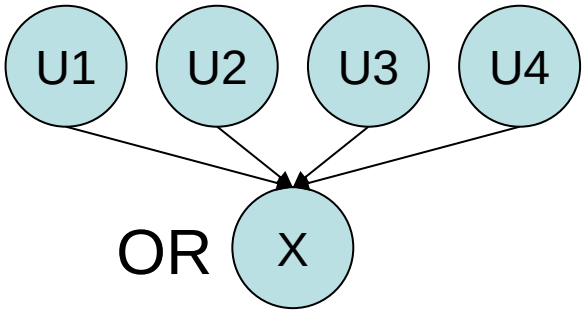


noisy-OR と noisy-AND

noisy-OR モデル [Pearl 1988]

- すべて2値変数。
- 親ノードの値が0のときは子ノードに影響を与えない。
- 親ノード Uk のみが1なら子ノードは確率 wk で1。
- 重みが十分小さいか、高々1つの親ノードのみ1の場合は線形和と一致。$P(X=1|u_1,\ldots,u_m) \approx \Sigma_k u_k w_k$

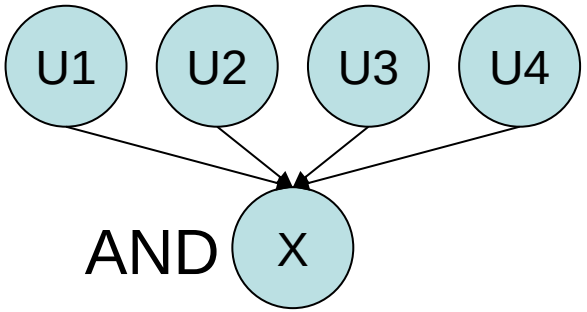
$$P(X=0|u_1,\ldots,u_m) = \Pi_{k=1}^{m}(1-w_k)^{u_k}$$
$$P(X=1|u_1,\ldots,u_m) = 1-\Pi_{k=1}^{m}(1-w_k)^{u_k}$$

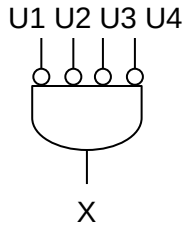


noisy-AND モデル

- 標準的な定義があるのか不明だが、例えば下記の式ように定義できる:
  - noisy-OR の X=0 と X=1 を入れ替えたもの
  - 親ノード Uk のみが1なら子ノードは確率 1 - wk で1
- この定義だとすべての親ノードの値が0のとき子ノードは1。wk は抑制性結合のように振る舞う。

$$P(X=0|u_1,\ldots,u_m) = 1 - \Pi_{k=1}^{m}(1-w_k)^{u_k}$$
$$P(X=1|u_1,\ldots,u_m) = \Pi_{k=1}^{m}(1-w_k)^{u_k}$$





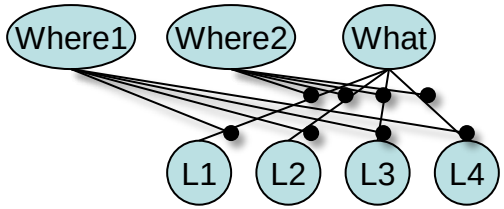
入力が負論理の
ANDゲート

QBC

QBCとは

- CPTを制限したベイジアンネットを、簡略化して定性的にしたものがQBC。
- 使用する3種類のノード:
  - noisy-OR
  - noisy-AND
  - 排他性制約ノード
- QBCのネットワークを図に書くときは、独自の記法を使うことにする。

例

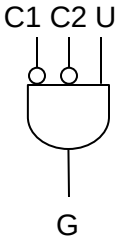


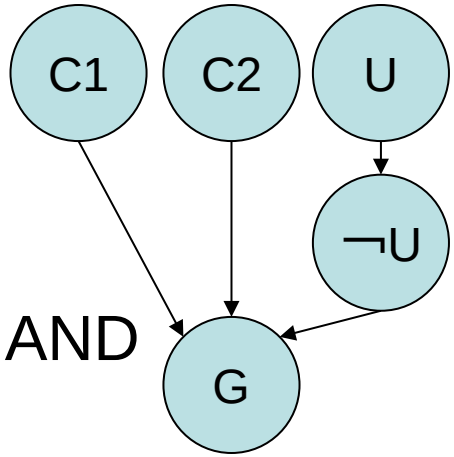
QBC設計の指導原理

- さまざまな認知機能が簡潔に書けること
- 脳との対応が付きやすいこと
- 学習しやすいこと
  - 発火のスパース性、重みのスパース性など
- 脳における耐故障性が説明できること
  - 細胞死、断線、エネルギー・酸素不足による発火頻度の低下等への耐性

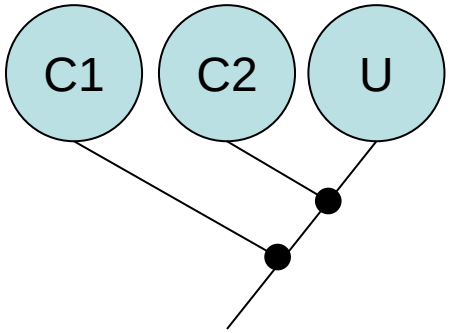
- 注：現在の設計が唯一の正解とは限らない。

QBCのANDゲートの定義





QBCでの記法

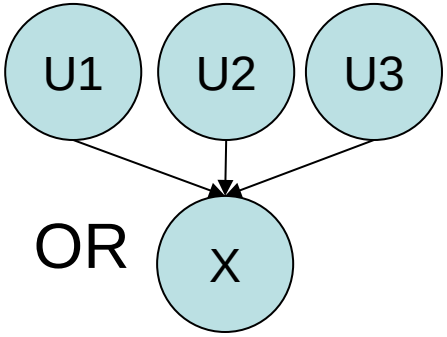


G	C1	C2	U	P(G | C1,C2,U)
0	1	_	_	non-zero
0	_	1	_	non-zero
0	_	_	0	non-zero
1	0	0	1	non-zero
これら以外				zero

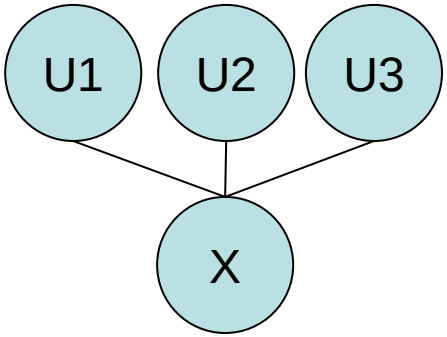
G に入力するすべての信号が
0のときに限り G の出力は1
（となる確率が non-zero）

注："_" は、ワイルドカード。
0でも1でもよい、という意味。

QBCのORゲートの定義



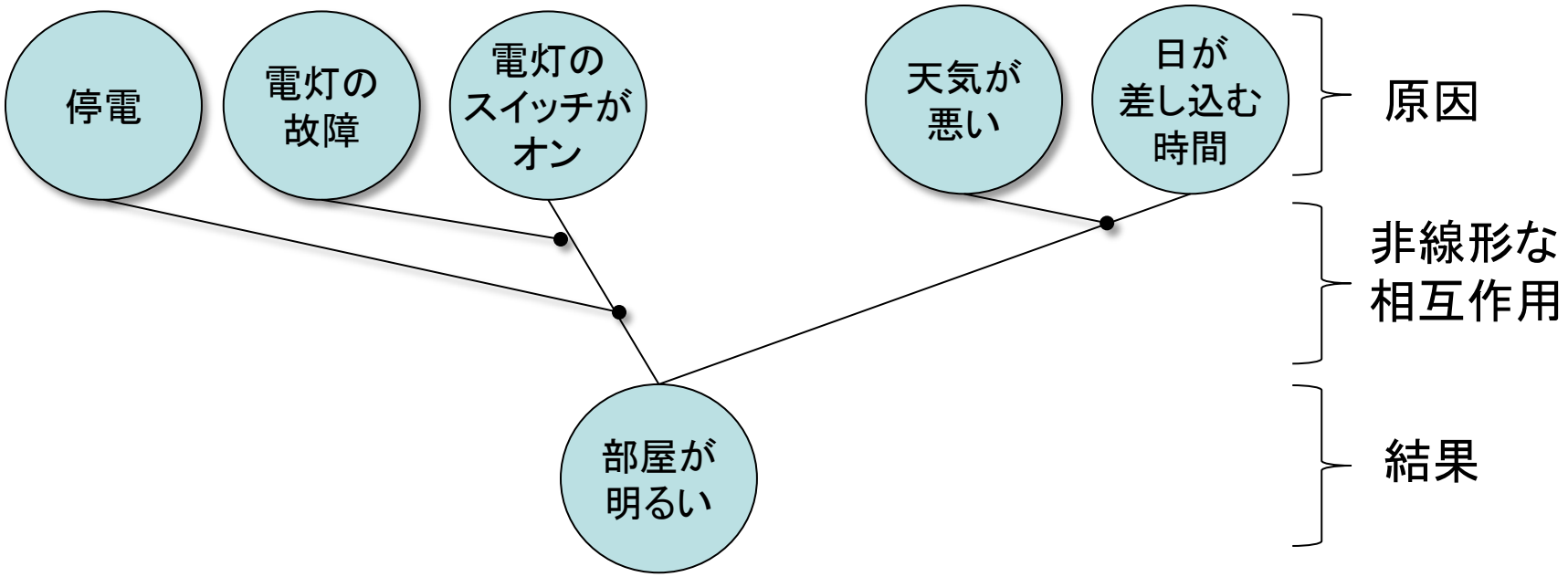
QBCでの記法



X	U1	U2	U3	P(X | U1,U2,U3)
0	0	0	0	non-zero
1	1	_	_	non-zero
1	_	1	_	non-zero
1	_	_	1	non-zero
これら以外				zero

X に入力するどれかの信号が
1のときに限り X の出力は1
（となる確率が non-zero）

簡単なQBCの例



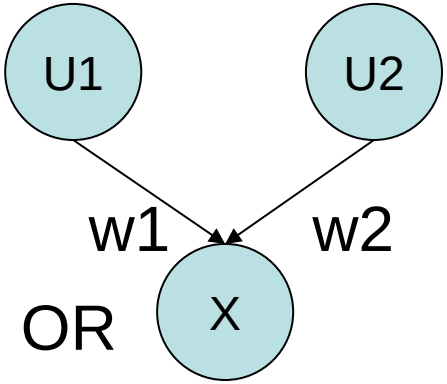
推論の例:

・夜、電灯のスイッチがオンなのに暗い。
　→ 停電または電灯の故障。

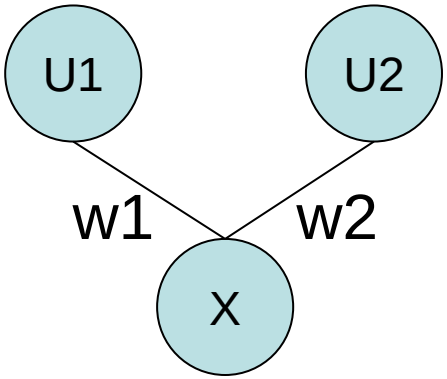
・停電なのに明るい。
　→ 日が差し込む時間で天気が良い。

自然界によくある
因果関係が
簡潔に表現可能

ORゲートの学習則（予想）

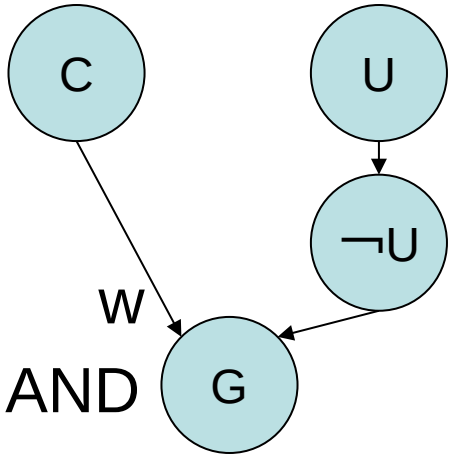


QBCでの記法



X	U1	U2	⊿w1	⊿w2
–	0	0	0	0
0	0	1	0	minus
0	1	0	minus	0
0	1	1	minus	minus
1	0	1	0	plus
1	1	0	plus	0
1	1	1	plus	plus

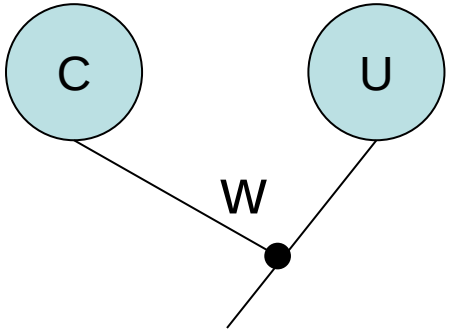
ANDゲートの学習則（予想）



NOT

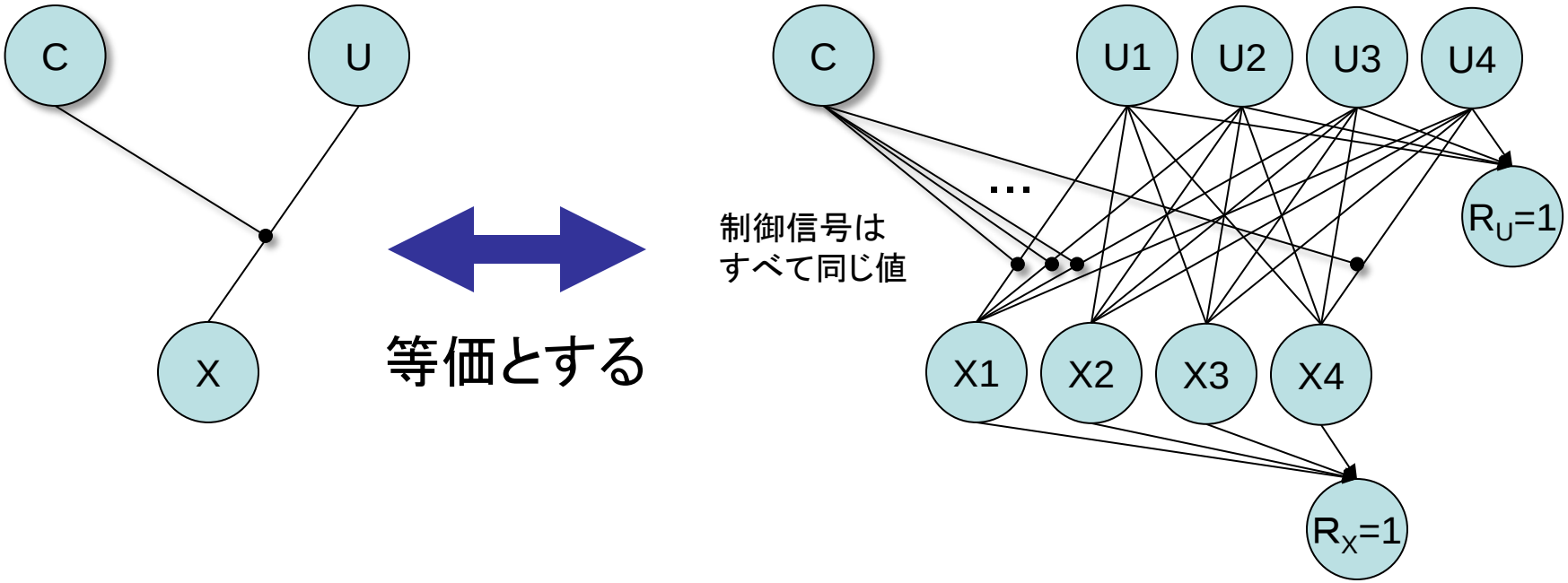
C	U	G	⊿w
0	–	–	0
1	0	0	minus
1	1	0	plus
1	0	1	plus
1	1	1	minus

