単一化アルゴリズムとHPSGパージング

二宮　崇

今日の講義の予定

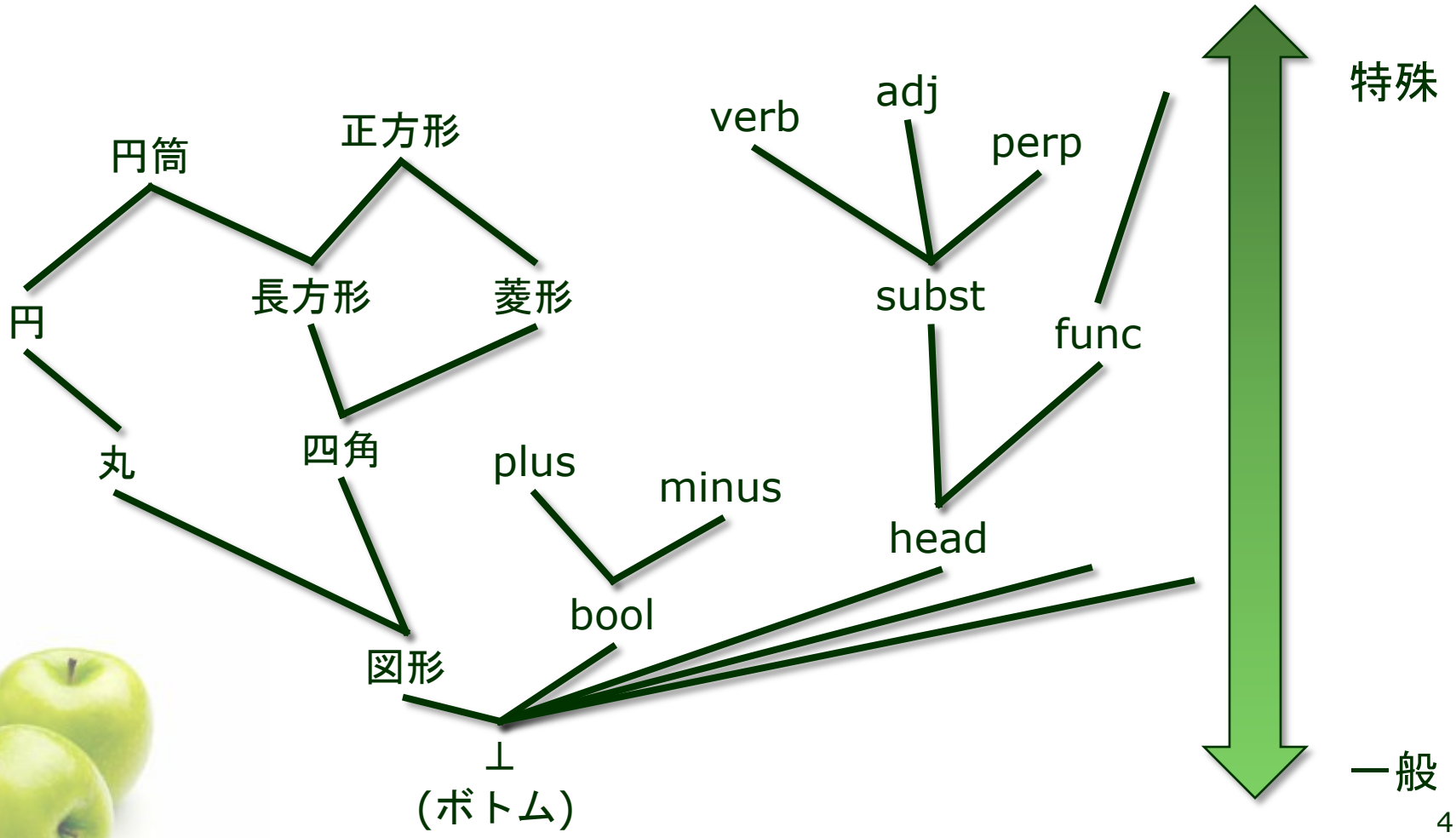
- 単一化アルゴリズム
- HPSGのフルパージング
- 教科書
  - C. D. Manning & Hinrich Schütze “FOUNDATIONS OF STATISTICAL NATURAL LANGUAGE PROCESSING” MIT Press, 1999

型付素性構造の単一化アルゴリズム

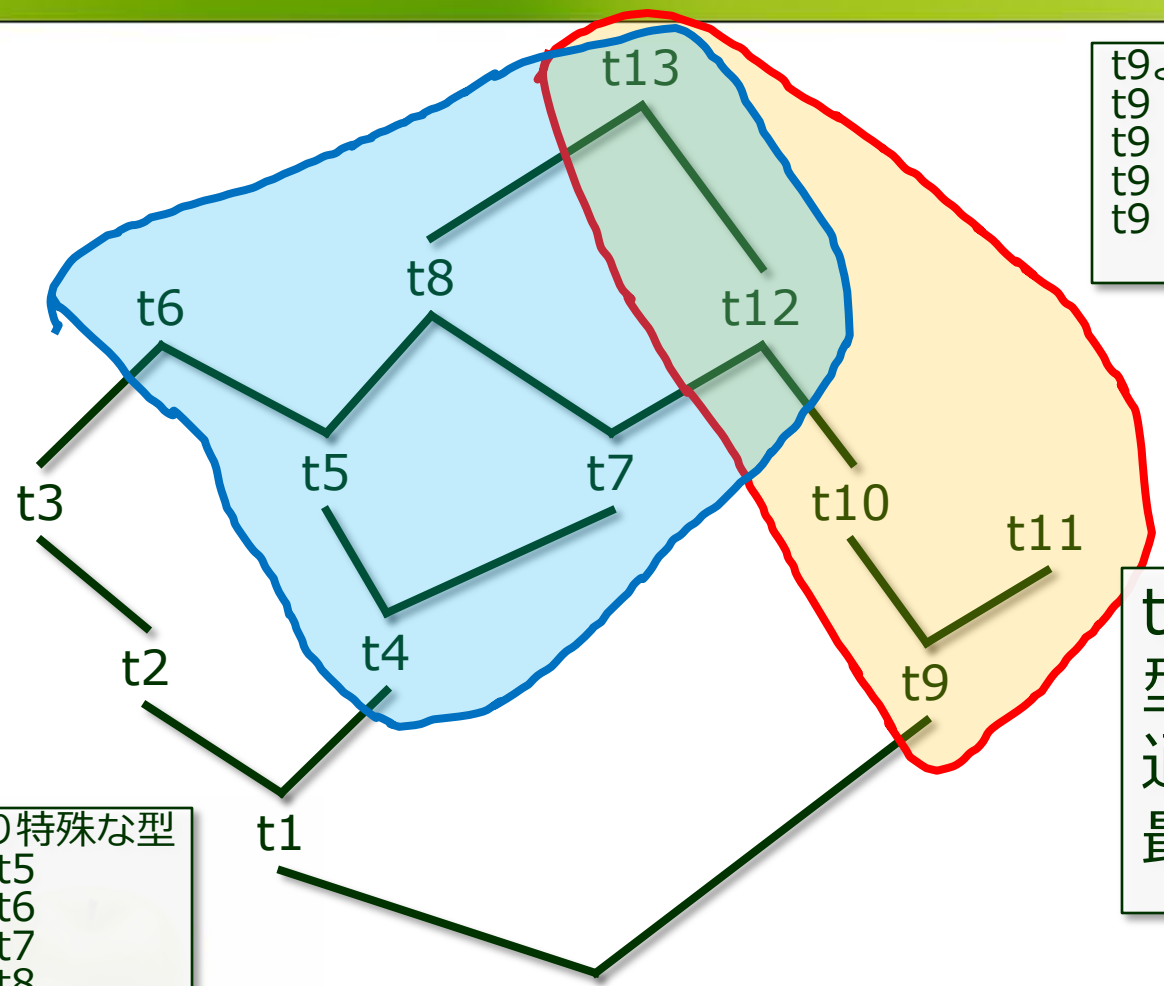
- データ構造
  - 型テーブル
  - ヒープ
  - ポインタ
- アルゴリズム
  - Unify(F, G)
    - 破壊的単一化アルゴリズム
    - F, G: ポインタ
    - 出力: F, Gのポインタは同じデータを指し、そのデータがF, Gの単一化の結果

型テーブル：型定義

- （復習）型階層



型テーブル:型の包摂関係



t9より特殊な型
t9 ⊏ t10
t9 ⊏ t11
t9 ⊏ t12
t9 ⊏ t13

特殊

t4 ⊔ t9=t12
型単一化の結果: 共通の特殊な型のうち最も一般な型

一般

t4より特殊な型
t4 ⊏ t5
t4 ⊏ t6
t4 ⊏ t7
t4 ⊏ t8
t4 ⊏ t12
t4 ⊏ t13

型テーブル

- 型定義は単一化実行前に与えられていると仮定
- 静的に全ての型の組み合わせに対する単一化を計算
  - t ⊔u=u ⊔tなので、表の半分だけ計算すれば良い
  - スパースになるケースが多いのでハッシュで実装することが多い

	⊥	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13
⊥	⊥	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13
t1		t1	t2	t3	t4	t5	t6	t7	t8	t12	t12	-	t12	t13
t2			t2	t3	t6	t6	t6	-	-	-	-	-	-	-
t3				t3	t6	t6	t6	-	-	-	-	-	-	-
t4					t4	t5	t6	t7	t8	t12	t12	-	t12	t13
t5						t5	t6	t8	t8	t13	t13	-	t13	t13
t6							t6	-	-	-	-	-	-	-
t7								t7	t8	t12	t12	-	t12	t13
t8									t8	t13	t13	-	t13	t13
t9										t9	t10	t11	t12	t13
t10											t10	-	t12	t13
t11												t11	-	-
t12													t12	t13
t13														t13

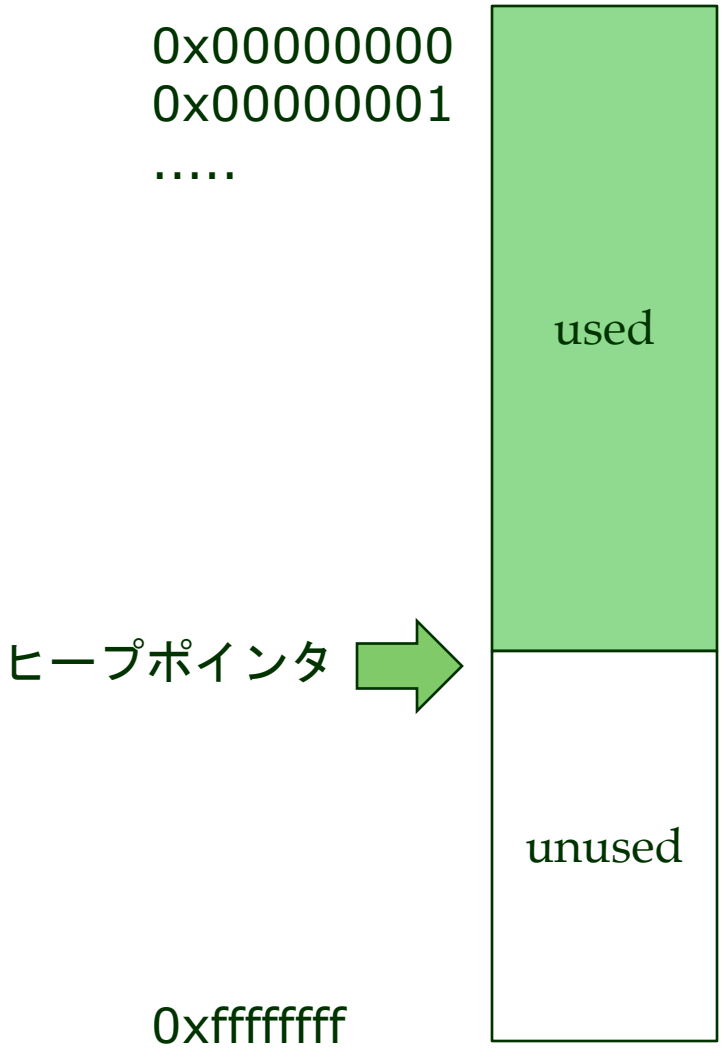
ヒープとセル

- ヒープ
  - セルの列
- セル
  - タグとデータのペア

アドレス	タグ	データ
0x00000000	VAR	a
0x00000001	STR	e
0x00000002	VAR	a
0x00000003	INT	15
0x00000004	FLO	0.335
0x00000005	STR	f
0x00000006	PTR	0x00000003
0x00000007	STG	“Taro”
…	…	…

ヒープとヒープポインタ

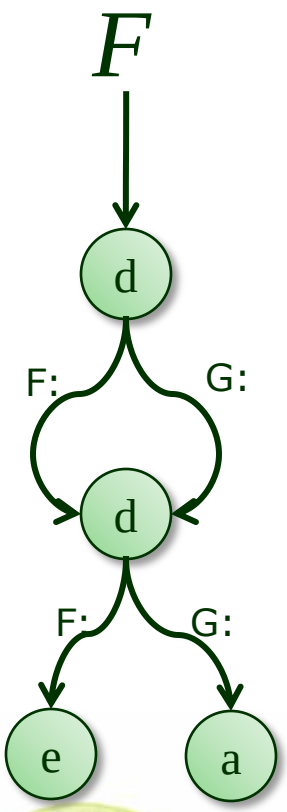
- ヒープには使用領域と未使用領域の二つがある
  - 新しいデータは未使用領域に追加される
- ヒープポインタ
  - 未使用領域の先頭アドレス（＝使用領域の末尾アドレス＋１）を格納した変数



タグの種類

- 重要なタグ
  - VAR: 枝をまだ持っていないノード。データ部は整数の型ID。
  - STR: 枝を持つノード。データ部は整数の型ID。
  - PTR: ポインタ。データ部はアドレス。
- 特殊なデータのためのタグ
  - INT: 整数(integer)
  - FLO: 浮動小数
  - STG: 文字列

ヒープ上の素性構造

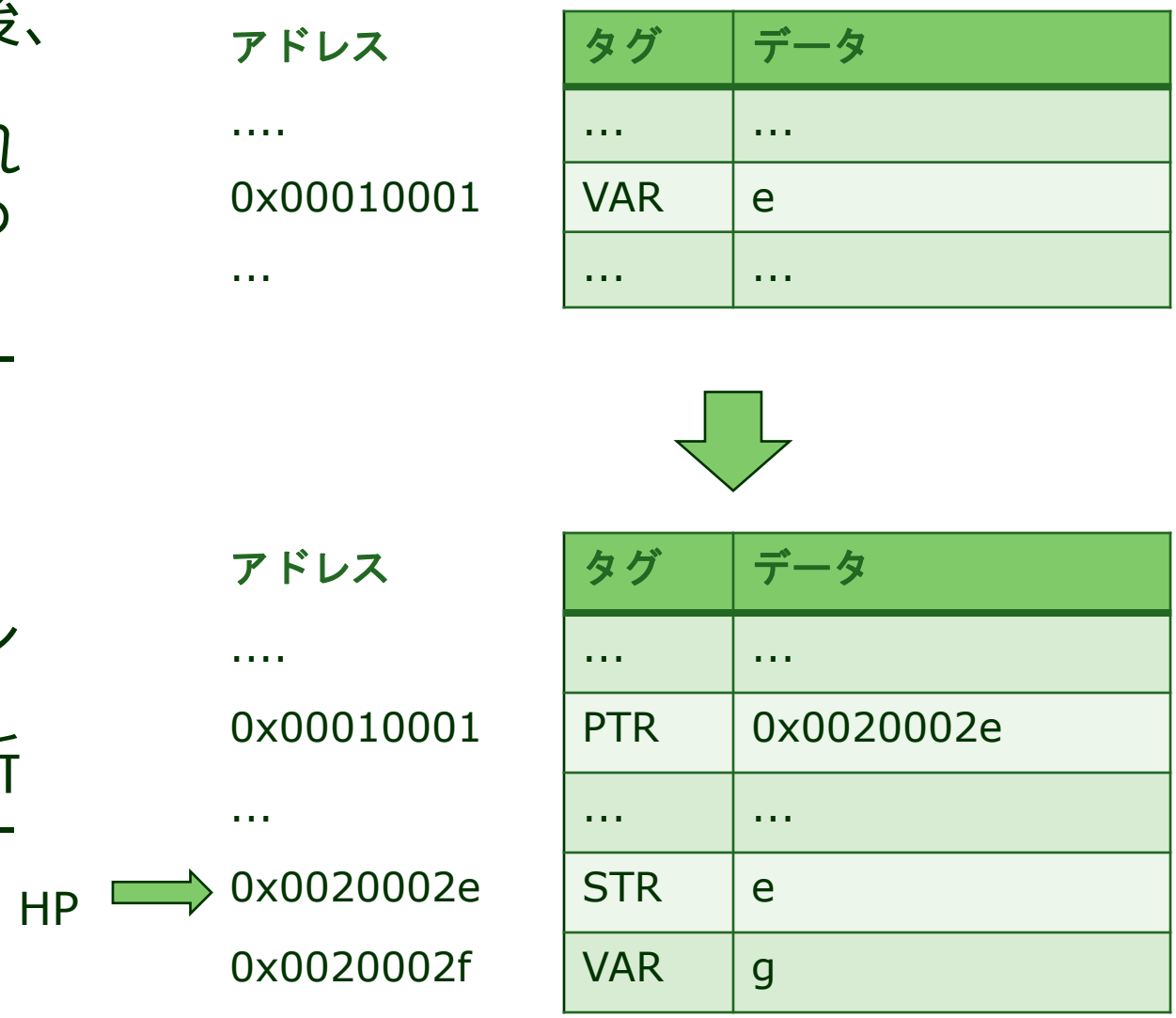


F ⇨

アドレス	タグ	データ
0x00010000	STR	d
0x00010001	VAR	e
0x00010002	VAR	a
0x00010003	STR	d
0x00010004	PTR	0x00010000
0x00010005	PTR	0x00010000
0x00010006		
0x00010007		
…	…	…

VAR

- もしかしたら今後、枝(素性)を持つようになるかもしれないけど今は持っていないノード
- データ部にはノードの型のID (整数値)を格納
- 枝を持つようになったら、ポインタで上書きして、ポインタの先に新しい枝つきのデータを作成



STR

- 枝(素性)つきノード
  - データ部にはノードの型のID(整数)
- 後続するアドレスのセルにはそのノードが持つ素性の値が入る
- 素性の順番はSTRタグを持ったノードの型と素性によって静的に決定しておく Index(f, t)=1, 2, 3,...,n_t
  - 例えば、素性にIDをつけて、型tがもつ素性集合のうちIDの昇順に並べる、など。型tが素性fを持たない時は0を返すようにすると便利
- 新しくSTRタグのデータをつくるときは、その素性値にはデフォルト値として、appropriateな型を格納する Approp(f, t)

アドレス	タグ	データ
....	...	...
0x00010001	STR	e
0x00010002	VAR	g
0x00010003	PTR	0x0001023f
....	...	...

PTR

- 実データを指すポインタ
- ポインタはチェーンになっていても構わない
  - 手続き型言語でよくあるポインタのポインタといった複雑な構造ではないので注意！

矢印が指すアドレスの素性構造は全部同じ(0x00010007のデータ)

アドレス
….
0x00010001
0x00010002
0x00010003
0x00010004
0x00010005
0x00010006
0x00010007
….

タグ	データ
…	…
PTR	0x00010005
STR	f
VAR	g
PTR	0x00010006
PTR	0x00010004
PTR	0x00010007
VAR	h
…	…

Deref

- ポインタを辿って実データのアドレスを得る操作

```
Deref( p )                          ## アドレス p
  while(Heap[p].tag = PTR )   ## タグがPTR
      p := Heap[p].data           ## pをアドレスpのセルのデータで置き換える
return p
```

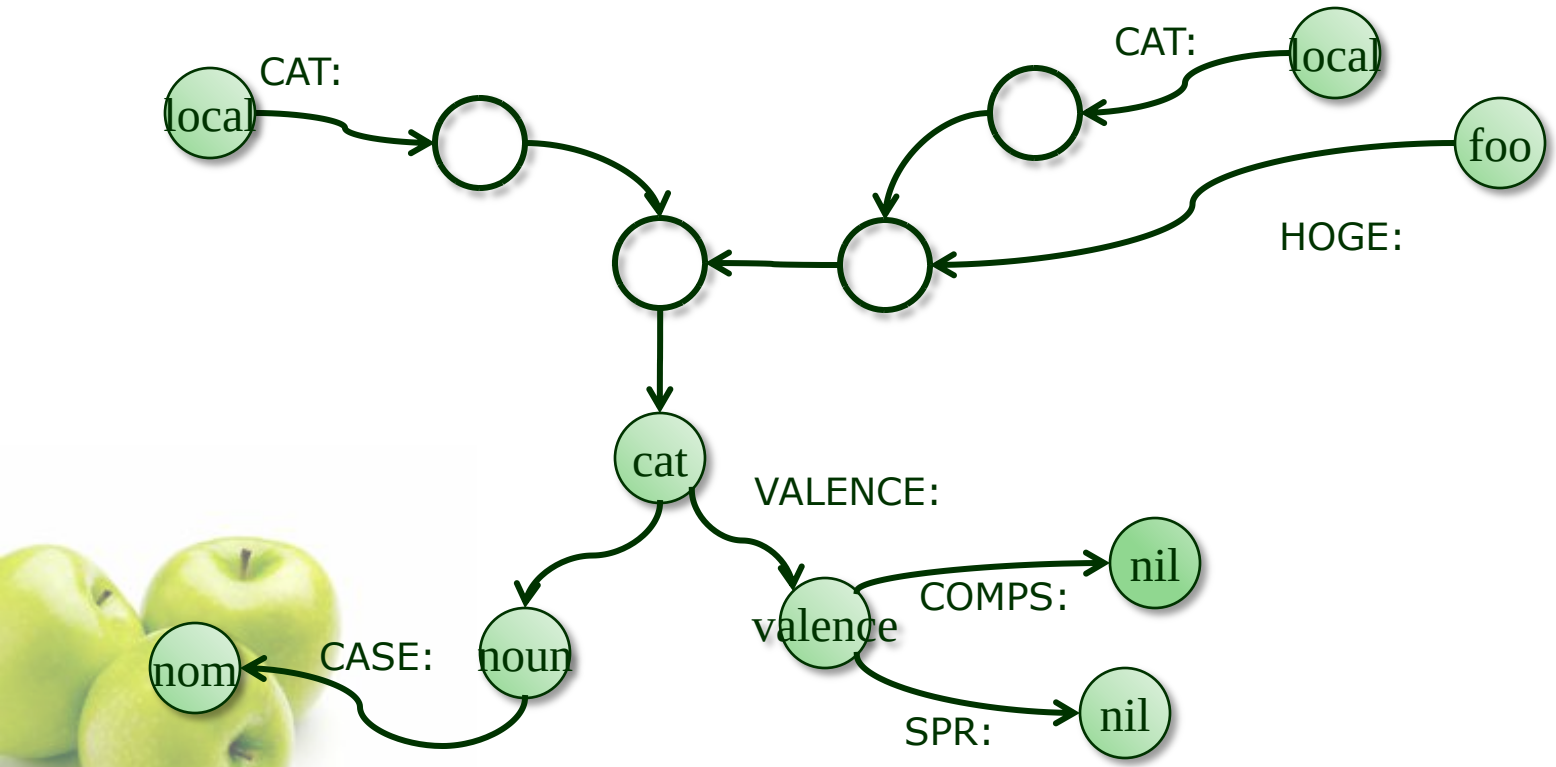
Deref(0x00010001)
= 0x00010007

0x00010001	PTR
0x00010002	STR
0x00010003	VAR
0x00010004	PTR
0x00010005	PTR
0x00010006	PTR
0x00010007	VAR