QBCでの記法



QBCを多値に拡張

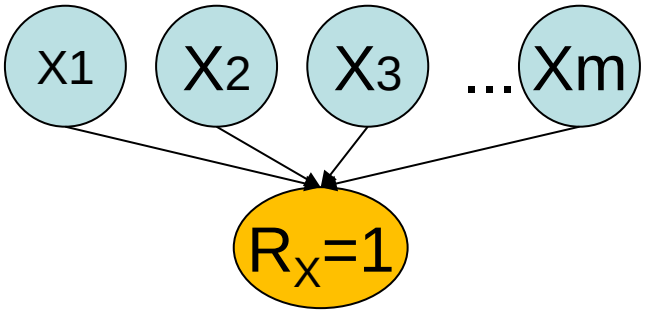
多値ノードの振る舞いは、2値ノードと排他性制約ノードを使って定義することにする。



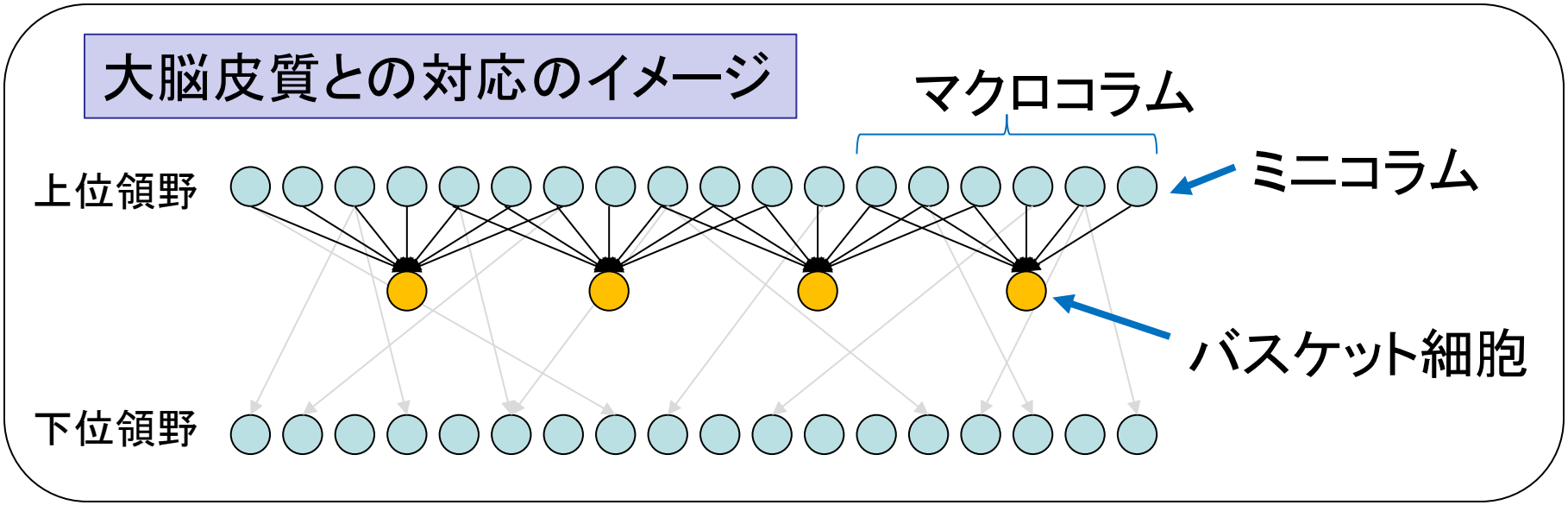
C は2値(0,1)、
U, Xは5値(0,1,2,3,4)

C, Ui, Xi, R_U, R_X は2値

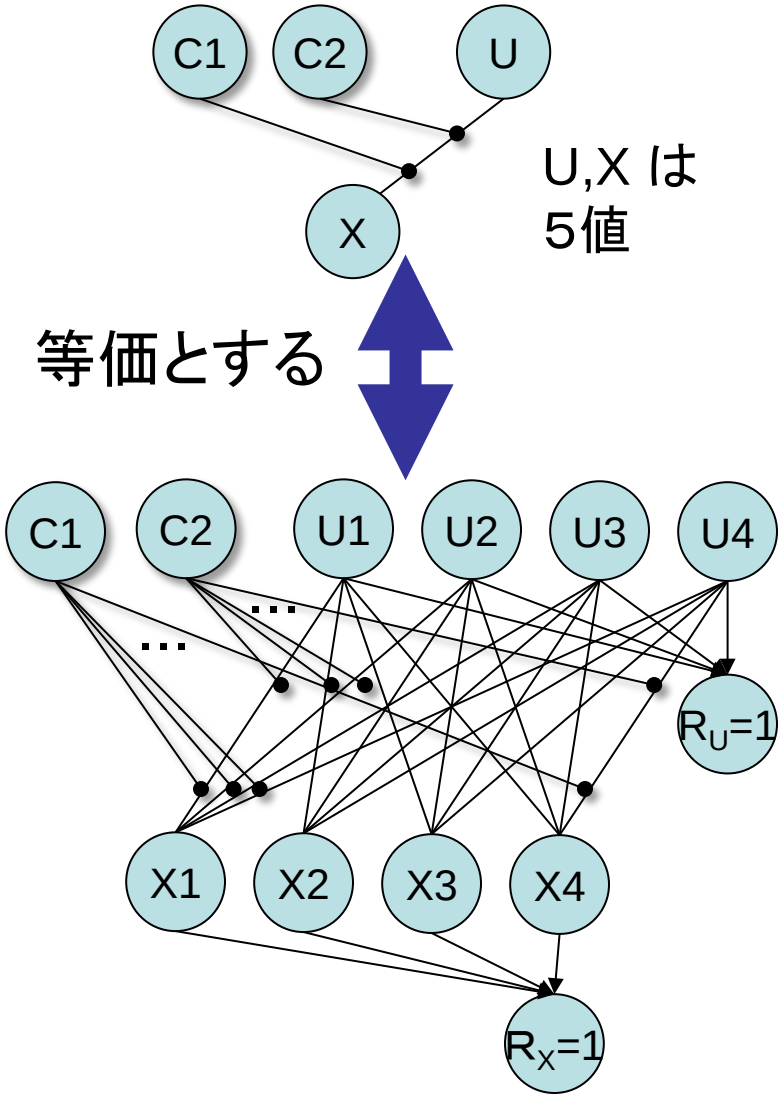
排他性制約ノード



$P(R_X = 1|X_1, \ldots, X_m) = 0$ (Xk のうち2つ以上が1の場合)
$P(R_X = 1|X_1, \ldots, X_m) = 1$ (Xkのうち高々1つが1の場合)



多値ANDゲートの定性的CPT

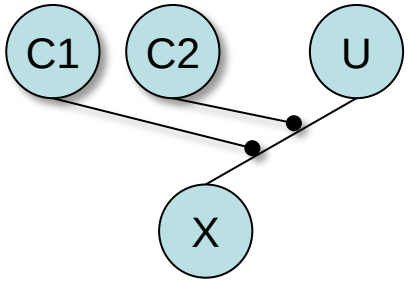


X	C1	C2	U	P(X | C1,C2,U)
0	1	_	_	non-zero
0	_	1	_	non-zero
A	0	0	gen(A)	non-zero
	これら以外			zero

注："_" は任意の値、
gen(A) は X=A を生成する
U の値の集合

$$u \in gen(x) \ \ iff \ \ P(X = x|U = u) > 0$$

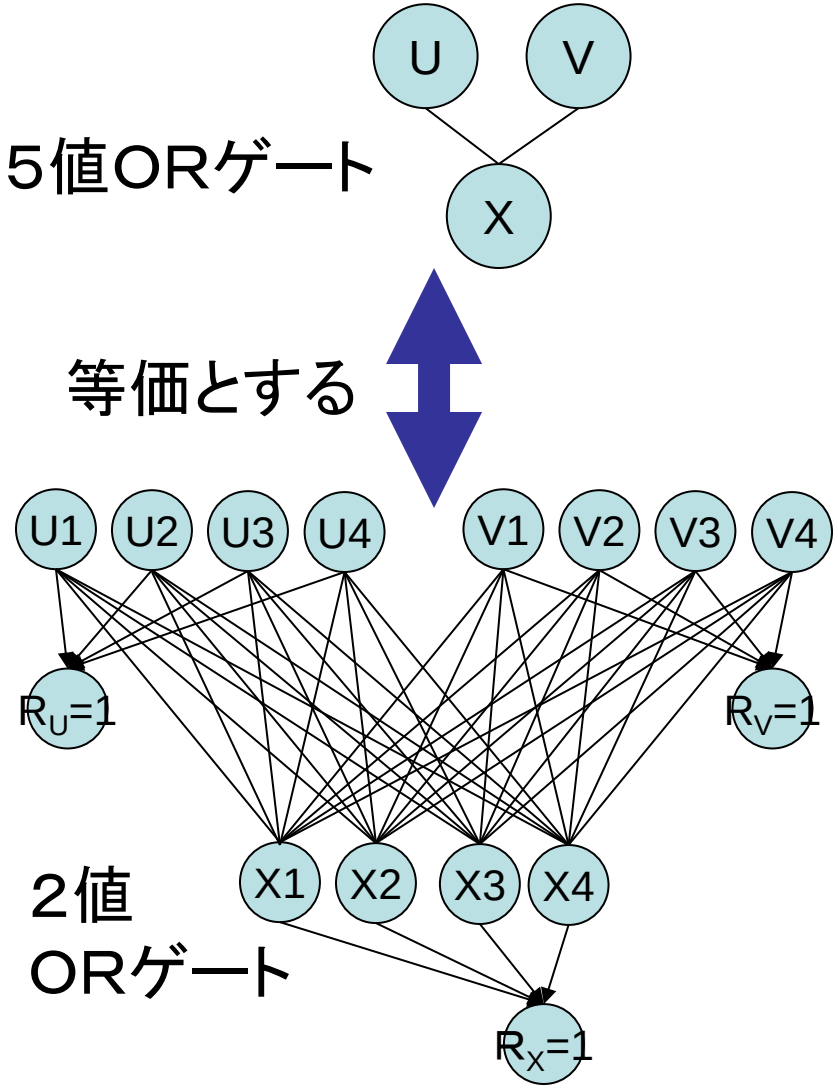
制御ノードも多値の場合の定性的CPT



X	C1	C2	U	P(X | C1,C2,U)
0	gen(I)	_	_	non-zero
0	_	gen(I)	_	non-zero
A	gen(O)	gen(O)	gen(A)	non-zero
これら以外				zero

gen(I), gen(O) の I, O は
Inhibit, Open の略。
gen(I) はANDゲートを抑制する値、
gen(O) は抑制しない値。

多値ORゲートのCPT



X	U	V	P(X | U1,U2,U3)
0	0	0	non-zero
A	gen(A)	0	non-zero
A	0	gen(A)	non-zero
A	gen(A)	gen(A)	non-zero
これら以外			zero

QBC の記述言語

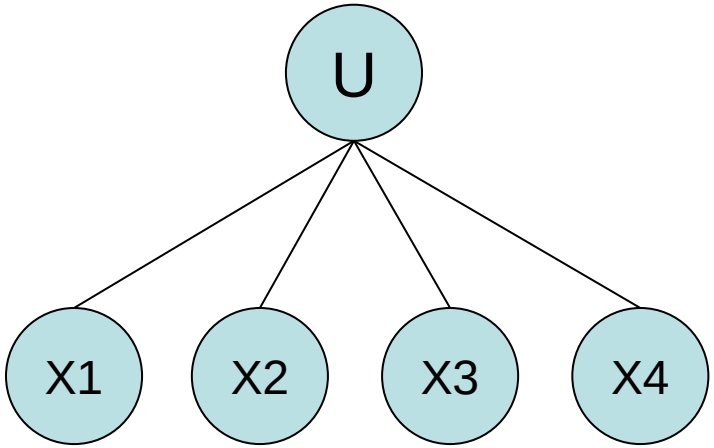
QBC の記述言語

- QBC をCPTの形で素朴に記述するとパラメタ数が爆発し煩雑。
  → **書きやすいDSL（domain-specific language, ドメイン固有言語）を提供。**
- DSLにおける QBC の構成要素：
  - テーブルノード
    - ノードをCPTではなく「受容野」で定義する。
  - ゲート行列
    - テーブルノード間の結合を定義する。

QBC記述のためのDSL

- Java のソースコード中で定義。
  - 利点 :
    - Eclipse で変数名・値のスペルチェック、文法・型チェック等がある程度可能。
    - QBCを Java プログラムで生成することも容易。
- QBC の構成要素 :
  - テーブルノード :
    tableNode(N, row(V1,A,B,...),row(V2,C,D,...),...)
  - ゲート行列:
    gateMatrix(list(C1,...),list(U1,...),list(X1,...))