0x00010005	
f	
g	
0x00010006	
0x00010004	
0x00010007	
h	

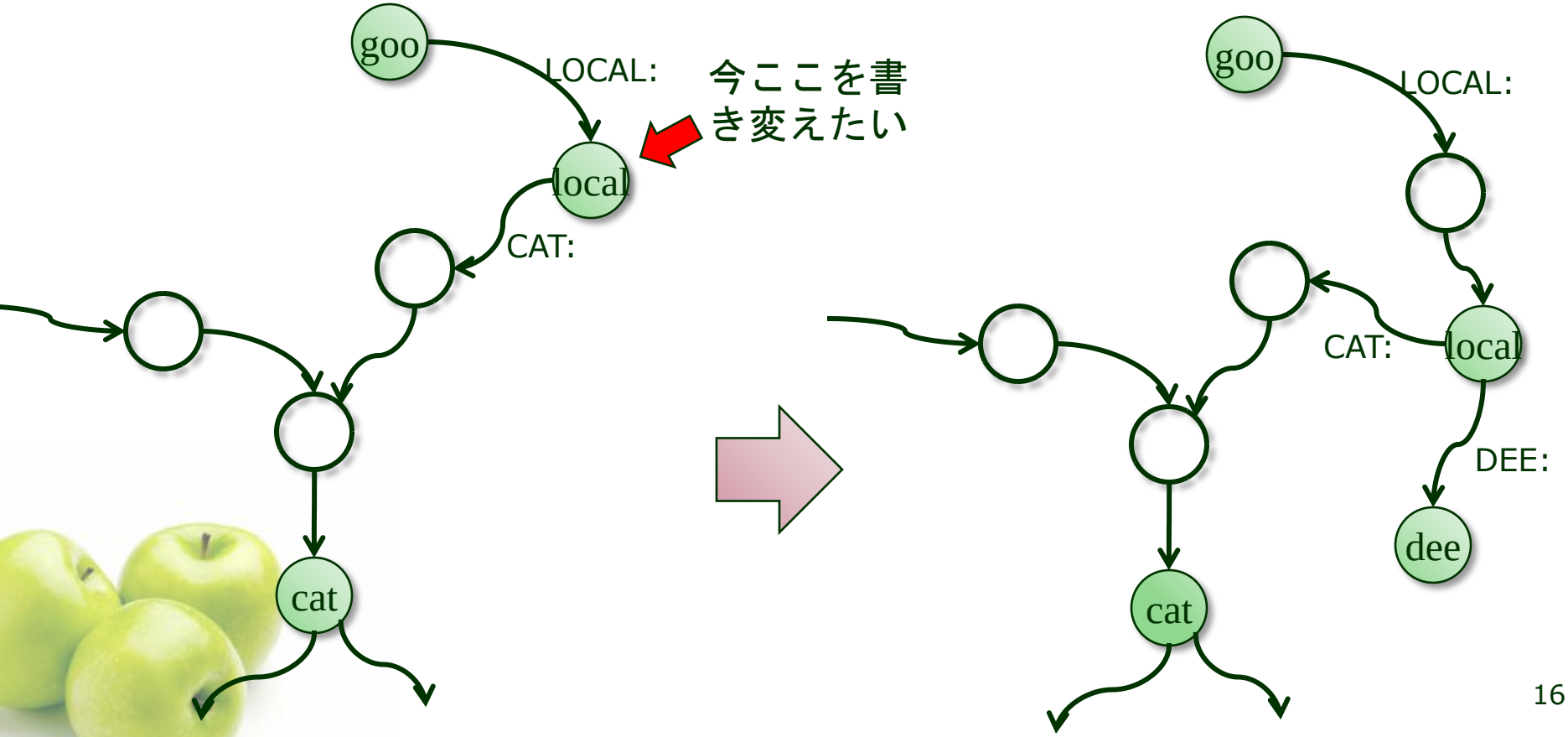
ポインタに対する考え方

- 同じノードを指すポインタの集合



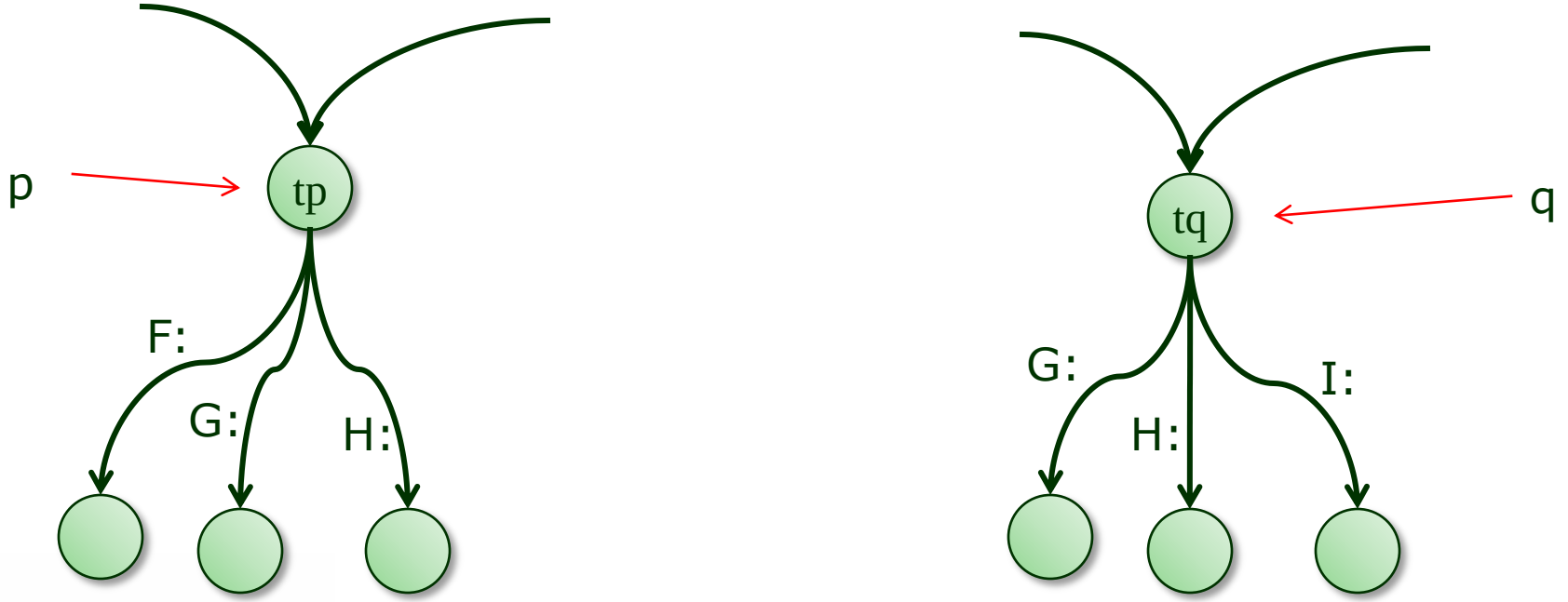
ポインタに対する考え方

- どこから指されているのか気にせず実データの書き換えを簡単に行える



単一化アルゴリズム(1/8)

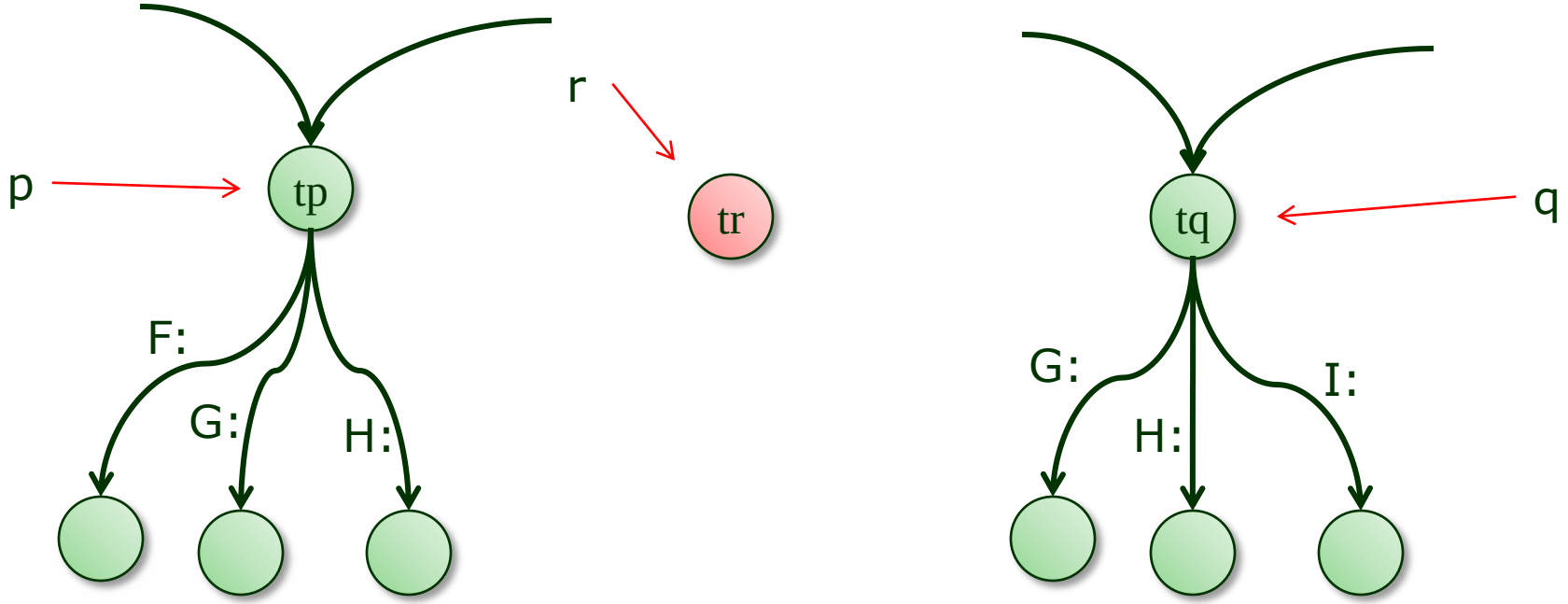
- Unify(p, q): アドレス pとアドレス qの素性構造を単一化



もしp=qなら終了

単一化アルゴリズム(2/8)

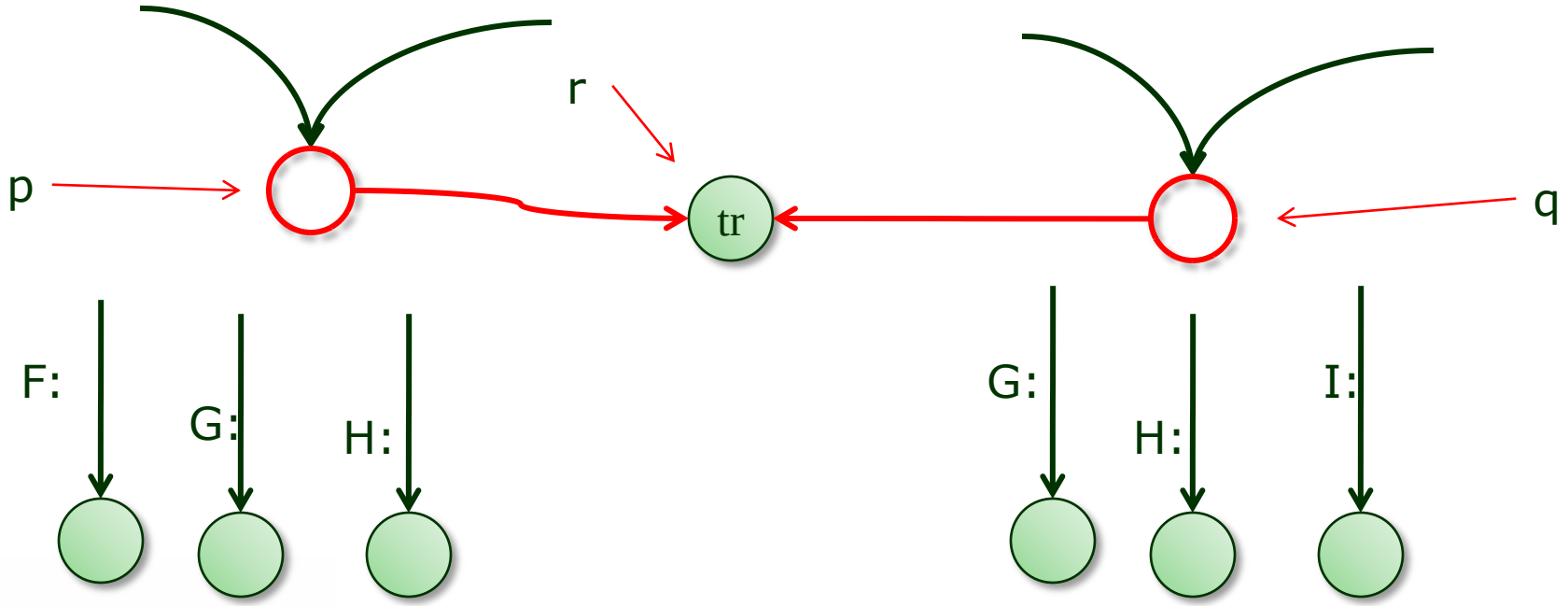
- Unify(p, q): アドレス pとアドレス qの素性構造を単一化



新しいノードを生成
ノードの型trはtr = tp ⊔ tq

単一化アルゴリズム(3/8)

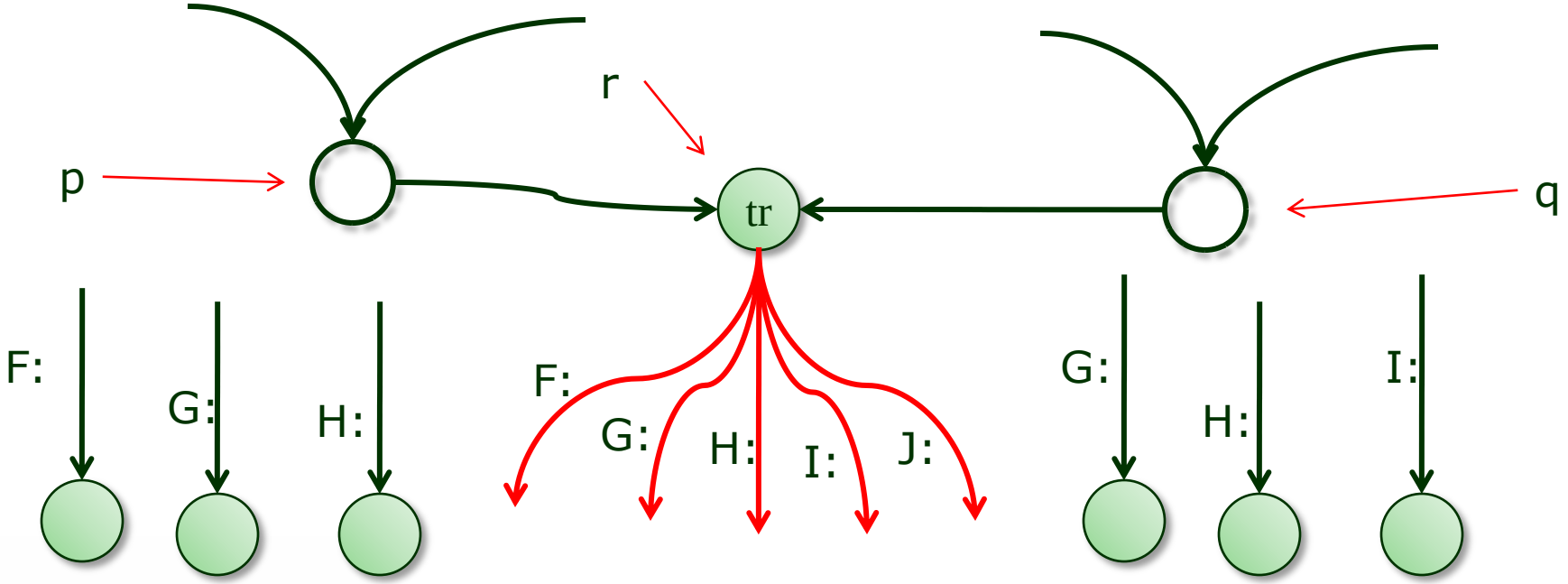
- Unify(p, q): アドレス pとアドレス qの素性構造を単一化



pとqをrを指すポインタに書換

単一化アルゴリズム(4/8)

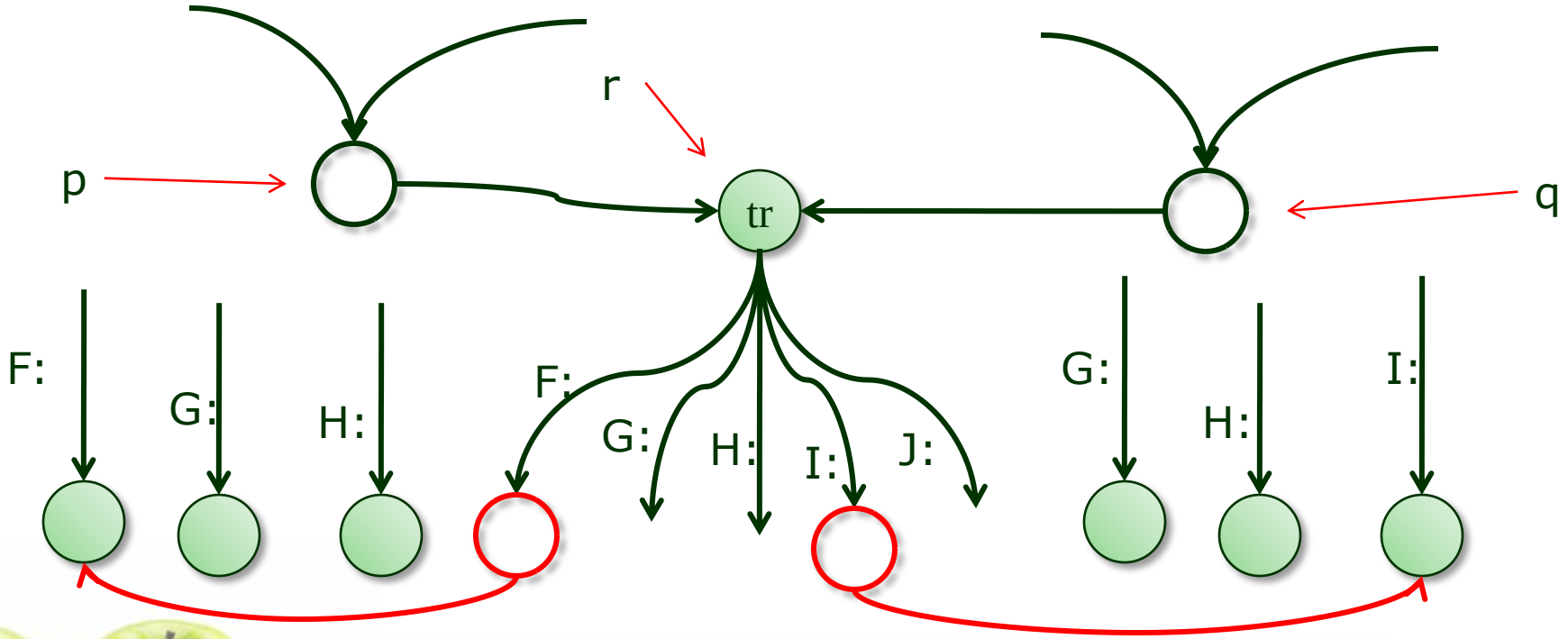
- Unify(p, q): アドレス pとアドレス qの素性構造を単一化



新しいノードの型trが持つ素性に対する枝を作る

単一化アルゴリズム(5/8)

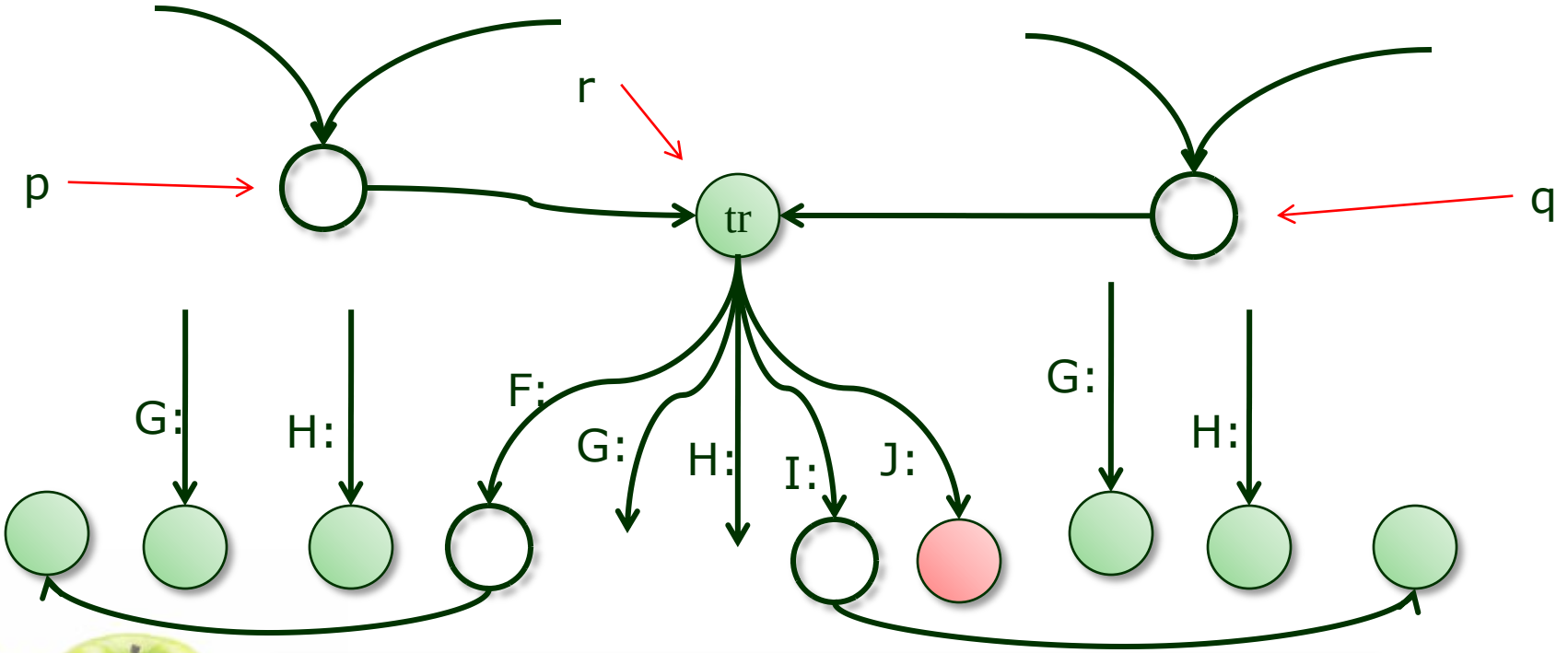
- Unify(p, q): アドレス pとアドレス qの素性構造を単一化



共通でない素性には単純にポインタをはる

単一化アルゴリズム(6/8)

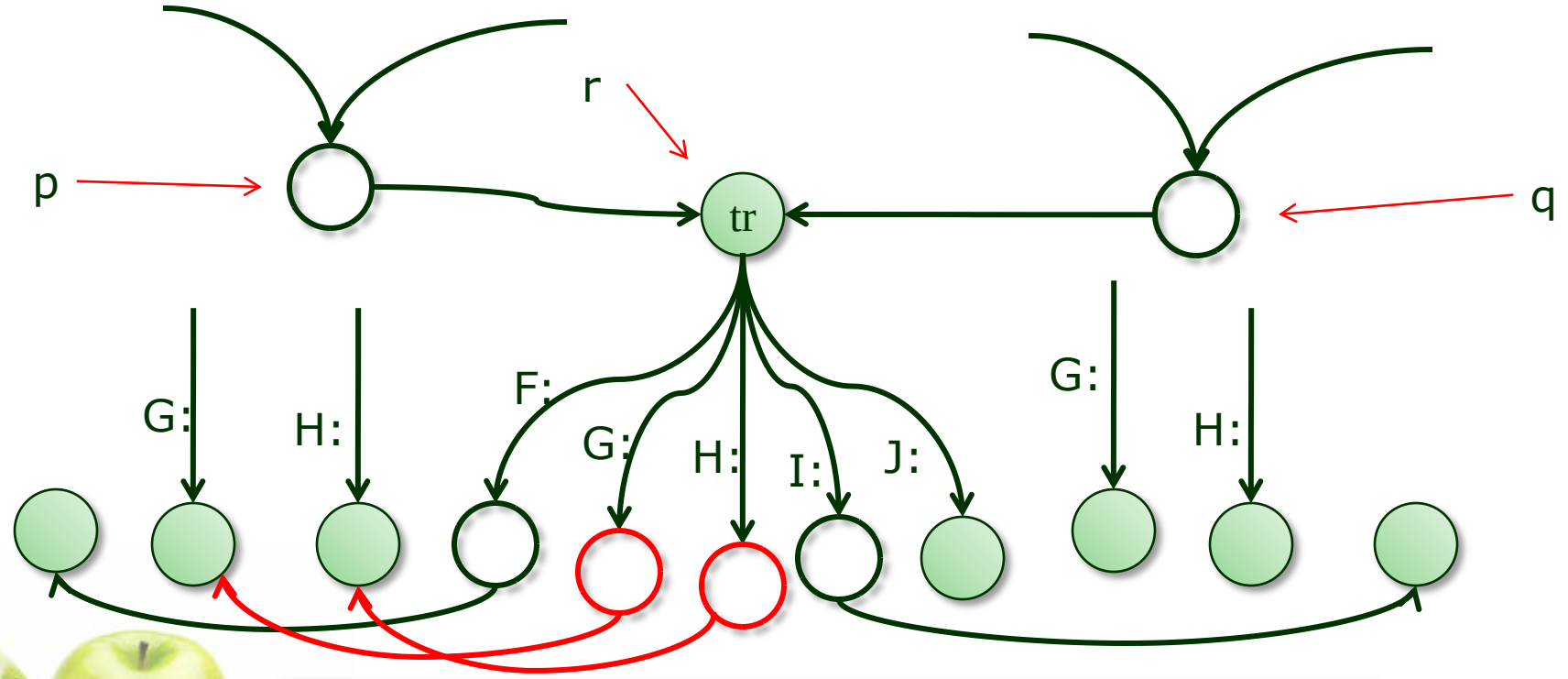
- Unify(p, q): アドレス pとアドレス qの素性構造を単一化



tpにもtqにもない新しい素性J:には
Appropriateな型(trと素性J:に対して定義
されるデフォルトの型）のノードを生成

単一化アルゴリズム(7/8)

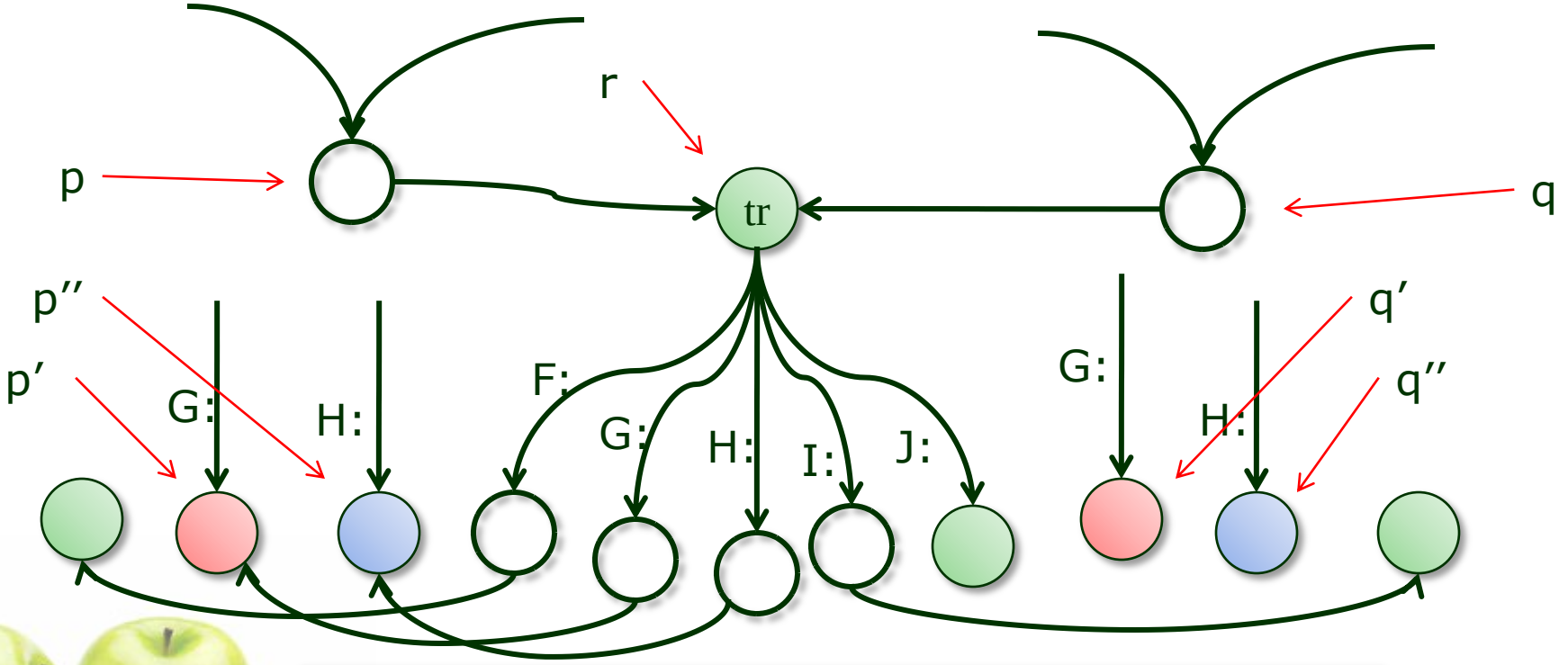
- Unify(p, q): アドレス pとアドレス qの素性構造を単一化



共通の素性G: H: に対しては、とりあえず、pかqのどちらかの素性G:H:が指すノードに対し、ポインタを貼る

単一化アルゴリズム(8/8)