テーブルノード(Table Node)



各ユニットの「受容野」を
テーブルの形で定義する。

U	X1	X2	X3	X4
u1	a	b	c	d
u2	e	f	g	h
u3	...			
...				

tableNode(***U, table(***
row(***U1,A,B,C,D),***
row(***U2,E,F,G,H),***
row(***U3,...),***
...)));

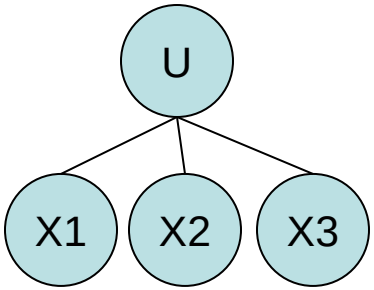
U の値とその時に起きうる子ノードの値の組を定義する。
子ノードがANDゲートの場合は、子ノードの値にはIかOを指定する。

テーブルの要素の or

- 子ノードの値が e1,e2,... のどれもよいときはテーブルの要素に or(e1,e2,...) と書く。
- 同じ行に複数の or(...) があるときは、すべての値の組み合わせが許される。
  - table(row(V,or(A,B),or(C,D))) は table(row(V,A,C),row(V,A,D),row(V,B,C),row(V,B,D)) と等価。
- CNNのプーリング層に似たものを表現可能。

特徴のプーリング

パーツの局所的な平行移動に対し不変な応答。
子ノードのユニットは特徴の位置の両方の情報を
持っていると仮定。




```
net.tableNodeList.add(tableNode(U, table(
  row(Vuvw,or(Vu0,V0u),or(Vv0,V0v),or(Vw0,V0w)),
  row(Vxyz,or(Vx0,V0x),or(Vy0,V0y),or(Vz0,V0z))
)));
net.tableNodeList.add(tableNode(X1, table(
  row(Vu0),row(V0u),row(Vx0),row(V0x)
)));
net.tableNodeList.add(tableNode(X2, table(
  row(Vv0),row(V0v),row(Vy0),row(V0y)
)));
net.tableNodeList.add(tableNode(X3, table(
  row(Vw0),row(V0w),row(Vz0),row(V0z)
)));
net.gateMatrixes.add(new GateMatrix(
  list(), list(U), list(X1,X2,X3)
));
```

すべての解

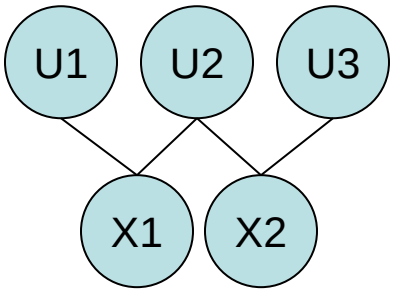
U, X1, X2, X3,
Vuvw, Vu0, Vv0, Vw0,
Vuvw, Vu0, V0v, Vw0,
Vuvw, Vu0, Vv0, V0w,
Vuvw, Vu0, V0v, V0w,
Vuvw, V0u, Vv0, Vw0,
Vuvw, V0u, V0v, Vw0,
Vuvw, V0u, Vv0, V0w,
Vuvw, V0u, V0v, V0w,
Vxyz, Vx0, Vy0, Vz0,
Vxyz, Vx0, V0y, Vz0,
Vxyz, Vx0, Vy0, V0z,
Vxyz, Vx0, V0y, V0z,
Vxyz, V0x, Vy0, Vz0,
Vxyz, V0x, V0y, Vz0,
Vxyz, V0x, Vy0, V0z,
Vxyz, V0x, V0y, V0z,

複数の親ノードと or

- ORゲートの各親ノードが or で複数の生成し得る値の集合を指定している場合、それらの**積集合**の要素が子ノードの値として許される。
- 次ページの実行例参照。

複数の親ノードの例

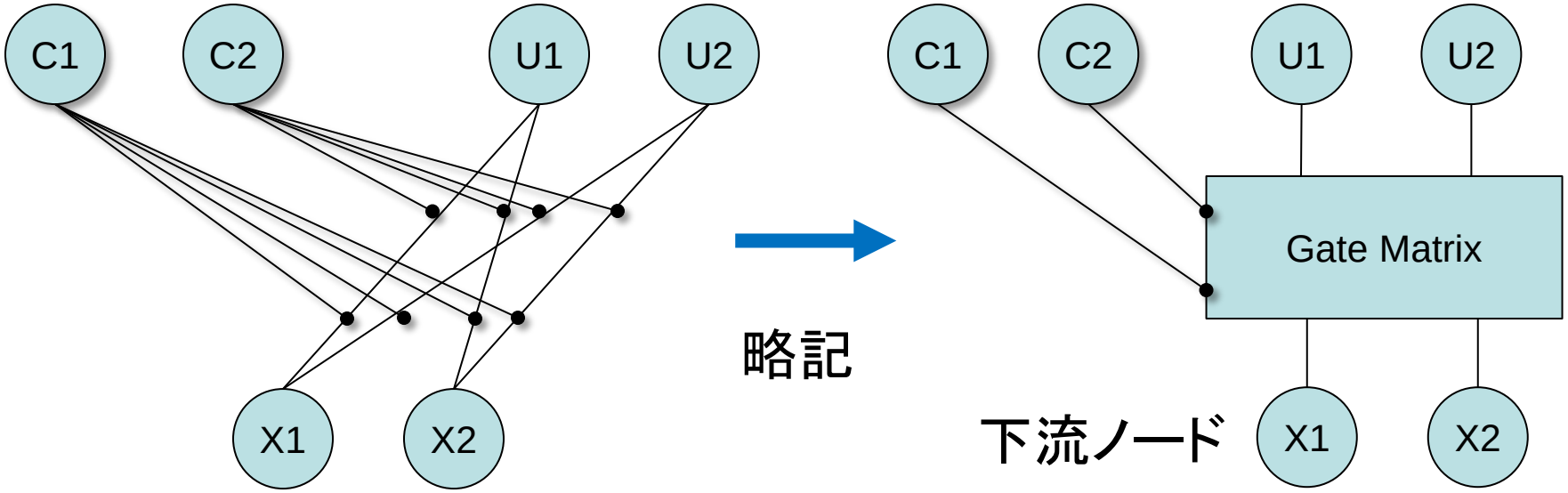
```
net.tableNodeList.add(tableNode(U1, table(
    // U1 の値が Vx, X1 の値は Vxy
    row(V1,or(V11,V12)),
    row(V2,or(V21,V22))
    )));
net.tableNodeList.add(tableNode(U2, table(
    // U2 の値が Vy, X1 の値は Vxy , X2 の値は Vyz
    row(V1,or(V11,V21),or(V11,V12)),
    row(V2,or(V12,V22),or(V21,V22))
    )));
net.tableNodeList.add(tableNode(U3, table(
    // U3 の値が Vz, X2 の値は Vyz
    row(V1,or(V11,V21)),
    row(V2,or(V12,V22))
    )));
net.tableNodeList.add(tableNode(X1, table(
    row(V11), row(V12), row(V21), row(V22))));
net.tableNodeList.add(tableNode(X2, table(
    row(V11), row(V12), row(V21), row(V22))));
net.gateMatrixes.add(new GateMatrix(
    list(), list(U1,U2), list(X1)
    ));
net.gateMatrixes.add(new GateMatrix(
    list(), list(U2,U3), list(X2)
    ));
```



すべての解

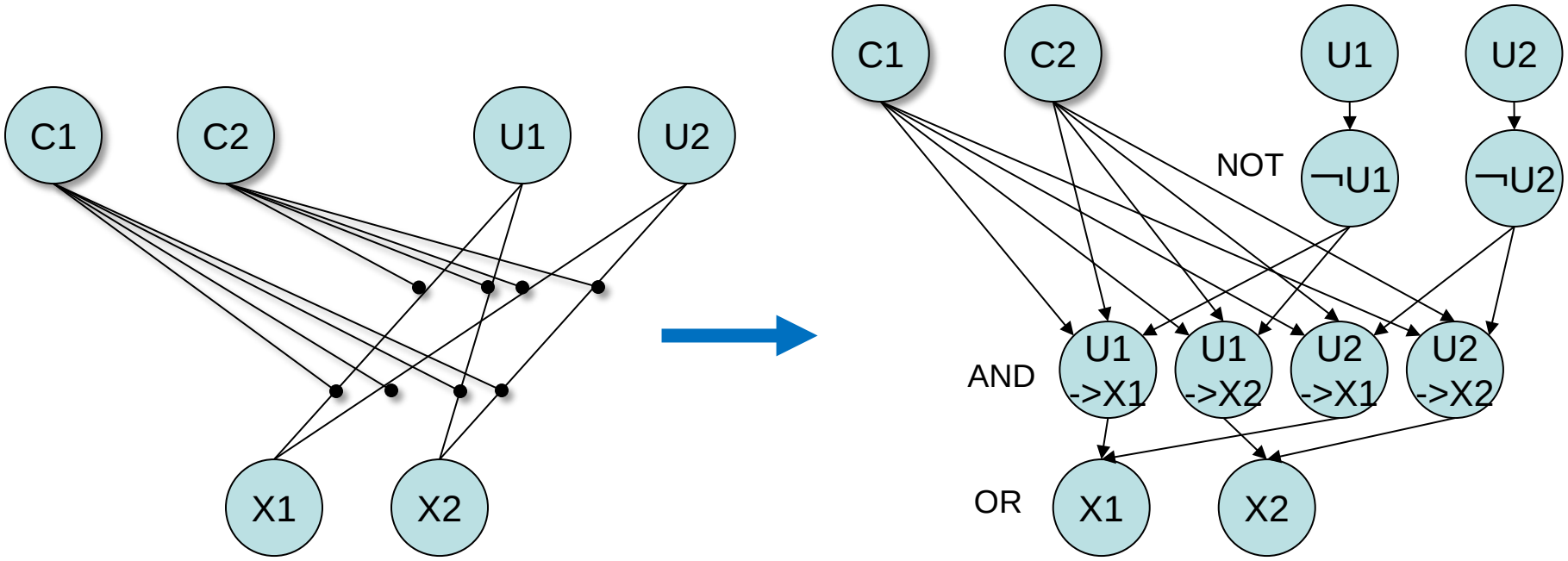
U1, U2, U3, X1, X2,
V1, V1, V1, V11, V11,
V1, V1, V2, V11, V12,
V1, V2, V1, V12, V21,
V1, V2, V2, V12, V22,
V2, V1, V1, V21, V11,
V2, V1, V2, V21, V12,
V2, V2, V1, V22, V21,
V2, V2, V2, V22, V22,

ゲート行列(Gate Matrix)




```
gateMatrix(
    list(C1,C2),
    list(U1,U2),
    list(X1,X2)
 ));
```

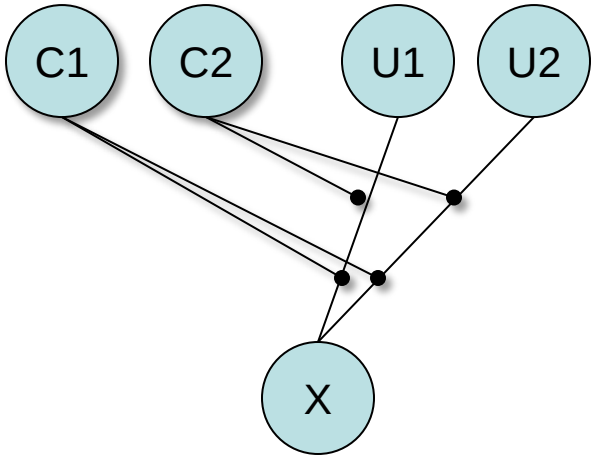
QBCは等価な疑似ベイジアンネットに変換可能



QBC

ベイジアンネット

真のベイジアンネットにおけるゲート行列



P(X|C1,C2,U1,U2)=
ΣG1,G2 P(X|G1,G2)P(G1|C1,C2,U1)P(G2|C1,C2,U2)

ただし

P(X=0|g1,g2) = (1-w1)^g1 x (1-w2)^g2
P(X=1|g1,g2) = 1-P(X=0|g1,g2)

P(G1=0|c1,c2,u1) = 1-P(G1=1|c1,c2,u1)
P(G1=1|c1,c2,u1) = (1-w_c1u1)^c1 x (1-w_c2u1)^c2 x (1-u1)

P(G2=0|c1,c2,u2) = 1-P(G2=1|c1,c2,u2)
P(G2=1|c1,c2,u2) = (1-w_c1u2)^c1 x (1-w_c2u2)^c2 x (1-u2)

このネットワークの場合、パラメタは6個。
パラメタ数は少ないが計算式は非常に複雑。
→ **単純な活性化関数と大量のパラメタを持つDNNとは対照的。**

ＱＢＣの実装

QBCの実装

- QBCからQBNに変換した後、全解探索をする。
- スケーラビリティに関する注意点：
  - QBNのCPTサイズは親ノード数に対して最悪指数オーダーで増える。
  - QBNの全解探索はQBNのノード数に対して最悪指数時間かかる。
- なお、真のベイジアンネット上で近似推論する場合はこれらスケーラビリティの問題は起きないと見込んでいる。

SAT ソルバの利用

- Java のSATソルバ SAT4J を使った QBN の全解探索が実装済み。非常に速い。

現状の実装の問題点

- この資料で説明した仕様と、現状の実装が食い違っている可能性がまだある。

制限付きベイジアンネットのスケーラビリティ

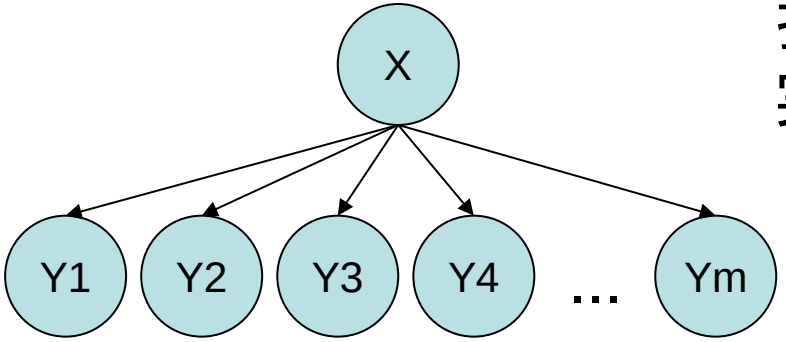
脳のモデルのスケーラビリティについて

- モデルのパラメタ数が爆発しないことに加え、脳のモデルは下記の要件を満たすべき:
  1. 認識・学習の反復アルゴリズムの1ステップが並列処理により実時間実行可能
  2. 認識・学習が多くの場合現実的なステップ数で収束
  3. 認識・学習が多くの場合現実的な精度で収束
- 1を満たすかどうかはアルゴリズムが決まれば判定可能。2，3はタスク依存。将来、実験で検証。

制限付きベイジアンネットのスケーラビリティ

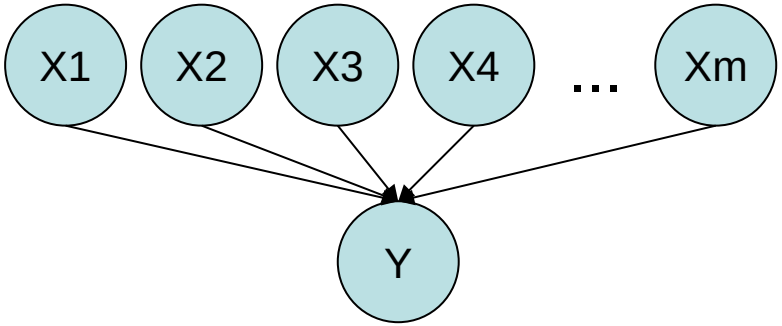
- **制限付きベイジアンネットの非常に重要な性質:**
  - noisy-OR, noisy-AND,排他性制約ノードはいずれも親ノード数 m のときサイズ O(m) の二分木のネットワークに変換できる。
- 並列実行によって、多くの反復アルゴリズムの反復の1ステップが**おそらく**実時間で動作:
  - 推論:loopy BP、平均場近似、MCMC
  - 学習:勾配降下法、stepwise EM