- Unify(p, q): アドレス pとアドレス qの素性構造を単一化



Unify(p', q')とUnify(p'', q'')を再帰呼び出し

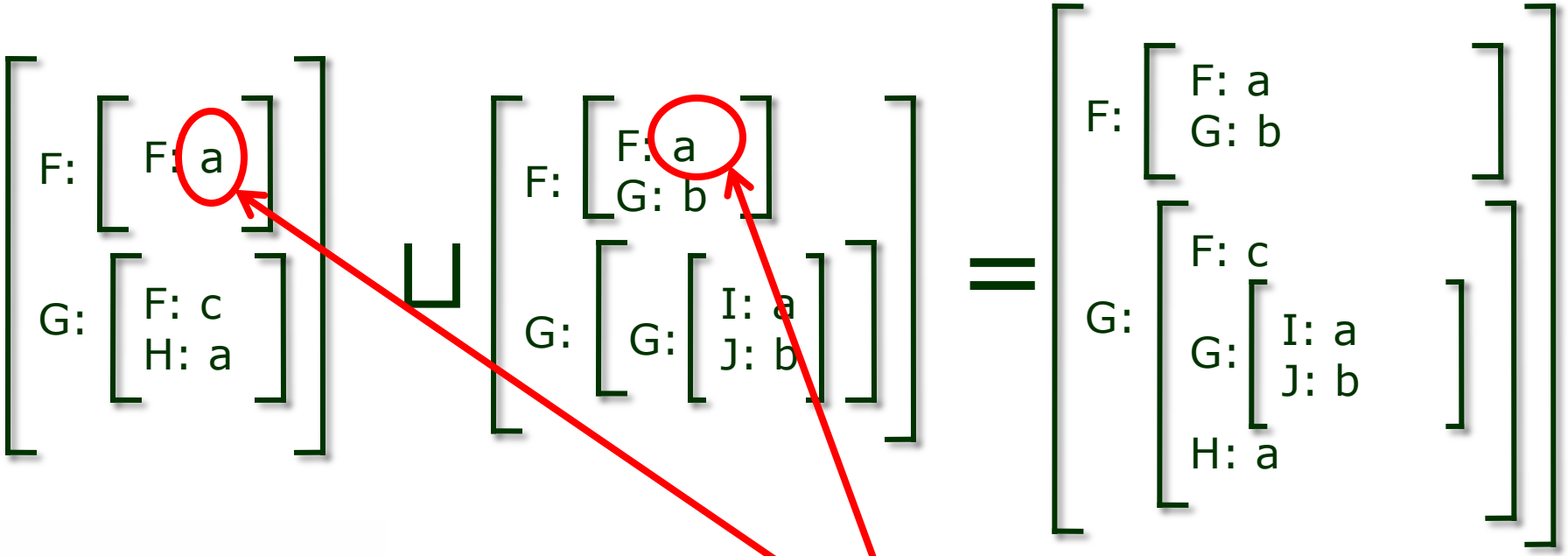
単一化アルゴリズム: まとめ

```
Unify(p, q)  # pとqはヒープ上のアドレス
   p := Deref(p); q := Deref(q);
   if( p = q )  return true;
   r := HP; HP := HP + 1;
   tagp := Heap[p].tag; tp := Heap[p].data;
   tagq: = Heap[q].tag; tq := Heap[q].data;
   tr :=Type-Unify(tp, tq);   if (tr is not defined ) return false;
   Heap[p].tag := PTR; Heap[p].data := r;
   Heap[q].tag := PTR; Heap[q].data := r;
   Heap[r].tag := STR;  Heap[r].data := tr;
   HP := HP + |Features(tr)|;
   foreach f ∈ Features(tr)
        if( f ∈ Features(tp)  ∧ tagp = STR )
             Heap[r + index(f, tr)].tag := PTR;  Heap[r + index(f, tr)].data := p + index(f, tp)
        else if( f ∈ Features(tq) ∧ tagq = STR )
             Heap[r + index(f, tr)].tag := PTR;  Heap[r + index(f, tr)].data := q + index(f, tq)
        else
             Heap[r + index(f, tr)].tag := VAR; Heap[r + index(f, tr)].data := Approp(f, tr)
   foreach f ∈ (Features(tp) ∩ Features(tq))
        if( ¬Unify(p + index(f, tp), q + index(f, tq)) ) return false;
   return true;
```

単一化アルゴリズムの特徴

- 素性構造にサイクルがあってもok
- 必ず終了する
  - Unifyが再帰呼び出しされる度に実体ノード（ポインタ以外のノード）が一つずつ減る
    - appropriateな型のノードが生成される時のみVARノードが増えるが、このノードを無視すれば、各ステップでノードが一つずつ減る
    - VARノードを無視しても良い大雑把な説明
      - アルゴリズムを改良して、タグの組み合わせで処理を分類すると、各ステップでSTRノードの数が増えないようにすることができる
        - VARとVAR→VAR (STRノード数は減らない)
        - VARとSTR→STR (STRノード数は減らない)
        - STRとSTR→STR(STRノード数がひとつ減る)
      - 各STRノードが持つことができるVARノードは全ての型に対する最大素性数で抑えられるため、実ノード数をSTRノード数×最大素性数と考えれば、この数は各ステップごとに一つずつ減るので、必ず終了する

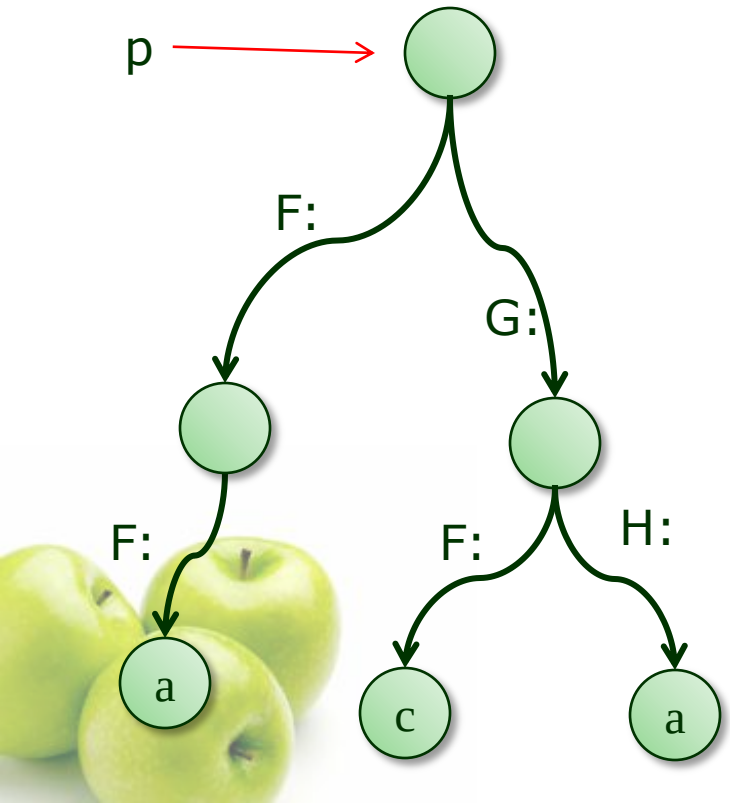
単一化の例 (1)

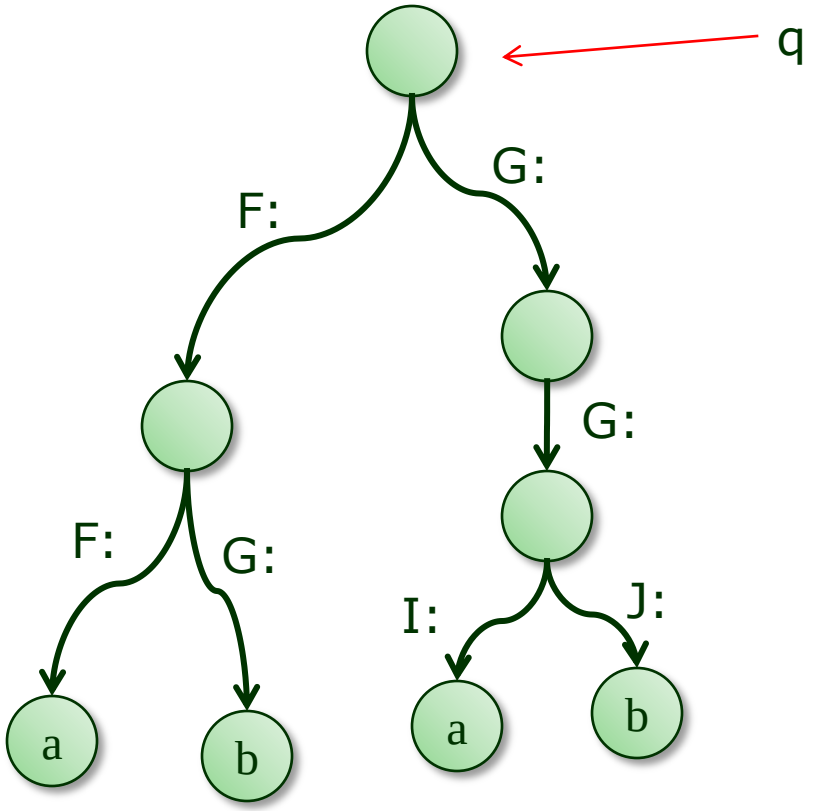


異なる型の場合は、型単一化を行う。型単一化に失敗すると、全体の単一化も失敗

単一化の例 (1)

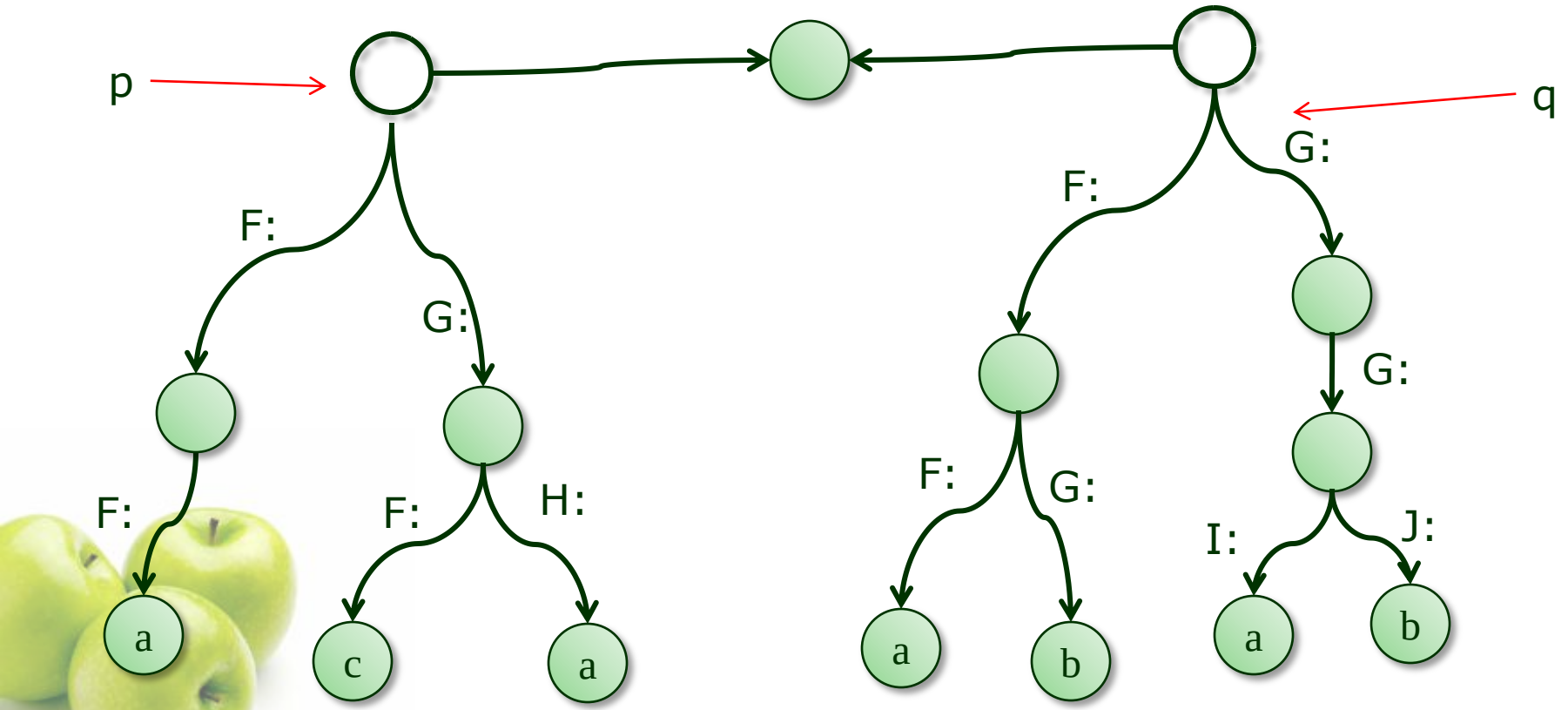
- Unify(p, q)





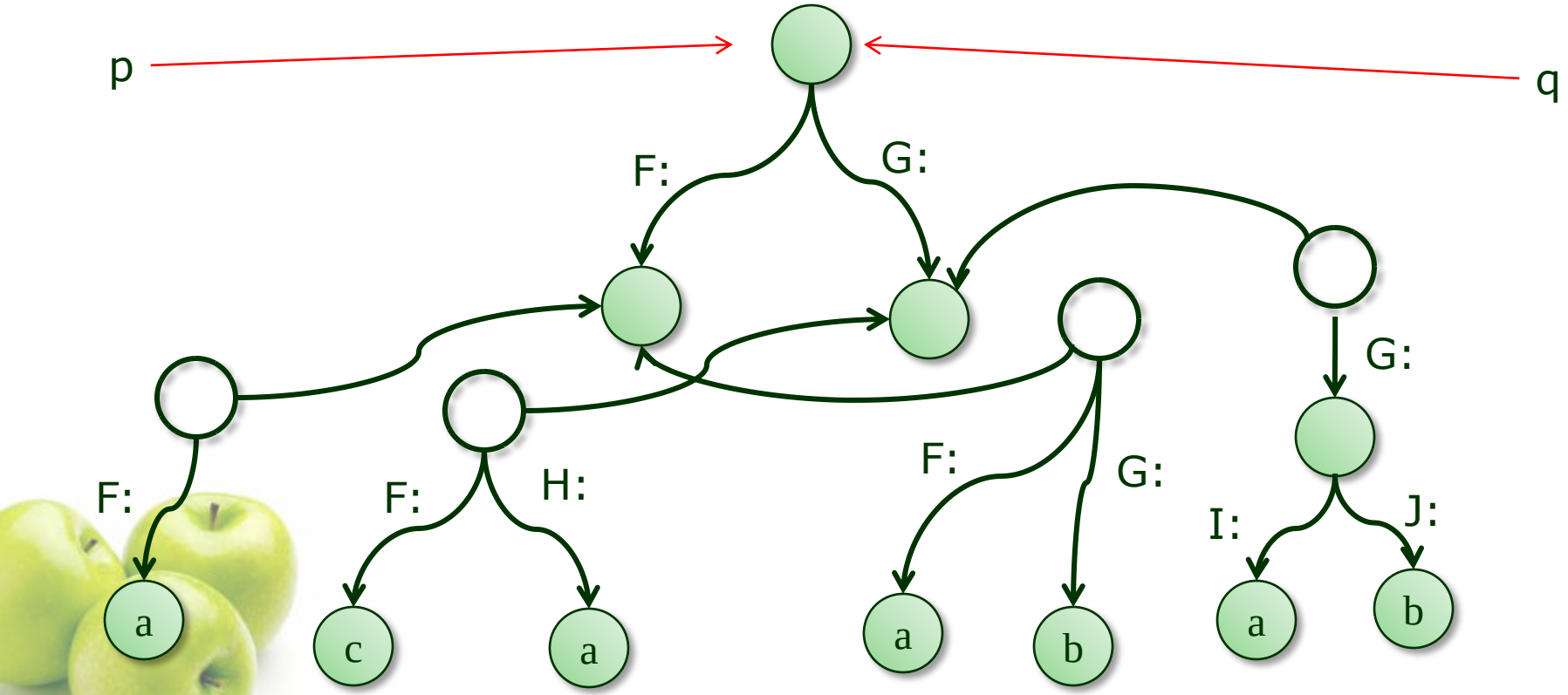
単一化の例 (1)

- Unify(p, q)



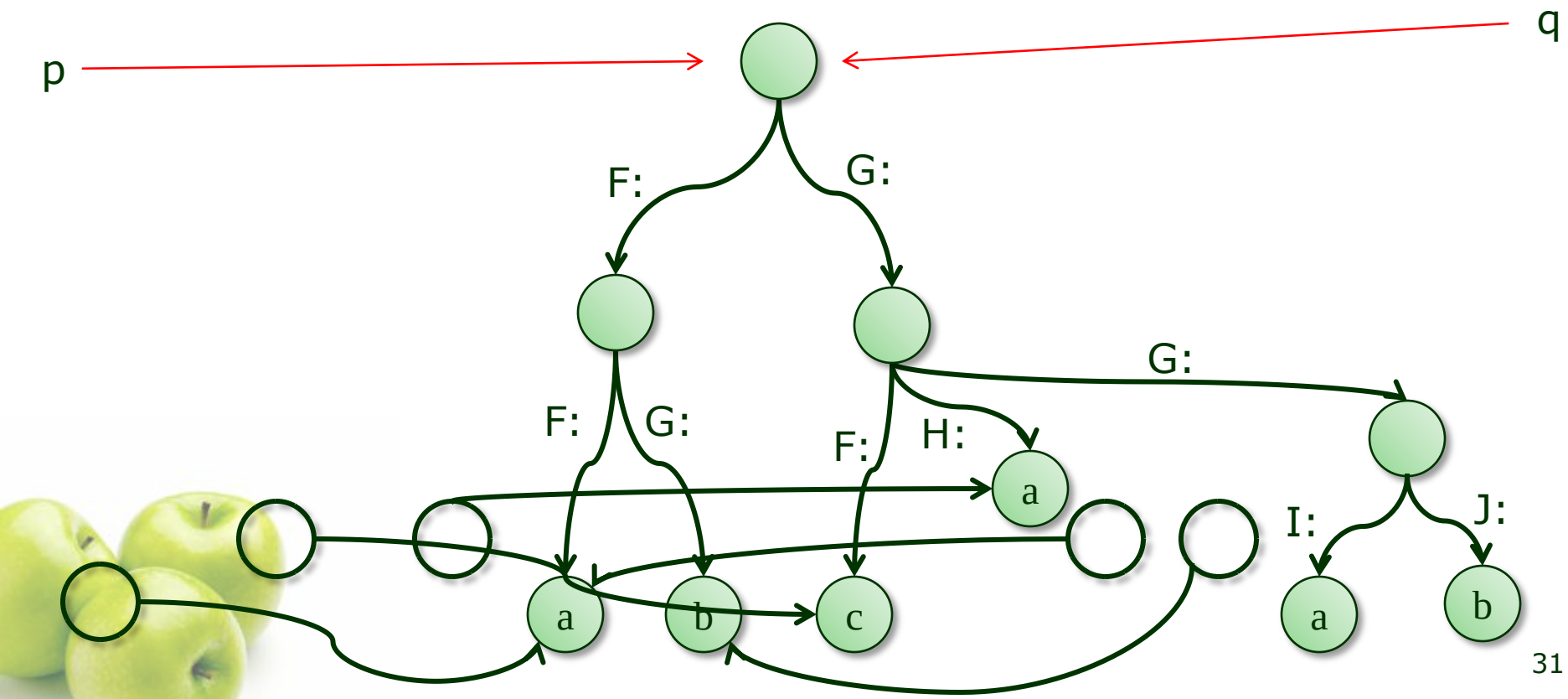
単一化の例 (1)

- Unify(p, q)

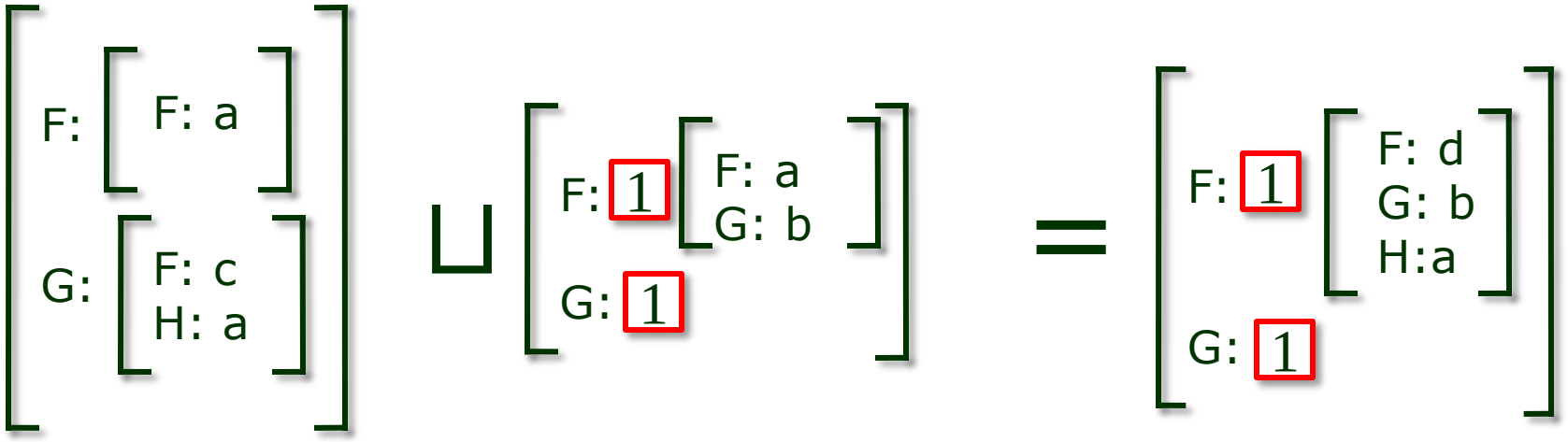


単一化の例 (1)

- Unify(p, q)



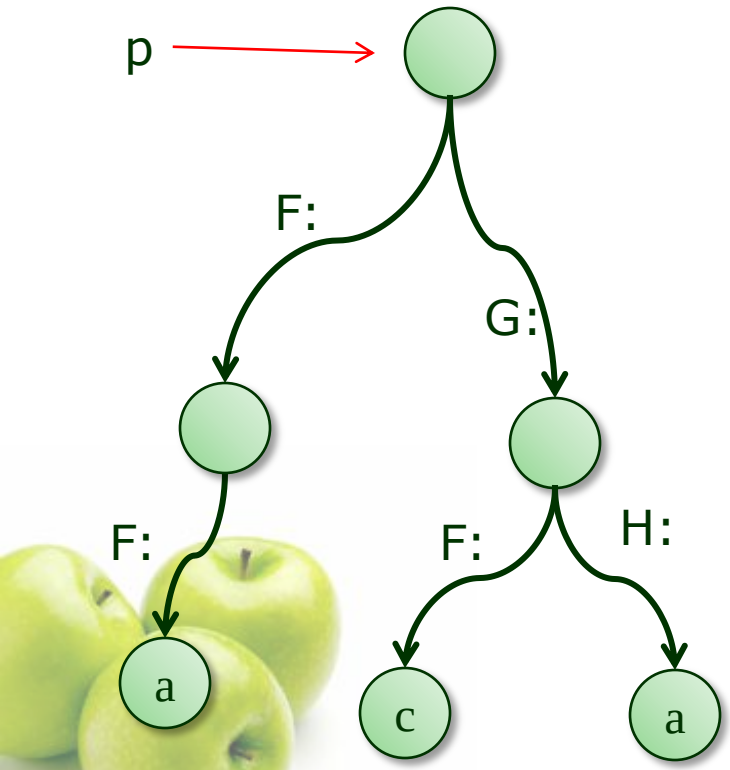
単一化の例 (2)

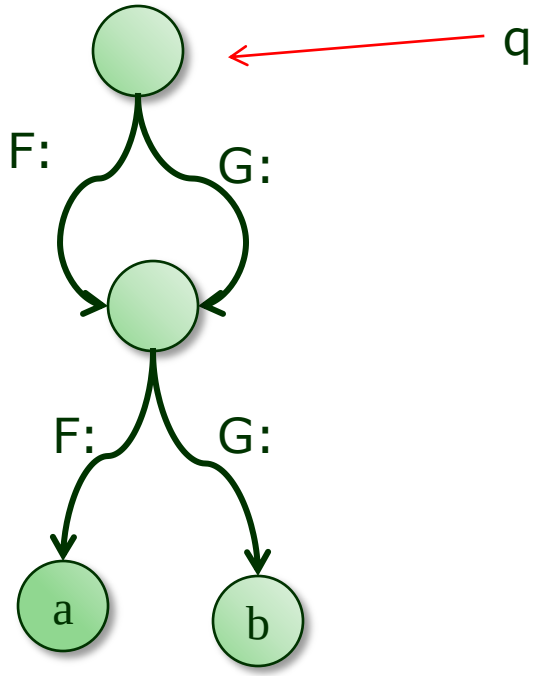


(a⊔c=dとする)