通常のベイジアンネットの問題点



推論・学習1ステップの
実行に必要な計算量

$O(m)$

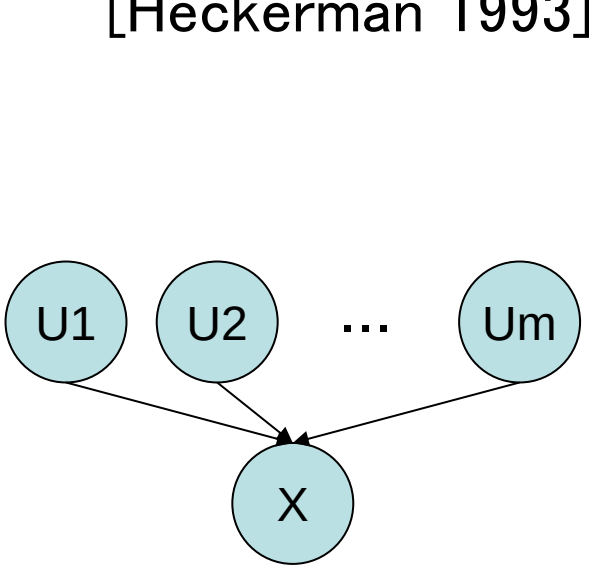


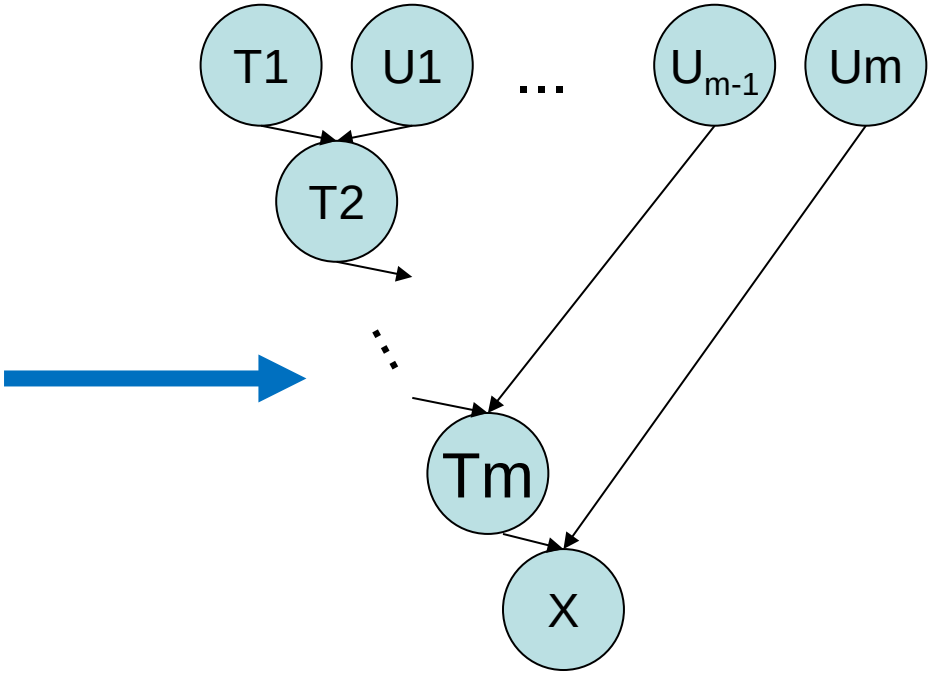
$O(2^m)$

多くの推論・学習アルゴリズムにおいて
親ノードの数に対し
指数関数的な計算量が必要

解決策：定数個の親ノードを持つ等価なネットワークに変換

[Heckerman 1993]





親ノードの数 m 個

親ノードの数は高々2個で深さが O(m)

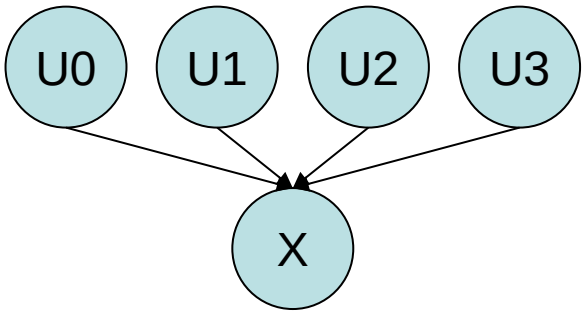
このような変換が可能な条件付確率表モデルであれば、
推論・学習の1ステップの実行が O(2^m) から **O(m) に高速化**

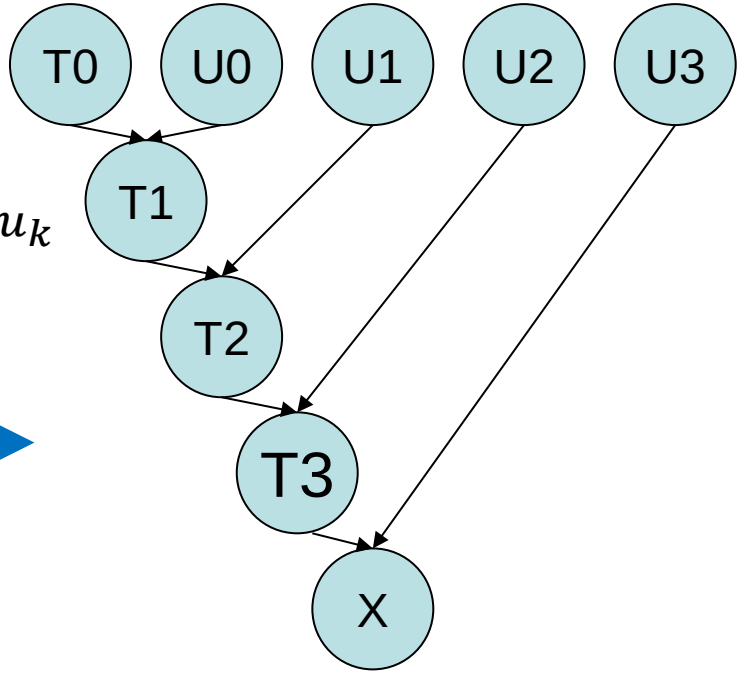
noisy-OR モデルは等価な2分木のネットワークに変換可能

noisy-AND も同様

$$P(T_0 = 0) = 1$$
$$P(T_{k+1} = 0 | T_k = 0, U_k = u_k) = (1 - w_k)^{u_k}$$
$$P(T_{k+1} = 0 | T_k = 1, U_k = u_k) = 0$$
$$X = T_m$$

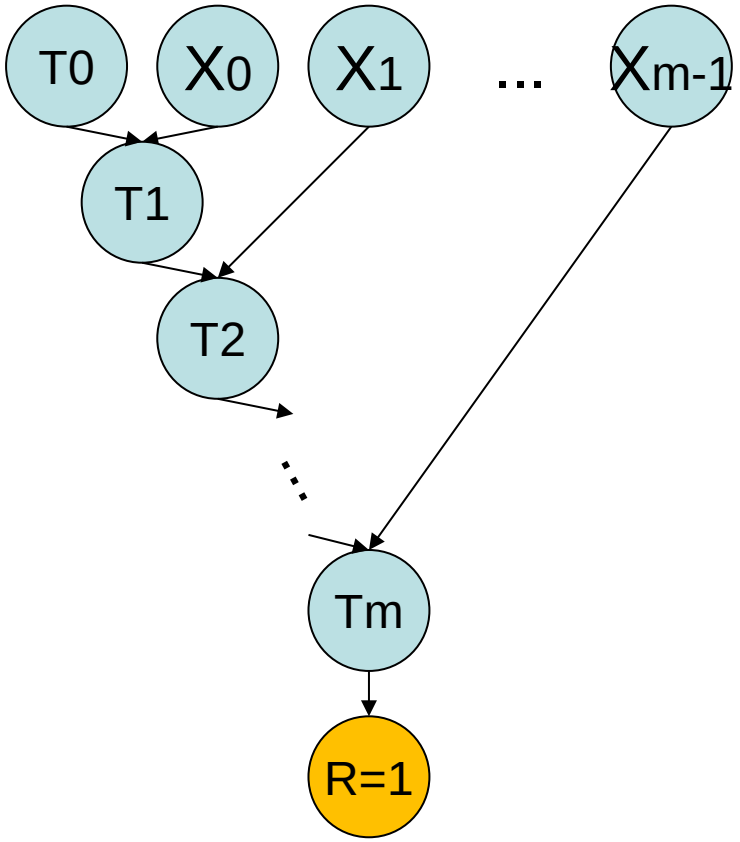
$$P(X = 0 | u_0, \ldots, u_{m-1}) = \Pi_{k=0}^{m-1} (1 - w_k)^{u_k}$$





排他性制約ノードも等価な2分木のネットワークに変換可能

$$P(T_0 = 0) = 1$$
$$P(T_{k+1} = 0 | T_k = 0, X_k = 0) = 1$$
$$P(T_{k+1} = 1 | T_k = 1, X_k = 0) = 1$$
$$P(T_{k+1} = 1 | T_k = 0, X_k = 1) = 1$$
$$P(T_{k+1} = 2 | T_k = 1, X_k = 1) = 1$$
$$P(T_{k+1} = 2 | T_k = 2, X_k = *) = 1$$
$$P(R = 1 | T_m = 0) = 1$$
$$P(R = 1 | T_m = 1) = 1$$
$$P(R = 1 | T_m = 2) = 0$$



Tk∈{0,1,2}
0:活性ノード数0
1:活性ノード数1
2:活性ノード数2以上

ＱＢＣを使った
認知機能モデルの
プロトタイピング

認知機能モデルのプロトタイピングの目的

- 制限付きベイジアンネットでさまざまな認知機能が実現できることを示したい。
- モデルのパラメタがデータから学習できそうなことを確認したい。
- 制限付きベイジアンネットの仕様に問題がないか確認し、問題があれば適宜改良したい。
- 神経科学的知見との対応も確認したい。

QBCが対象とする認知機能

1. 当面、カーネマンのいう「システム１」が対象
   (カーネマン「ファースト＆スロー」2014)
   – システム１：直感的、無意識的、高速、不正確
   – システム２：論理的、意識的、低速、正確
2. 当面、大脳皮質の領野のみが対象
   – 特に、視覚野、言語野、前頭葉に注力
3. 推論・認識のみが対象、学習は対象外
4. 本資料では短期記憶・時系列処理は対象外だが、今後は扱う

ベイジアンネットの推論アルゴリズムと脳

- ２種類の推論アルゴリズム：
  - 短時間で解が求まるが不正確な近似アルゴリズム：確率伝搬、平均場近似、etc.
  - 時間をかければいくらでも精度を上げられるアルゴリズム：MCMC
- 前者が脳のシステム１、後者がシステム2に近いと考えている。後者はおそらく前頭前野やメタ認知が深く関与。単純なMCMCではない。
  - 前述のとおりシステム2は本資料では扱わない。

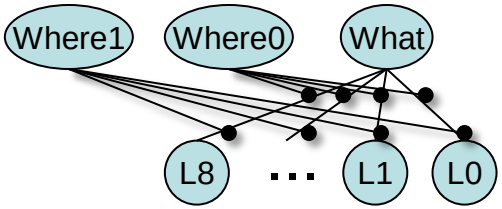
パラメタを学習可能なモデルが満たすべき条件？

- 同一階層内のノードはすべて独立。
- ユニットの発火は一様かつスパース。
- 生成モデルになっている。（root ノードの任意の値の組み合わせに対し解が必ず存在。）
- 親ノードは子ノードの値の組を抽象化。
- 問題のサイズに対してパラメタ数が爆発しない。

いまのところ、実装ずみのすべてのモデルは
これらの条件をだいたい満たしているように見える。

視覚野モデル

背側経路とグリッド細胞



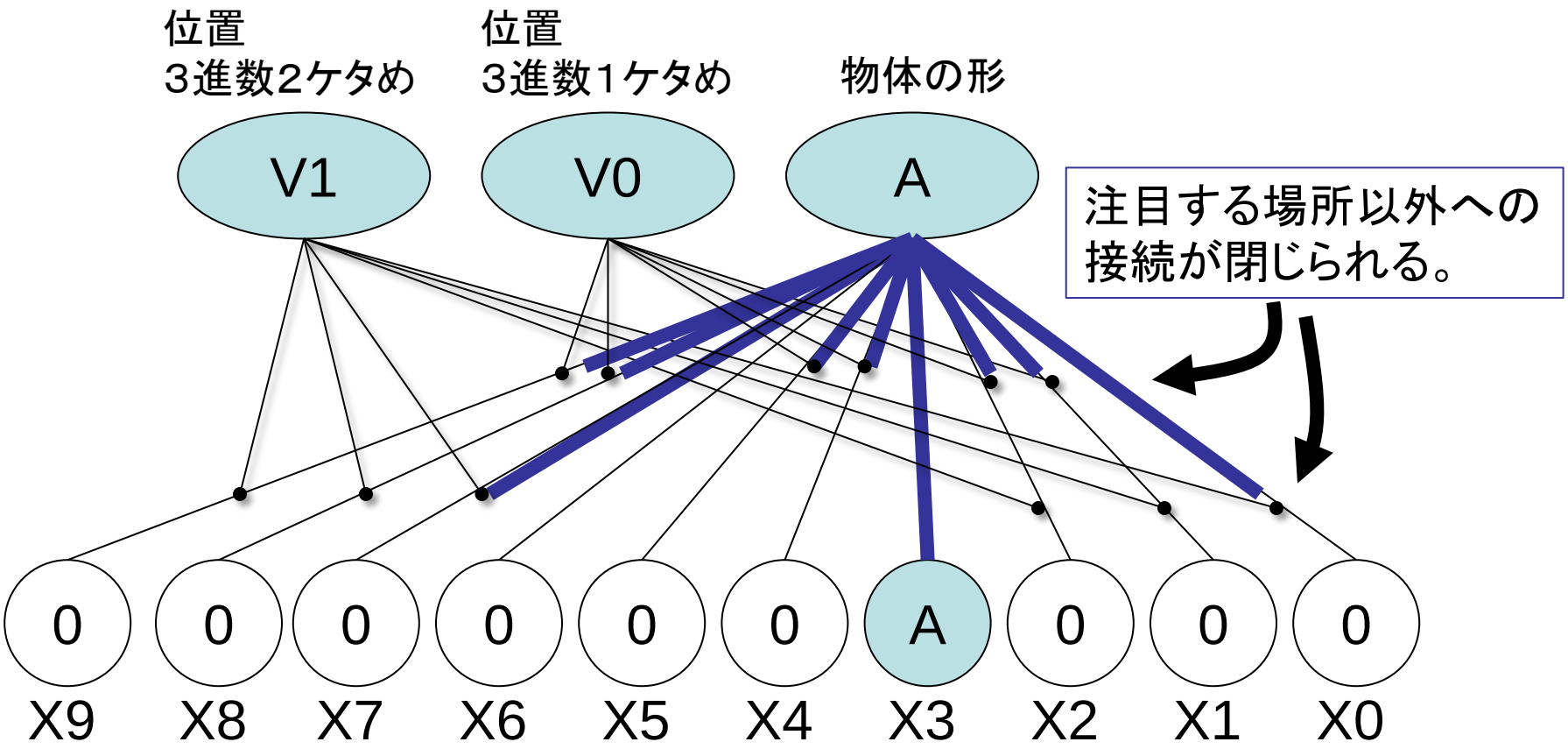

```
net.tableNodeList.add(tableNode(Where1, table(
        row(V0,I,I,I,I,I,O,O,O),
        row(V1,I,I,I,O,O,O,I,I,I),
        row(V2,O,O,O,I,I,I,I,I,I)
        )));
net.tableNodeList.add(tableNode(Where0, table(
        row(V0,I,I,O,I,I,O,I,I,O),
        row(V1,I,O,I,I,O,I,I,O,I),
        row(V2,O,I,I,O,I,I,O,I,I)
        )));
net.tableNodeList.add(tableNode(What, table(
        row(A,A,A,A,A,A,A,A,A,A),
        row(B,B,B,B,B,B,B,B,B,B)
        )));
net.tableNodeList.add(tableNode(L8, table(row(A),row(B))));
net.tableNodeList.add(tableNode(L7, table(row(A),row(B))));
net.tableNodeList.add(tableNode(L6, table(row(A),row(B))));
net.tableNodeList.add(tableNode(L5, table(row(A),row(B))));
net.tableNodeList.add(tableNode(L4, table(row(A),row(B))));
net.tableNodeList.add(tableNode(L3, table(row(A),row(B))));
net.tableNodeList.add(tableNode(L2, table(row(A),row(B))));
net.tableNodeList.add(tableNode(L1, table(row(A),row(B))));
net.tableNodeList.add(tableNode(L0, table(row(A),row(B))));
net.gateMatrixes.add(new GateMatrix(
        list(Where1,Where0), list(What),
        list(L8,L7,L6,L5,L4,L3,L2,L1,L0)
        ));
```