単一化の例 (2)

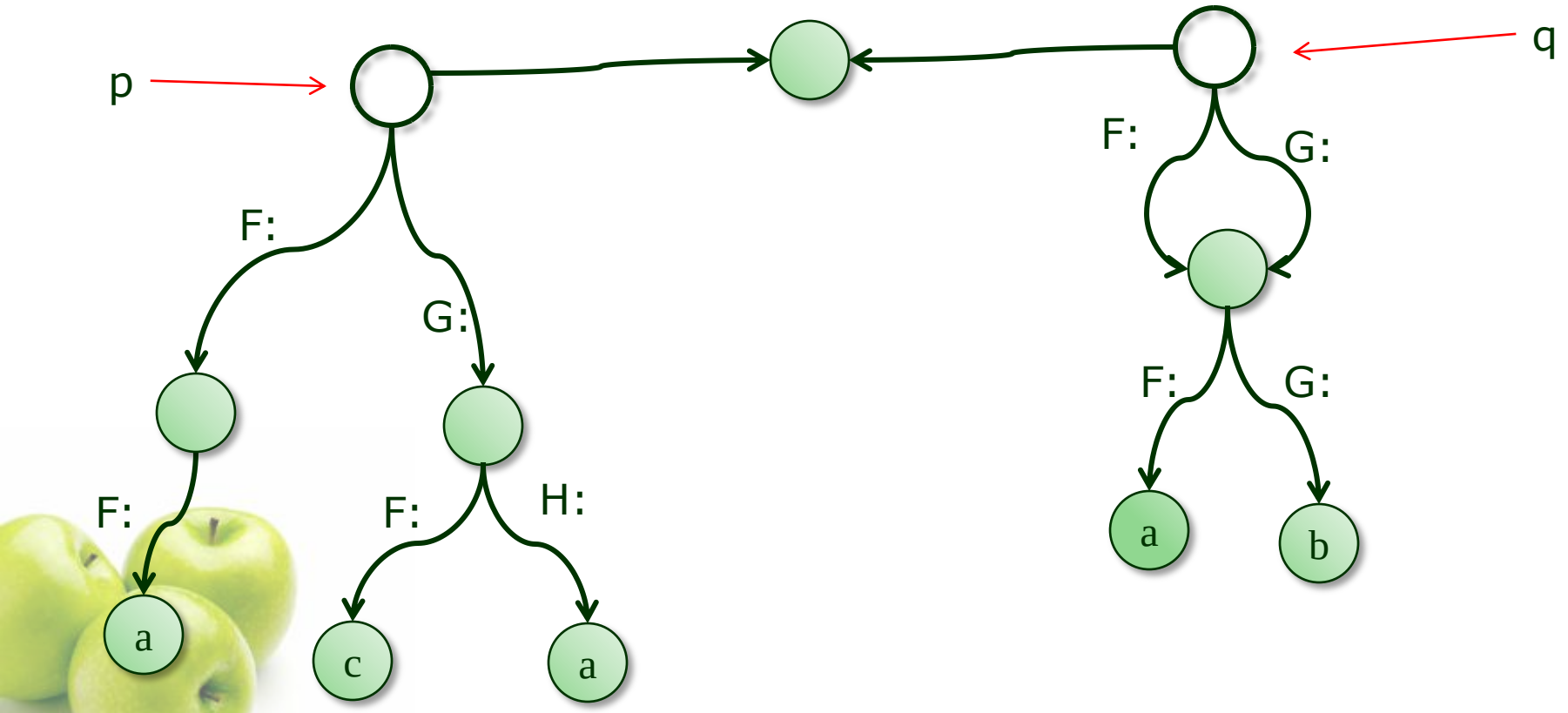
- Unify(p, q)





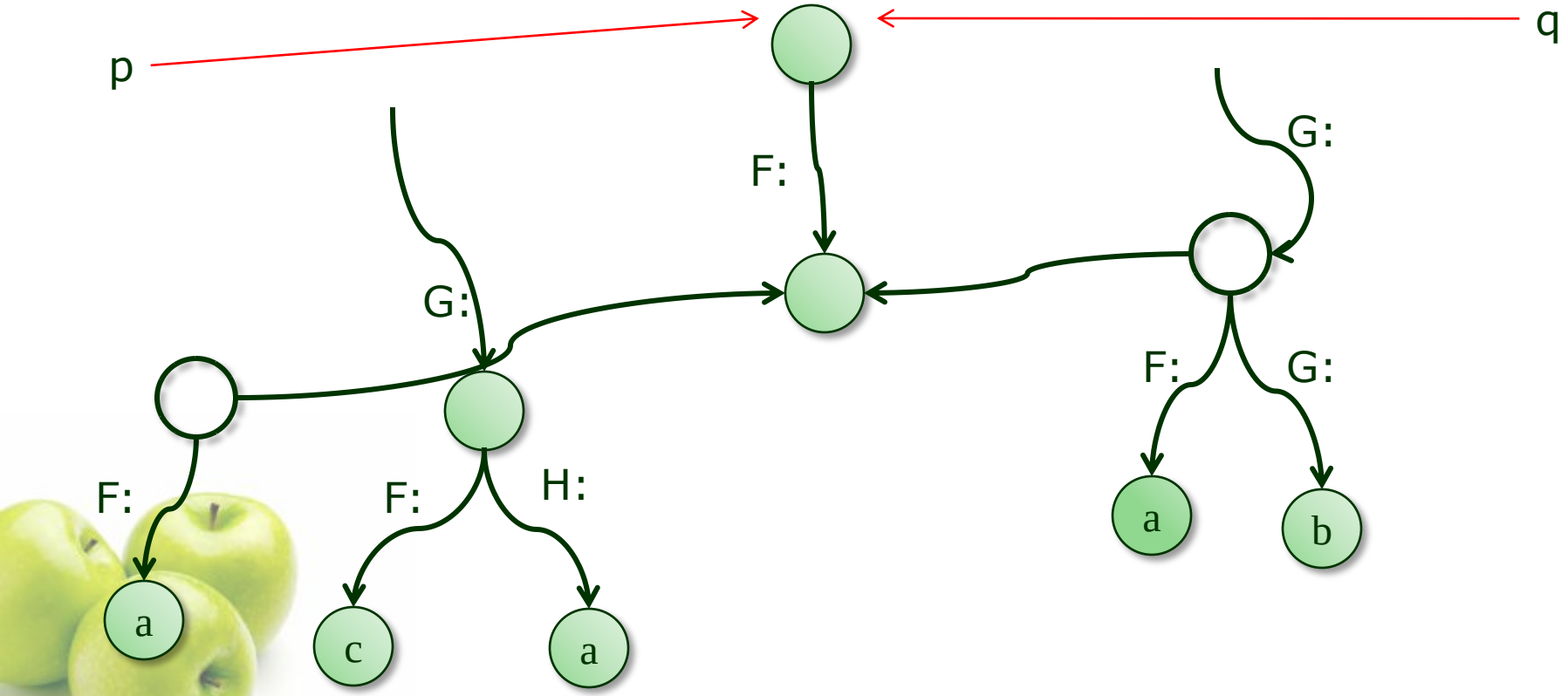
単一化の例 (2)

- Unify(p, q)



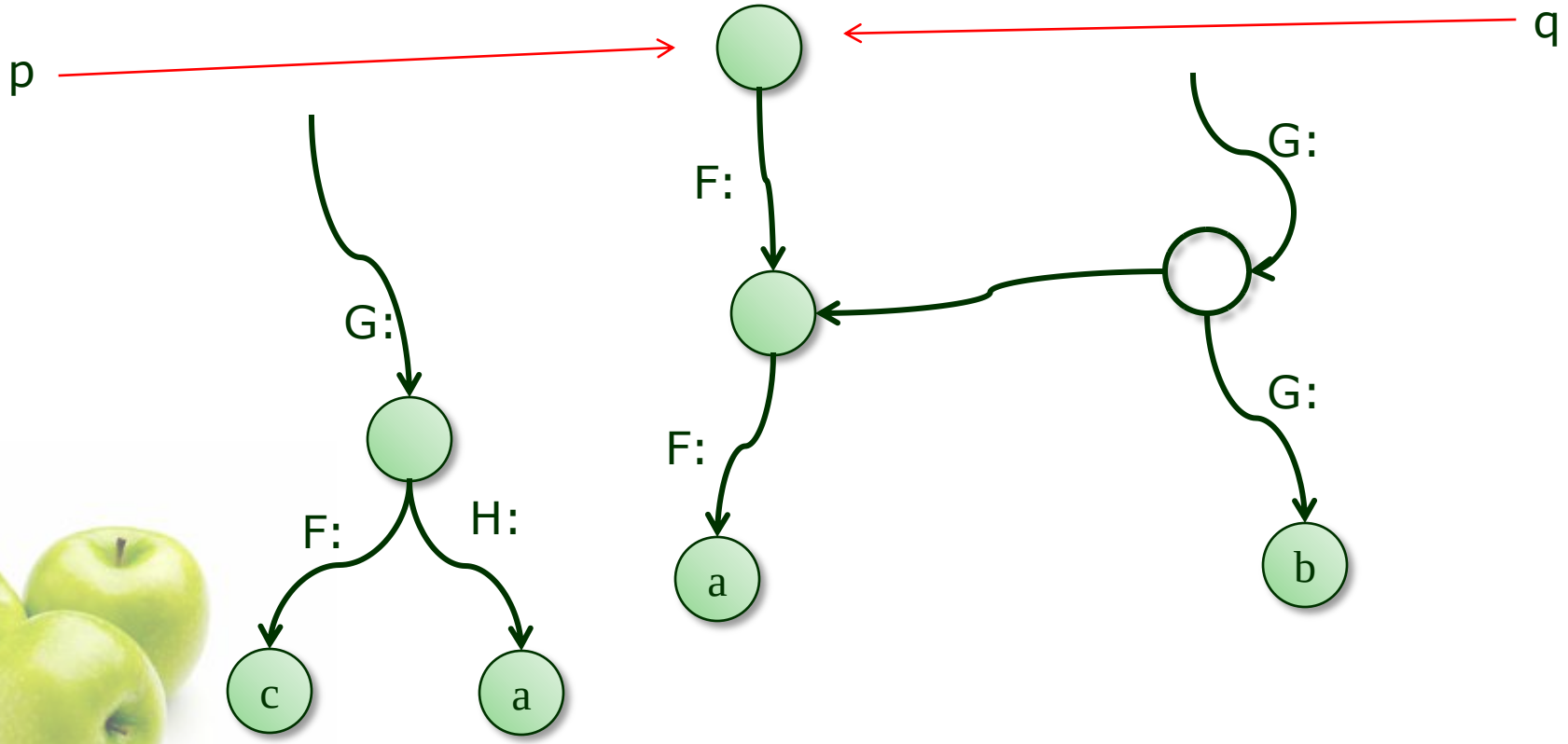
単一化の例 (2)

- Unify(p, q)