すべての解

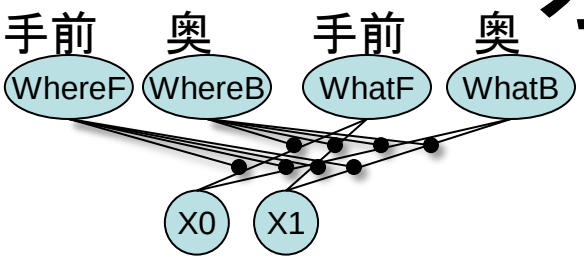
```
Where1, Where0, What, L8, L7, L6, L5, L4, L3, L2, L1, L0,
V0, V0, A, 0, 0, 0, 0, 0, 0, 0, 0, A,
V0, V0, B, 0, 0, 0, 0, 0, 0, 0, 0, B,
V0, V1, A, 0, 0, 0, 0, 0, 0, 0, A, 0,
V0, V1, B, 0, 0, 0, 0, 0, 0, 0, B, 0,
V0, V2, A, 0, 0, 0, 0, 0, 0, A, 0, 0,
V0, V2, B, 0, 0, 0, 0, 0, 0, B, 0, 0,
V1, V0, A, 0, 0, 0, 0, 0, A, 0, 0, 0,
V1, V0, B, 0, 0, 0, 0, 0, B, 0, 0, 0,
V1, V1, A, 0, 0, 0, 0, A, 0, 0, 0, 0,
V1, V1, B, 0, 0, 0, 0, B, 0, 0, 0, 0,
V1, V2, A, 0, 0, 0, A, 0, 0, 0, 0, 0,
V1, V2, B, 0, 0, 0, B, 0, 0, 0, 0, 0,
V2, V0, A, 0, 0, A, 0, 0, 0, 0, 0, 0,
V2, V0, B, 0, 0, B, 0, 0, 0, 0, 0, 0,
V2, V1, A, 0, A, 0, 0, 0, 0, 0, 0, 0,
V2, V1, B, 0, B, 0, 0, 0, 0, 0, 0, 0,
V2, V2, A, A, 0, 0, 0, 0, 0, 0, 0, 0,
V2, V2, B, B, 0, 0, 0, 0, 0, 0, 0, 0,
```

Where0 のユニットは
一次元格子点上に受容野を持つ。

ネットワークの状態の例



オクルージョン



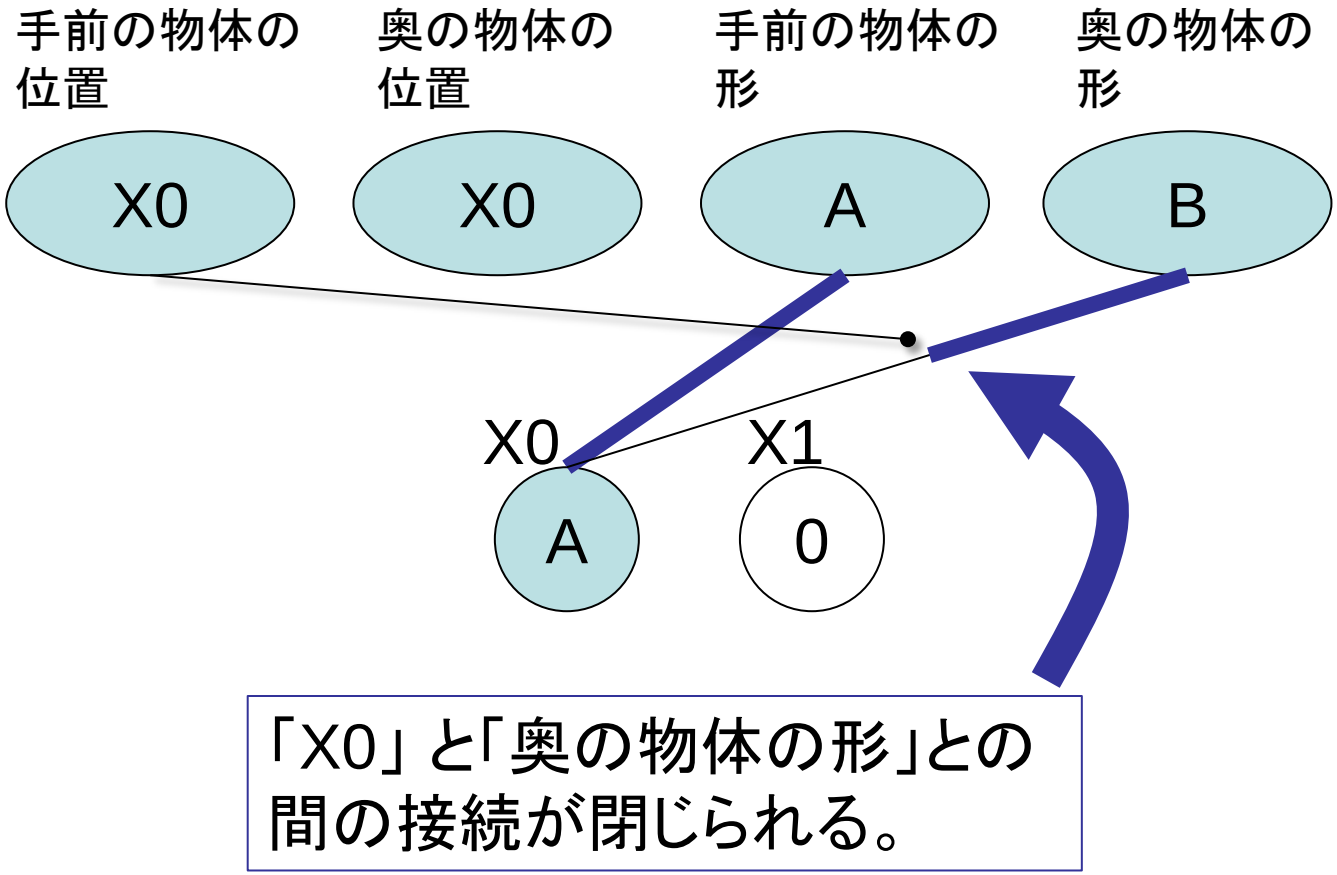
2つの物体の位置が重なっているとき、
手前のものだけが見える。

```
net.tableNodeList.add(tableNode(WhereF, table(
        // WhatF->X0, WhatB->X0, WhatF->X1, WhatB->X1
        row(X0,O,I,I,O),
        row(X1,I,O,O,I)
        )));
net.tableNodeList.add(tableNode(WhereB, table(
        // WhatF->X0, WhatB->X0, WhatF->X1, WhatB->X1
        row(X0,O,O,O,I),
        row(X1,O,I,O,O)
        )));
net.tableNodeList.add(tableNode(WhatF, table(
        row(A,A,A),
        row(B,B,B),
        row(O,O,O)
        )));
net.tableNodeList.add(tableNode(WhatB, table(
        row(A,A,A),
        row(B,B,B),
        row(O,O,O)
        )));
net.tableNodeList.add(tableNode(X0, table(row(A),row(B))));
net.tableNodeList.add(tableNode(X1, table(row(A),row(B))));
net.gateMatrixes.add(new GateMatrix(
        list(WhereF,WhereB), list(WhatF,WhatB), list(X0,X1) ));
```

すべての解

WhereF, WhereB, WhatF, WhatB, X0, X1,
X0, X0, 0, 0, 0, 0,
X0, X0, 0, A, 0, 0,
X0, X0, 0, B, 0, 0,
X0, X0, A, 0, A, 0,
X0, X0, A, A, A, 0,
X0, X0, A, B, A, 0,
X0, X0, B, 0, B, 0,
X0, X0, B, A, B, 0,
X0, X0, B, B, B, 0,
X0, X1, 0, 0, 0, 0,
X0, X1, 0, A, 0, A,
X0, X1, 0, B, 0, B,
X0, X1, A, 0, A, 0,
X0, X1, A, A, A, A,
X0, X1, A, B, A, B,
X0, X1, B, 0, B, 0,
X0, X1, B, A, B, A,
X0, X1, B, B, B, B,
X1, X0, 0, 0, 0, 0,
X1, X0, 0, A, A, 0,
X1, X0, 0, B, B, 0,
X1, X0, A, 0, 0, A,
X1, X0, A, A, A, A,
X1, X0, A, B, B, A,
X1, X0, B, 0, 0, B,
X1, X0, B, A, A, B,
X1, X0, B, B, B, B,
X1, X1, 0, 0, 0, 0,
X1, X1, 0, A, 0, 0,
X1, X1, 0, B, 0, 0,
X1, X1, A, 0, 0, A,
X1, X1, A, A, 0, A,
X1, X1, A, B, 0, A,
X1, X1, B, 0, 0, B,
X1, X1, B, A, 0, B,
X1, X1, B, B, 0, B,

ネットワークの状態の例



言語野モデル

CCGのCYKパーザの実現に向けて

- いきなり大きいパーザを作るのは大変な上、推論に時間がかかるので、3つの機能に分解して試作する。

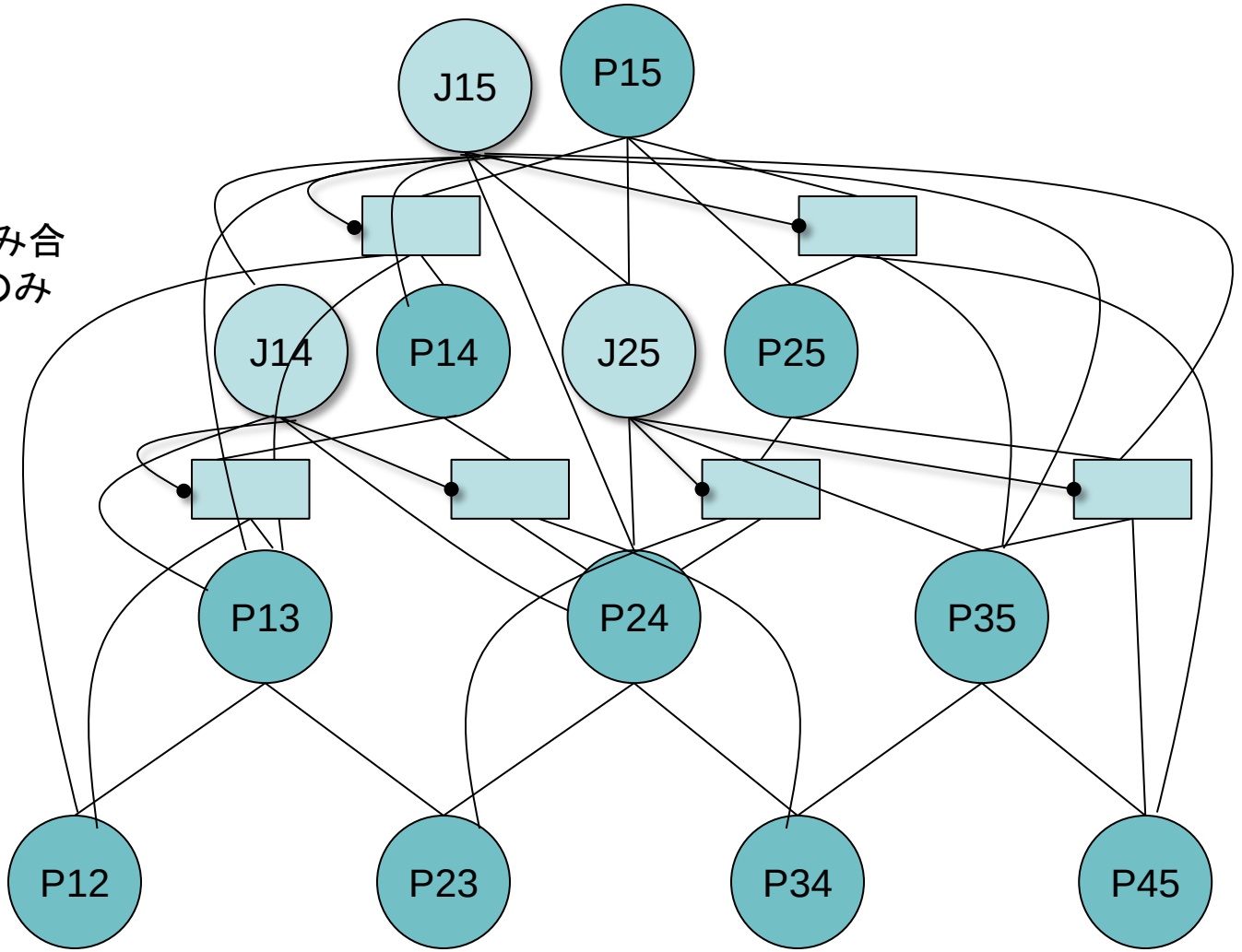
  1. すべての構文木を生成するネットワーク。

  2. CCGの2つの統語範疇を1つに統合するネットワーク。

  3. CCGの意味規則（ラムダ式の適用・合成）に相当する機能を実現するネットワーク。

構文解析をする回路

J15 は、
区間
[1,2]と[2,5]、
[1,3]と[3,5]、
[1,4]と[4,5]の
いずれかの組み合
わせのゲートのみ
を開ける。



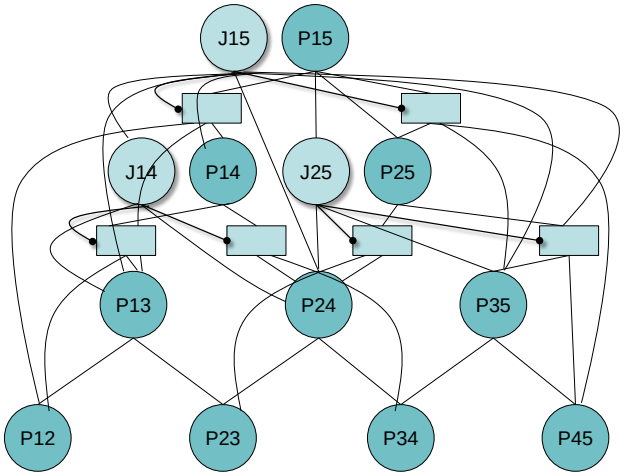
CYKパーザの QBC 定義

QBC定義の一部

```
net.tableNodeList.add(tableNode(P13, table(
        row(N1,N11,N12),
        row(N2,N21,N22),
        row(N11,N111,N112),
        row(N12,N121,N122),
        row(N21,N211,N212),
        row(N22,N221,N222),
        row(I,__,__))));
net.tableNodeList.add(tableNode(J15, table(
        //childNames: [P13, J14, P14, J25, P25,
P35, P24, P15->P12, P15->P13, P15->P14, P15->P25,
P15->P35, P15->P45]
        row(J2, // 区間 [1,2],[2,5]
            I,I,I,
            or(J3,J4),__,__,
            __,
            O,I,I,
            O,I,I),
        row(J3, // 区間 [1,3],[3,5]
            __,I,I,
            I,I,__,
            I,
            I,O,I,
            I,O,I),
        row(J4, // 区間 [1,4],[4,5]
            __,or(J2,J3),__,
            I,I,I,
            __,
            I,I,O,
            I,I,O)
        )));
```

文法

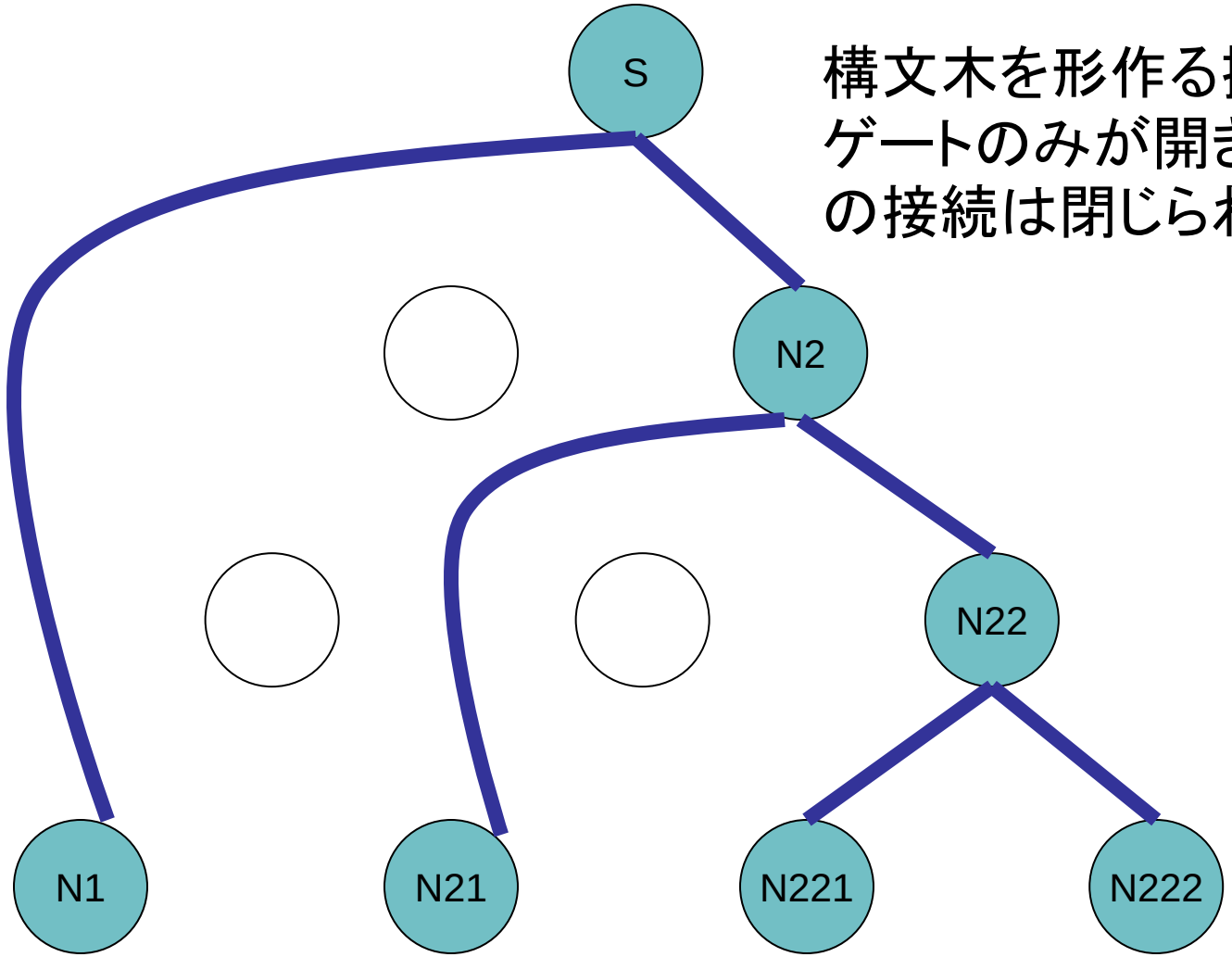
S -> N1 N2
N1 -> N11 N12
N2 -> N21 N22
N11 -> N111 N112
N12 -> N121 N122
N21 -> N211 N212
N22 -> N221 N222



すべての解

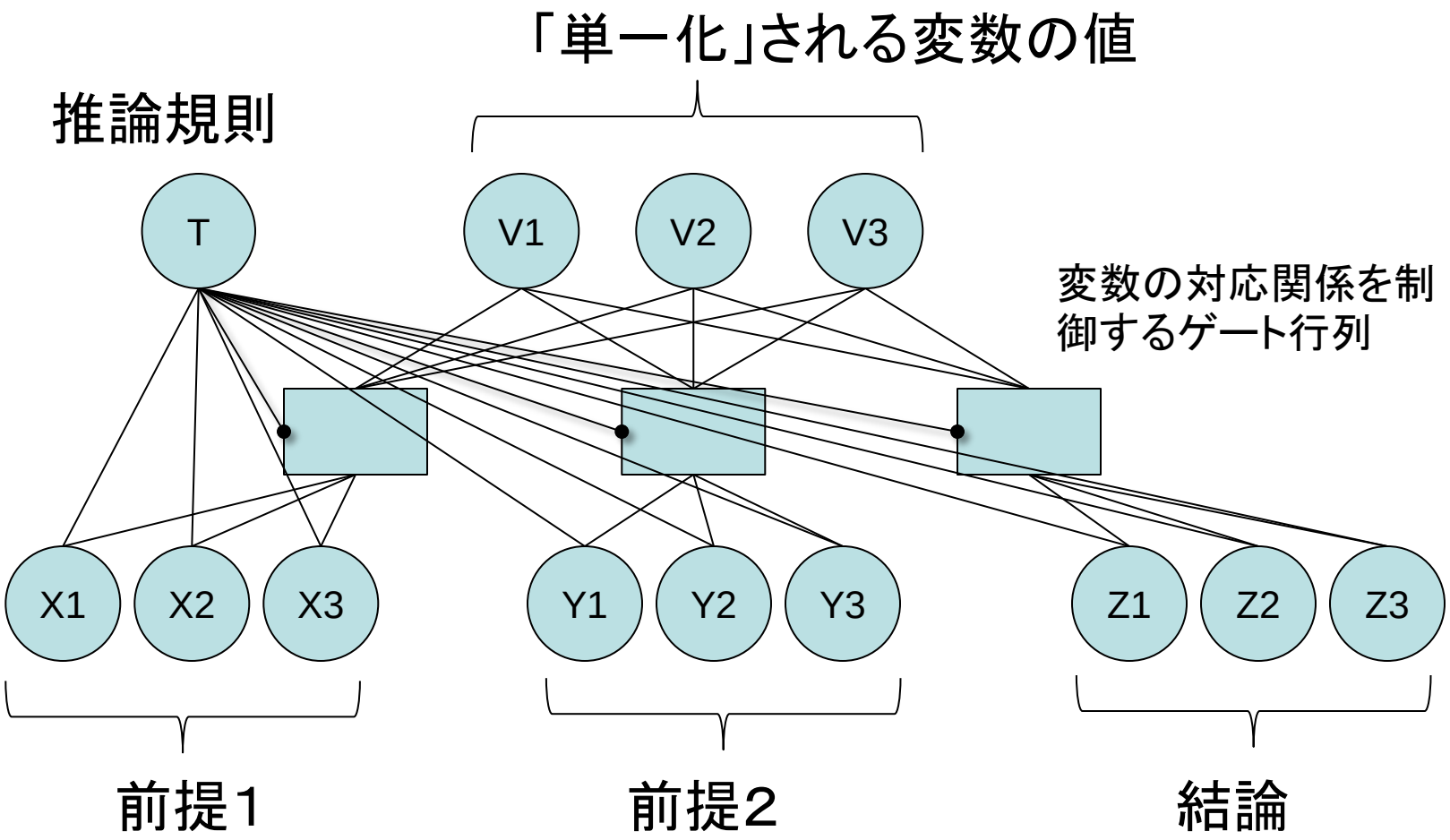
P15, P14, P25, P13, P24, P35, P12, P23, P34, P45, J15, J14, J25,
S, 1, N2, 1, 1, N22, N1, N21, N221, N222, J2, 1, J3,
S, 1, N2, 1, N21, 1, N1, N211, N212, N22, J2, 1, J4,
S, 1, 1, N1, 1, N2, N11, N12, N21, N22, J3, 1, 1,
S, N1, 1, 1, N12, 1, N11, N121, N122, N2, J4, J2, 1,
S, N1, 1, N11, 1, 1, N111, N112, N12, N2, J4, J3, 1,
5 solutions.

構文木の形が決まった時のネットワークの状態の例



構文木を形作る接続のゲートのみが開き、他の接続は閉じられる。

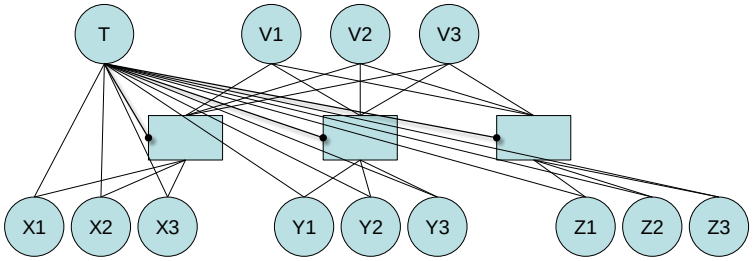
ユニフィケーションを使って推論規則の適用を実行する回路



推論規則適用のQBC定義

```
QBCnet net = new QBCnet();
Object var = or(A,B,C);
net.tableNodeList.add(tableNode(T, table(
      // childNames: [X1, X2, X3, Y1, Y2, Y3, Z1, Z2, Z3,
      // V1->X1, V2->X1, V3->X1, V1->X2, V2->X2, V3->X2, V1->X3, V2->X3, V3->X3,
      // V1->Y1, V2->Y1, V3->Y1, V1->Y2, V2->Y2, V3->Y2, V1->Y3, V2->Y3, V3->Y3,
      // V1->Z1, V2->Z1, V3->Z1, V1->Z2, V2->Z2, V3->Z2, V1->Z3, V2->Z3, V3->Z3]
      row(T1, // V1, V1->V2  ==>  V2
            var,N,N, var,THEN,var, var,N,N,
            O,I,I, I,I,I, I,I,I,
            O,I,I, I,I,I, I,O,I,
            I,O,I, I,I,I, I,I,I),
      row(T2, // not V2, V1->V2  ==>  not V1
            N,NOT,var, var,THEN,var, N,NOT,var,
            I,I,I, I,I,I, I,O,I,
            O,I,I, I,I,I, I,O,I,
            I,I,I, I,I,I, O,I,I),
      row(T3, // V1 and V2, not V1  ==>  V2
            var,AND,var, N,NOT,var, var,N,N,
            O,I,I, I,I,I, I,O,I,
            I,I,I, I,I,I, O,I,I,
            I,O,I, I,I,I, I,I,I),
      row(T4, // V1->V2, V2->V3  ==>  V1->V3
            var,THEN,var, var,THEN,var, var,THEN,var,
            O,I,I, I,I,I, I,O,I,
            I,O,I, I,I,I, I,I,O,
            O,I,I, I,I,I, I,I,O)
      )));
List<TableRow> vars9Table = table(
      row(A, A,A,A, A,A,A, A,A,A),
      row(B, B,B,B, B,B,B, B,B,B),
      row(C, C,C,C, C,C,C, C,C,C),
      row(N, N,N,N ,N,N,N, N,N,N));
List<TableRow> varsTable = table(
      row(A), row(B), row(C), row(N));
List<TableRow> opsTable = table(
      row(AND), row(OR), row(NOT), row(THEN), row(N));
net.tableNodeList.add(tableNode(V1, vars9Table));
net.tableNodeList.add(tableNode(V2, vars9Table));
net.tableNodeList.add(tableNode(V3, vars9Table));
net.tableNodeList.add(tableNode(X1, varsTable));
net.tableNodeList.add(tableNode(X2, opsTable));
net.tableNodeList.add(tableNode(X3, varsTable));
net.tableNodeList.add(tableNode(Y1, varsTable));
net.tableNodeList.add(tableNode(Y2, opsTable));
net.tableNodeList.add(tableNode(Y3, varsTable));
net.tableNodeList.add(tableNode(Z1, varsTable));
net.tableNodeList.add(tableNode(Z2, opsTable));
net.tableNodeList.add(tableNode(Z3, varsTable));
net.gateMatrixList.add(gateMatrix(
      list(), list(T),
      list(X1,X2,X3,Y1,Y2,Y3,Z1,Z2,Z3)
      ));
net.gateMatrixList.add(gateMatrix(
      list(T), list(V1,V2,V3),
      list(X1,X2,X3,Y1,Y2,Y3,Z1,Z2,Z3)
      ));
return net;
```

推論規則

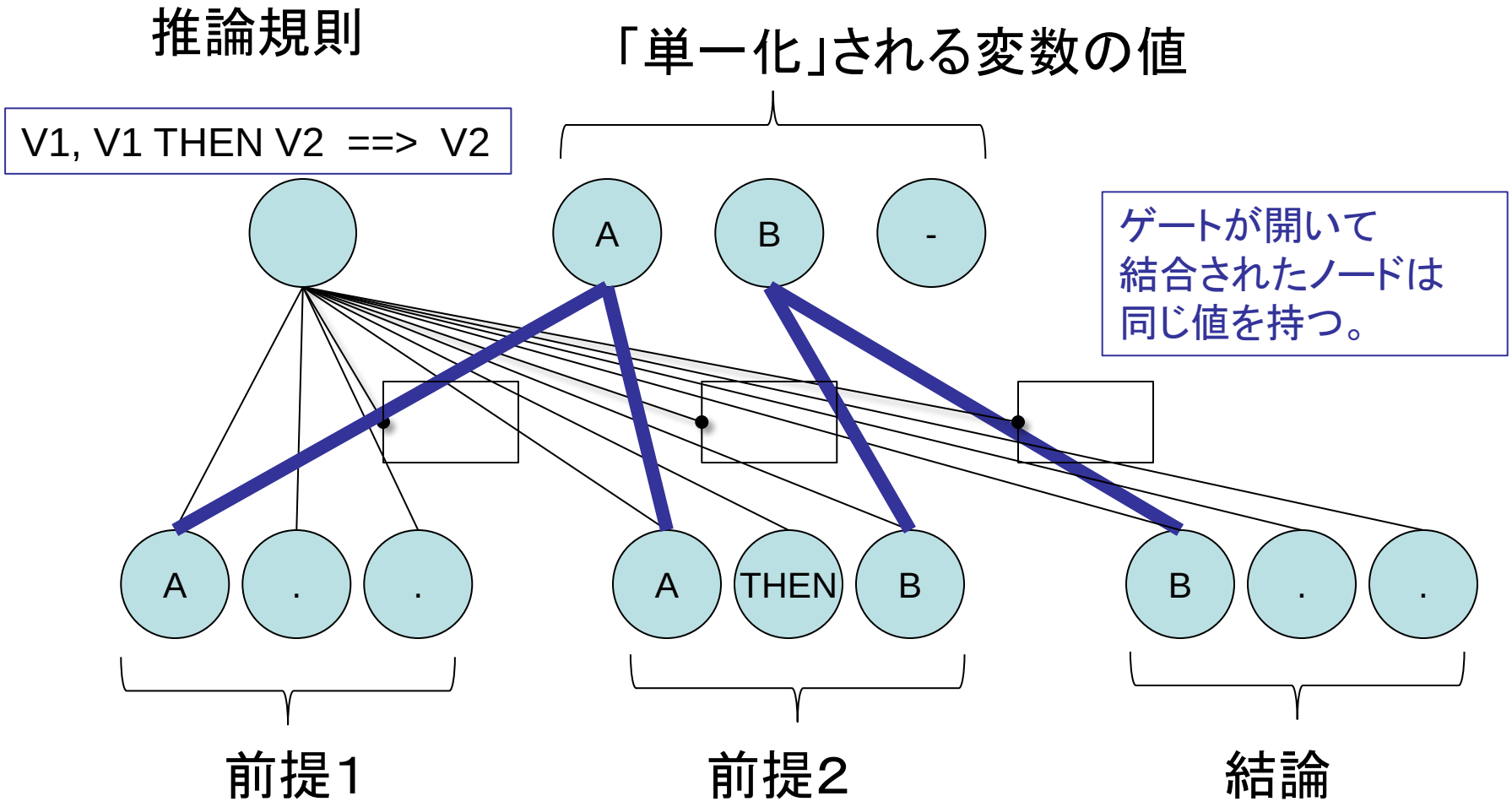


V1, V1 THEN V2 ==> V2
NOT V2, V1 THEN V2 ==> NOT V1
V1 AND V2, NOT V1 ==> V2
V1 THEN V2, V2 THEN V3 ==> V1 THEN V3

すべての解

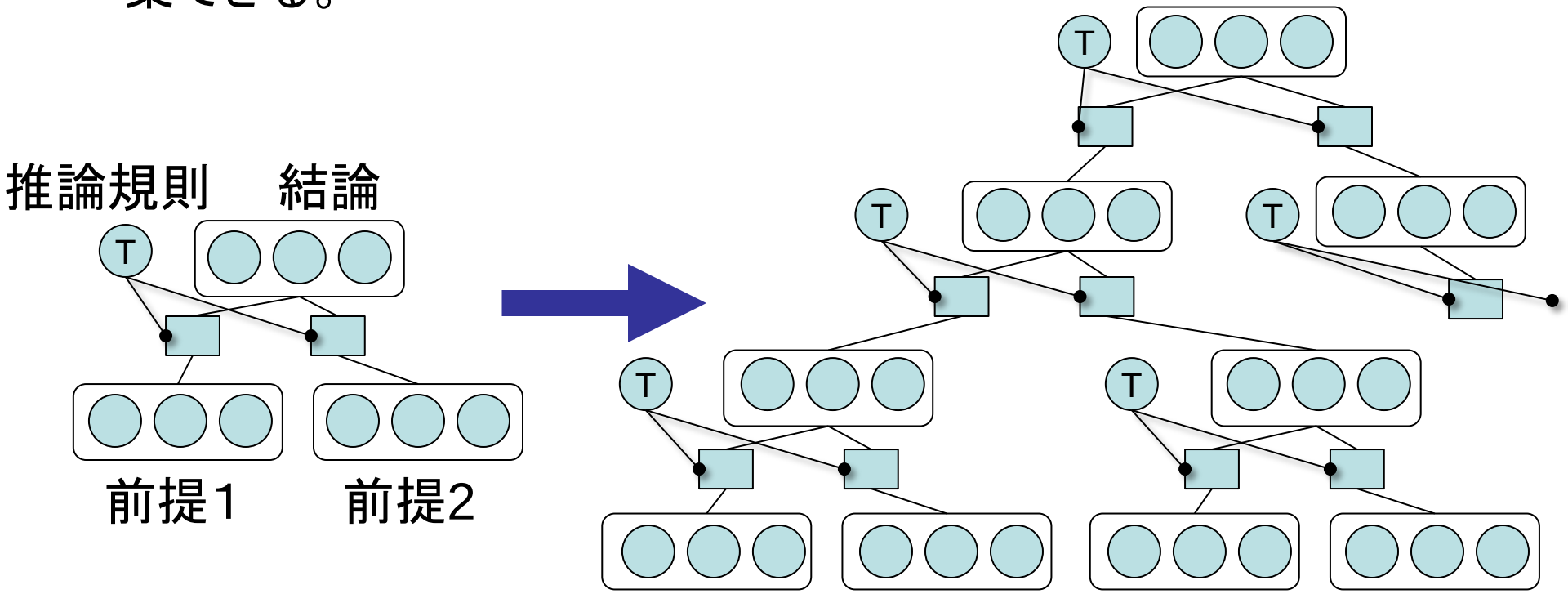
T, V1, V2, V3, X1, X2, X3, Y1, Y2, Y3, Z1, Z2, Z3,
T1, A, A, A, A, ., ., A, THEN, A, A, ., .,
T1, B, A, A, B, ., ., B, THEN, A, A, ., .,
T1, C, A, A, C, ., ., C, THEN, A, A, ., .,
T1, A, A, B, A, ., ., A, THEN, A, A, ., .,
T1, B, A, B, B, ., ., B, THEN, A, A, ., .,
T1, C, A, B, C, ., ., C, THEN, A, A, ., .,
…
T4, A, C, A, A, THEN, C, C, THEN, A, A, THEN, A,
T4, B, C, A, B, THEN, C, C, THEN, A, B, THEN, A,
T4, C, C, A, C, THEN, C, C, THEN, A, C, THEN, A,
T4, A, C, B, A, THEN, C, C, THEN, B, A, THEN, B,
T4, B, C, B, B, THEN, C, C, THEN, B, B, THEN, B,
T4, C, C, B, C, THEN, C, C, THEN, B, C, THEN, B,
T4, A, C, C, A, THEN, C, C, THEN, C, A, THEN, C,
T4, B, C, C, B, THEN, C, C, THEN, C, B, THEN, C,
T4, C, C, C, C, THEN, C, C, THEN, C, C, THEN, C,
162 solutions.

推論規則を1つ選択したときのネットワークの状態の例

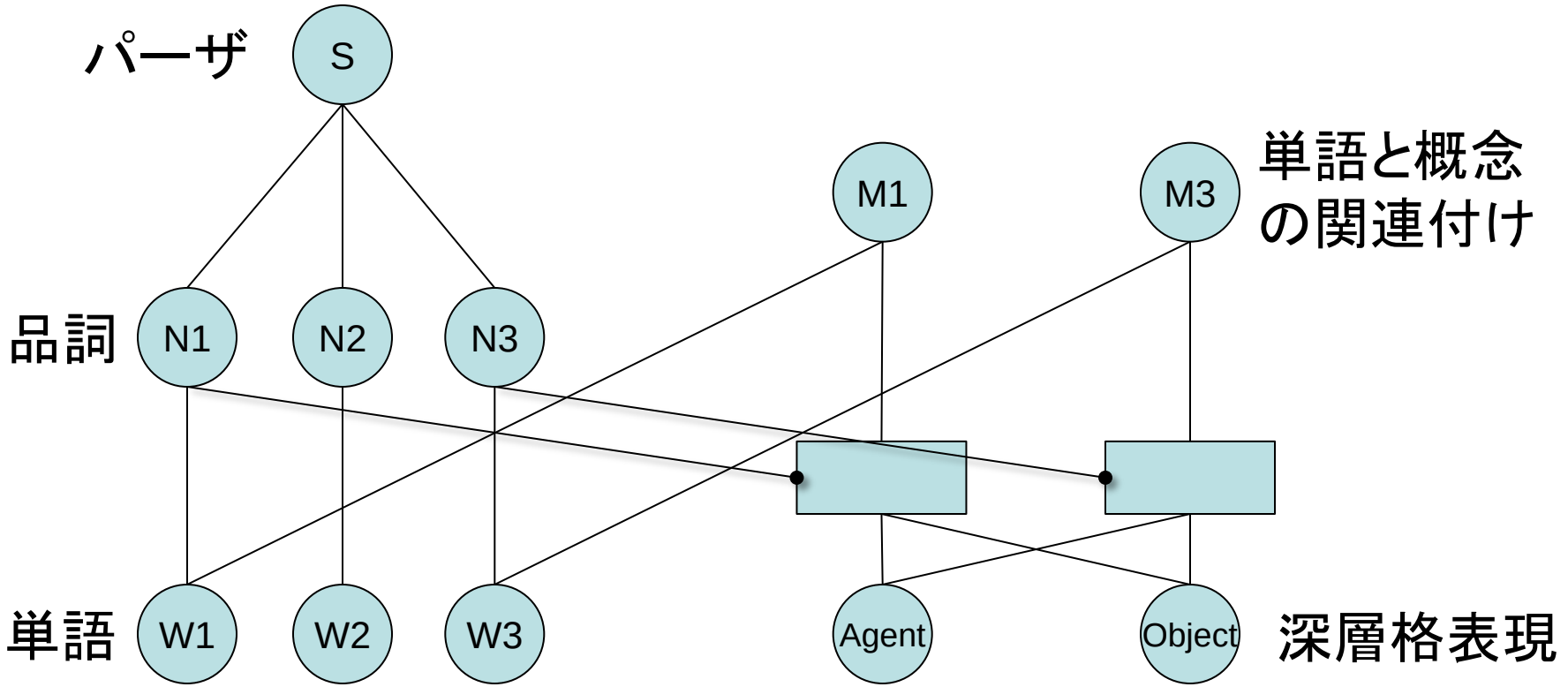


推論器を積み重ねる方法

結論を上流ノードに置けば、ピラミッド状に積み重ねて複数の推論を一度に行う回路を構築できる。

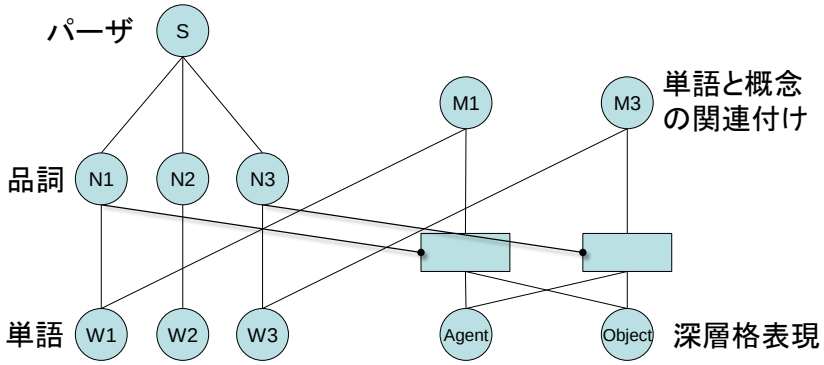


単語列から深層格への変換の原理を説明する回路



深層格への変換のQBC定義

```
QBCnet net = new QBCnet();
net.tableNodeList.add(tableNode(S, table(
      row(Sactive,Nagent,NactiveVerb,Nobject),
      row(Spassive,Nobject,NpassiveVerb,Nagent))));
net.tableNodeList.add(tableNode(N1, table(
      row(Nagent,or(Wcat,Wfish), O,I),
      row(Nobject,or(Wcat,Wfish), I,O))));
net.tableNodeList.add(tableNode(N2, table(
      row(NactiveVerb, Weat),
      row(NpassiveVerb,WisEatenBy))));
net.tableNodeList.add(tableNode(N3, table(
      row(Nagent,or(Wcat,Wfish), O,I),
      row(Nobject,or(Wcat,Wfish), I,O))));
net.tableNodeList.add(tableNode(M1, table(
      row(V1, Wcat, Cat, Cat),
      row(V2, Wfish, Fish, Fish))));
net.tableNodeList.add(tableNode(M3, table(
      row(V1, Wcat, Cat, Cat),
      row(V2, Wfish, Fish, Fish))));
net.tableNodeList.add(tableNode(W1, table(
      row(Wcat), row(Wfish))));
net.tableNodeList.add(tableNode(W2, table(
      row(Weat), row(WisEatenBy))));
net.tableNodeList.add(tableNode(W3, table(
      row(Wcat), row(Wfish))));
net.tableNodeList.add(tableNode(Agent, table(
      row(Cat), row(Fish))));
net.tableNodeList.add(tableNode(Object, table(
      row(Cat), row(Fish))));
net.gateMatrixList.add(gateMatrix(
      list(), list(S), list(N1,N2,N3)));
net.gateMatrixList.add(gateMatrix(
      list(), list(N1), list(W1)));
net.gateMatrixList.add(gateMatrix(
      list(), list(N2), list(W2)));
net.gateMatrixList.add(gateMatrix(
      list(), list(N3), list(W3)));
net.gateMatrixList.add(gateMatrix(
      list(), list(M1), list(W1)));
net.gateMatrixList.add(gateMatrix(
      list(), list(M3), list(W3)));
net.gateMatrixList.add(gateMatrix(
      list(N1), list(M1), list(Agent,Object)));
net.gateMatrixList.add(gateMatrix(
      list(N3), list(M3), list(Agent,Object)));
return net;
```



文法

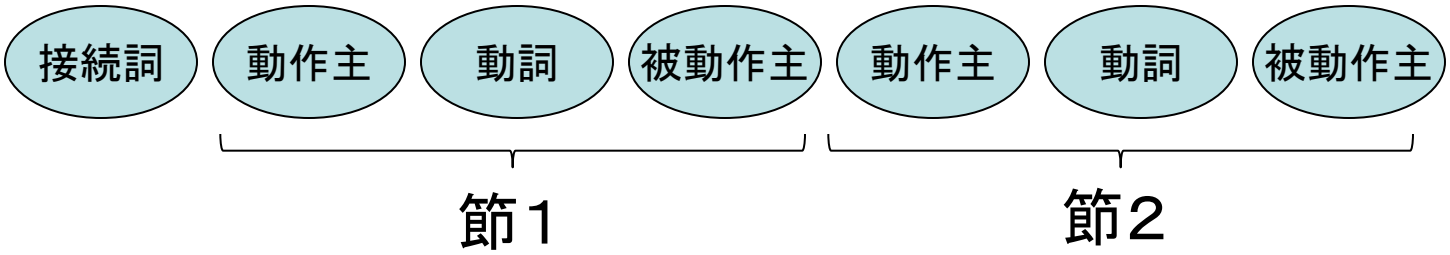
S -> Sactive　　S -> Spassive
Sactive -> Nagent NactiveVerb Nobject
Spassive, Nobject NpassiveVerb Nagent
Nagent -> Wcat　　Nobject -> Wcat
Nagent -> Wfish　　Nobject -> Wfish
NactiveVerb -> Weat
NpassiveVerb -> WisEatenBy

すべての解

S, N1, N2, N3, M1, M3, W1, W2, W3, Agent, Object,
Sactive, Nagent, NactiveVerb, Nobject, V1, V1, Wcat, Weat, Wcat, Cat, Cat,
Sactive, Nagent, NactiveVerb, Nobject, V1, V2, Wcat, Weat, Wfish, Cat, Fish,
Sactive, Nagent, NactiveVerb, Nobject, V2, V1, Wfish, Weat, Wcat, Fish, Cat,
Sactive, Nagent, NactiveVerb, Nobject, V2, V2, Wfish, Weat, Wfish, Fish, Fish,
Spassive, Nobject, NpassiveVerb, Nagent, V1, V1, Wcat, WisEatenBy, Wcat, Cat, Cat,
Spassive, Nobject, NpassiveVerb, Nagent, V1, V2, Wcat, WisEatenBy, Wfish, Fish, Cat,
Spassive, Nobject, NpassiveVerb, Nagent, V2, V1, Wfish, WisEatenBy, Wcat, Cat, Fish,
Spassive, Nobject, NpassiveVerb, Nagent, V2, V2, Wfish, WisEatenBy, Wfish, Fish, Fish,
8 solutions.

複文に対応するための拡張方針

- 幼児の言語に相当する単純な文に限ることにする。
  - If A, B. など、「接続詞 節１ 節２」の形に限定。
  - 深層格（動作主、被動作主など）は定数個。
- 名詞の意味を入れる場所は、節番号と深層格の組で表現できる。
  - この「場所」を統語範疇の属性値として表現。



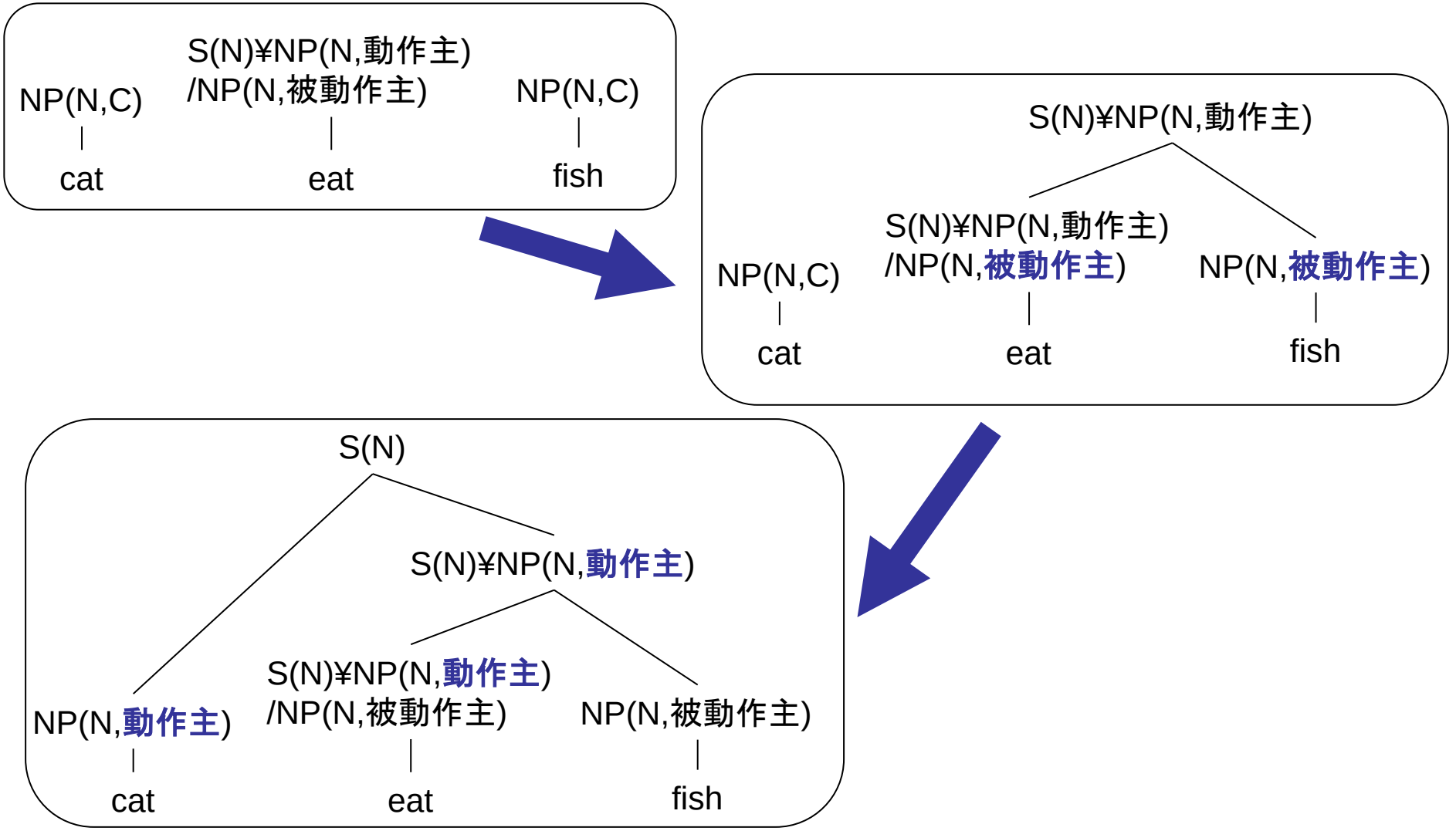
場所を属性値として持つ統語範疇

if ：CS/S(節1)/S(節2)
cat : NP(N,C)
fish : NP(N,C)
eat : S(N)¥NP(N,動作主)/NP(N,被動作主)
isEatenBy : S(N)¥NP(N,被動作主) /NP(N,動作主)
white : NP(N,C)/NP(N,C)
black : NP(N,C)/NP(N,C)

節番号 N ∈ {節1、節2}
深層格 C ∈ {動作主、被動作主、・・・}

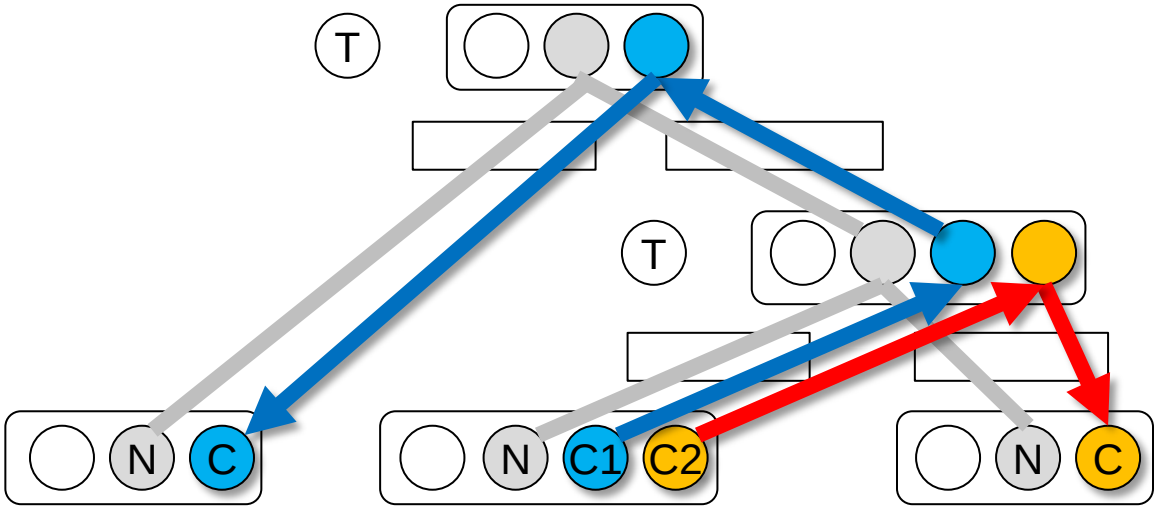
単語の段階ではどの「場所」に値を埋めるべきかあいまいだが、統語範疇の統合が進むにつれ属性値がユニフィケーションにより確定していき、最終的には埋める場所がすべて確定する。

ユニフィケーションで
属性値が決まっていく過程



ベイジアンネット上で
属性値が伝搬していく様子

・構文木と使用する推論規則が決まると、
ユニファイすべき変数どうしが結合される。
・結合を通じて属性値が伝搬する。



C1=動作主,
C2=被動作主

依存型意味論との融合に向けて

QBCと依存型

- 「値」が特徴ベクトル表現になっていれば、ワイルドカードを使うことで、is-a 階層が表現可能。
  - 値が型の情報を持っている、すなわち依存型。

(足、毛、キバ、たてがみ、大きさ)

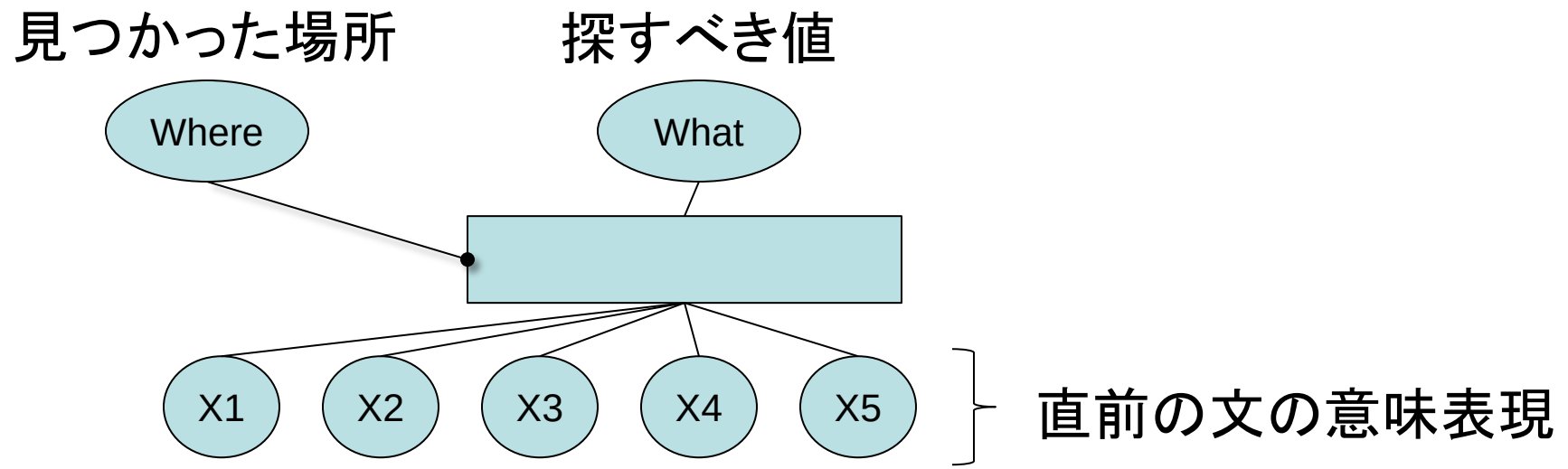
けもの ＝ (4本、あり、 _ 、 _ 、 _)

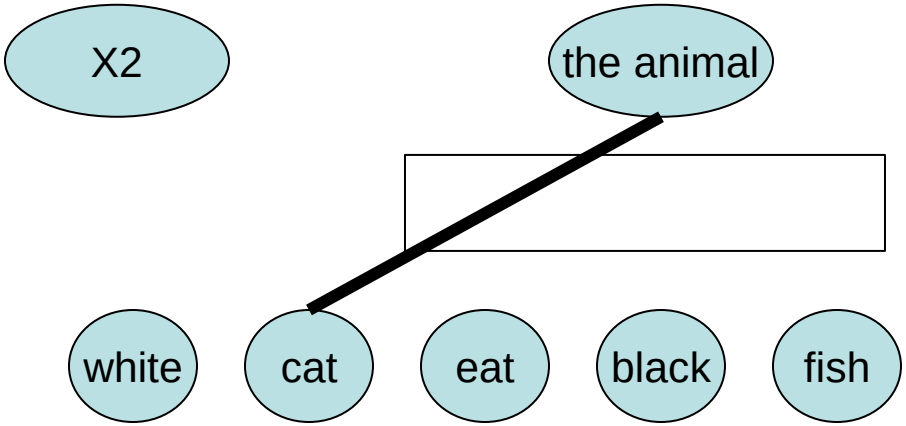
猛獣　＝ (4本、あり、あり、 _ 、 大)

ライオン=(4本、あり、あり、あり、 大)

牛　　＝ (4本、あり、なし、なし、大)

型が一致する値がある場所を見つけ出す回路(案)





"the animal" と一致する値 "cat" がある場所 X2 が Where ノードの値になる

まとめ

- 制限付きベイジアンネットは様々な点で有望
  - 大規模化可能。
  - 少ないパラメタで対象を簡潔に表現可能。
    → 学習データが少ないときに有利
  - 神経科学的知見との対応もよさそう。
- QBCは制限付きベイジアンネットを使った認知機能モデル設計のための便利な道具
  - 比較的書きやすく読みやすい。
  - 大規模機械学習の黒魔術的難しさを回避し、モデル設計に集中できる。

今後やるべき仕事

- ドキュメント
- トランスレータのエラー処理
- GUIによる推論結果の可視化、値のインタラクティブな変更
- ブラウザでのデモ
- さまざまな認知モデルの実装