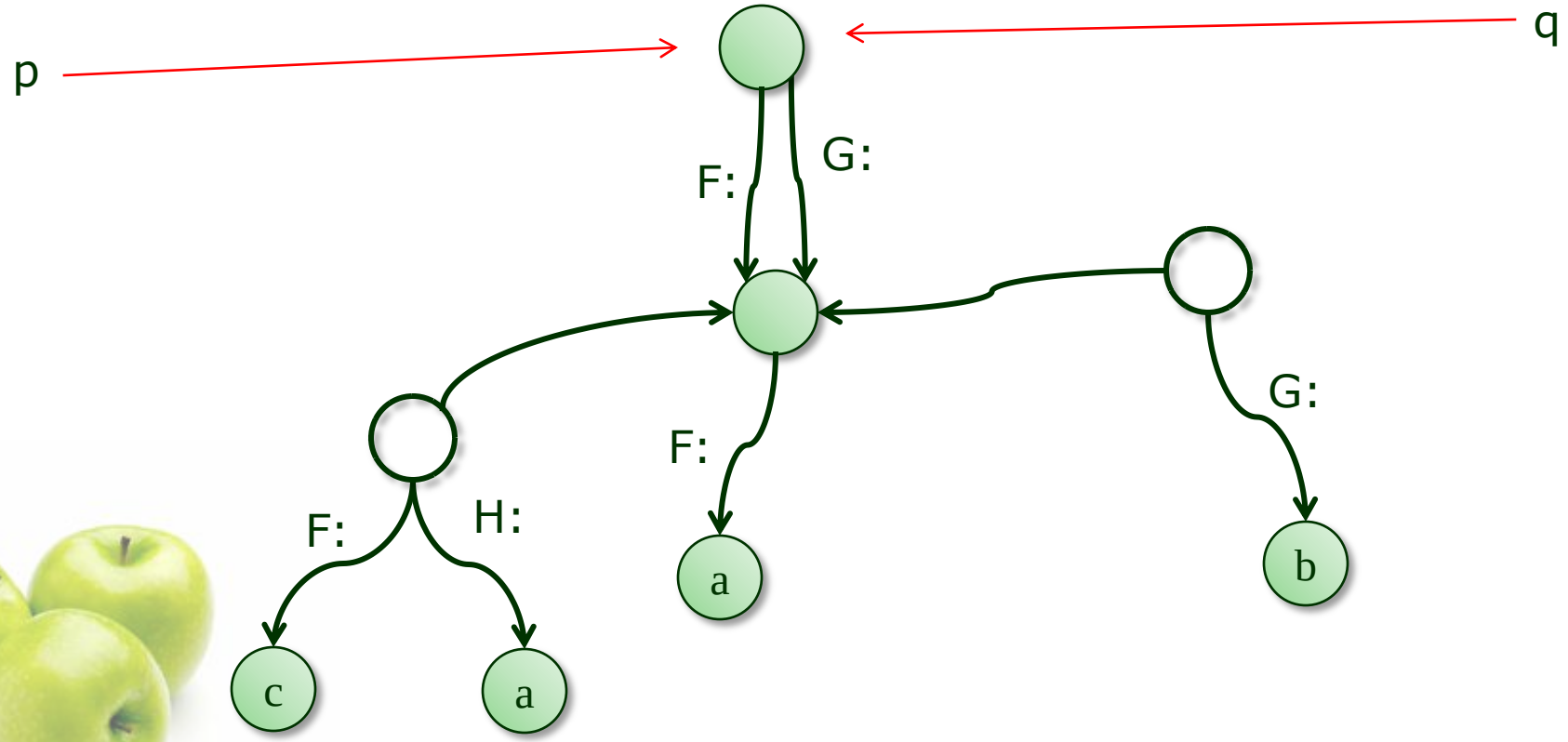
単一化の例 (2)

- Unify(p, q)



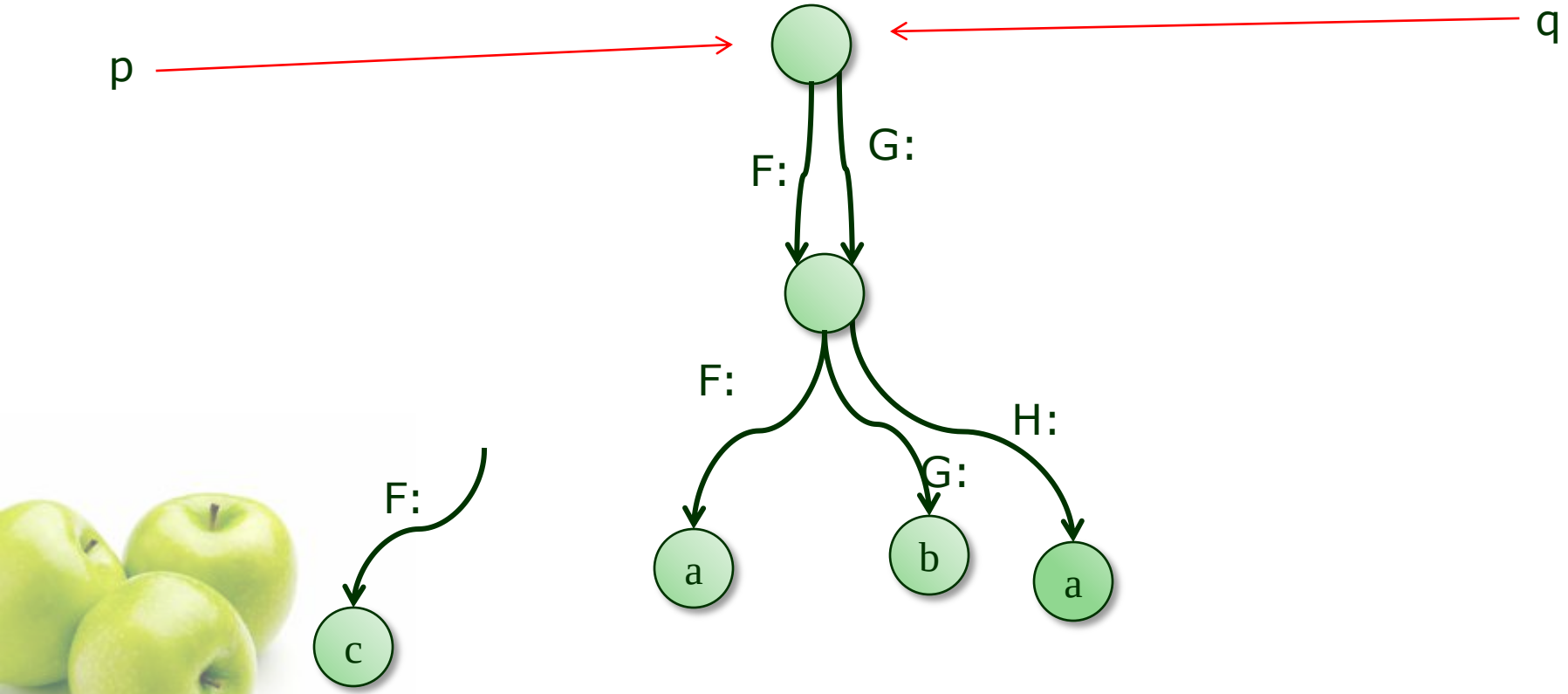
単一化の例 (2)

- Unify(p, q)



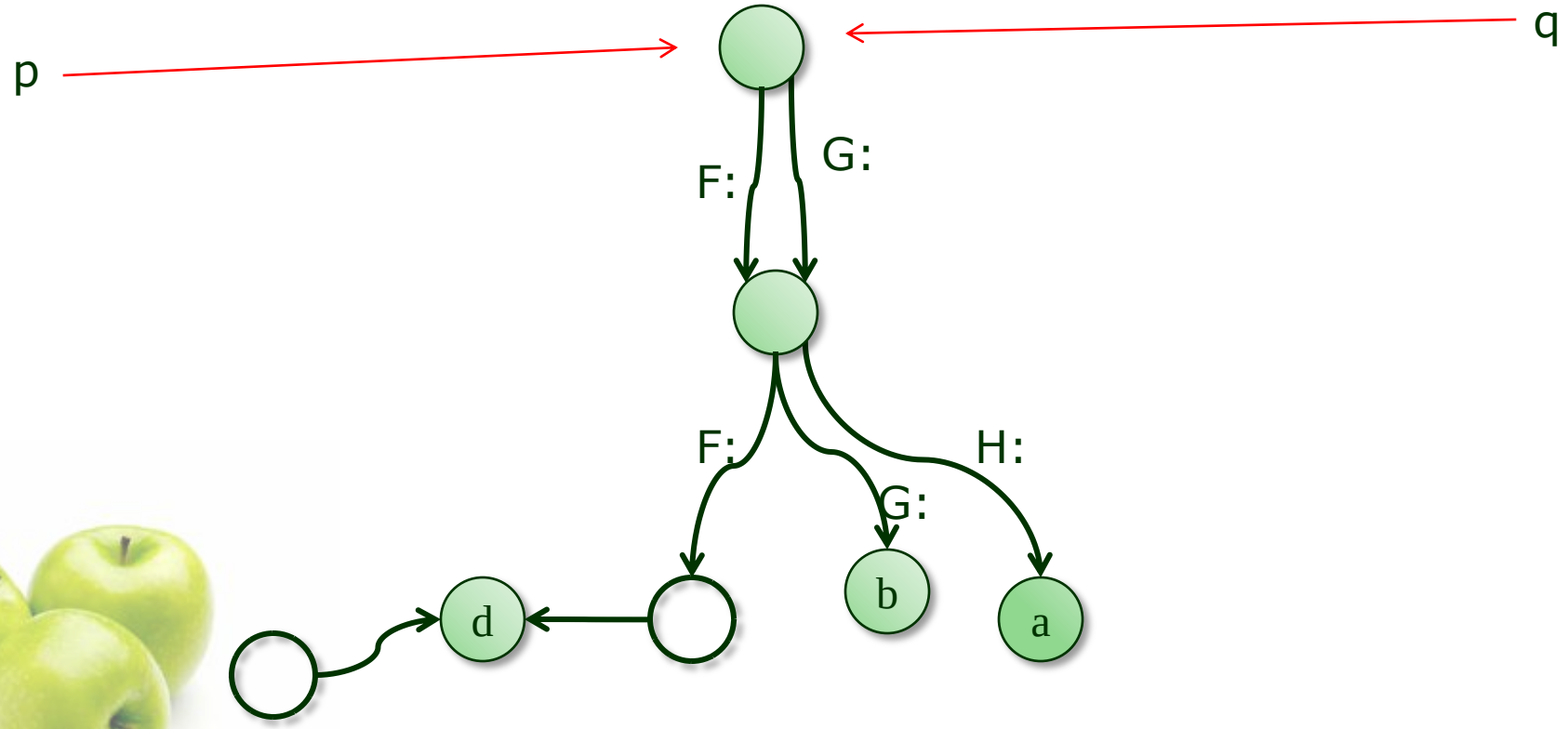
単一化の例 (2)

- Unify(p, q)

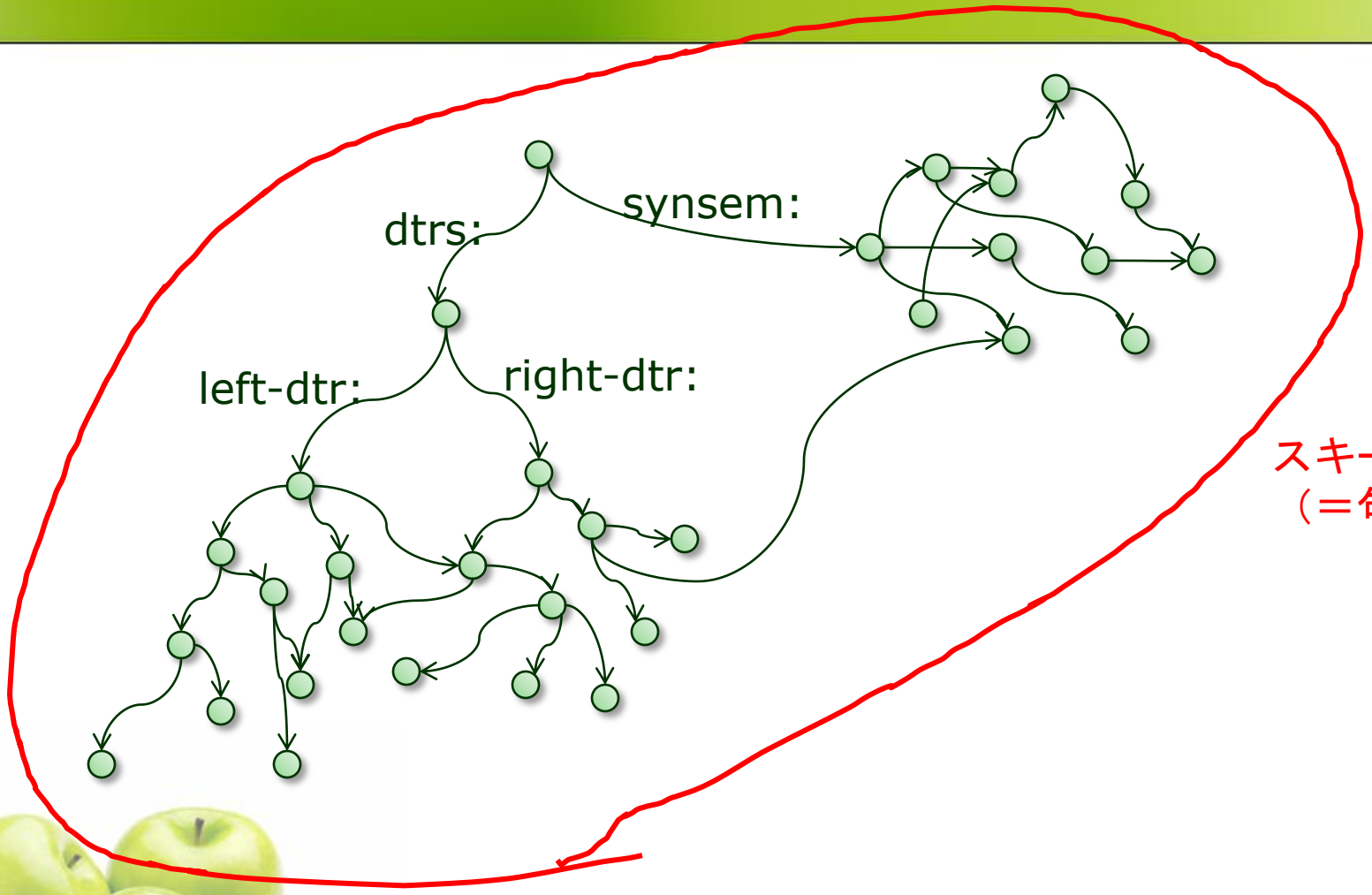


単一化の例 (2)

- Unify(p, q)

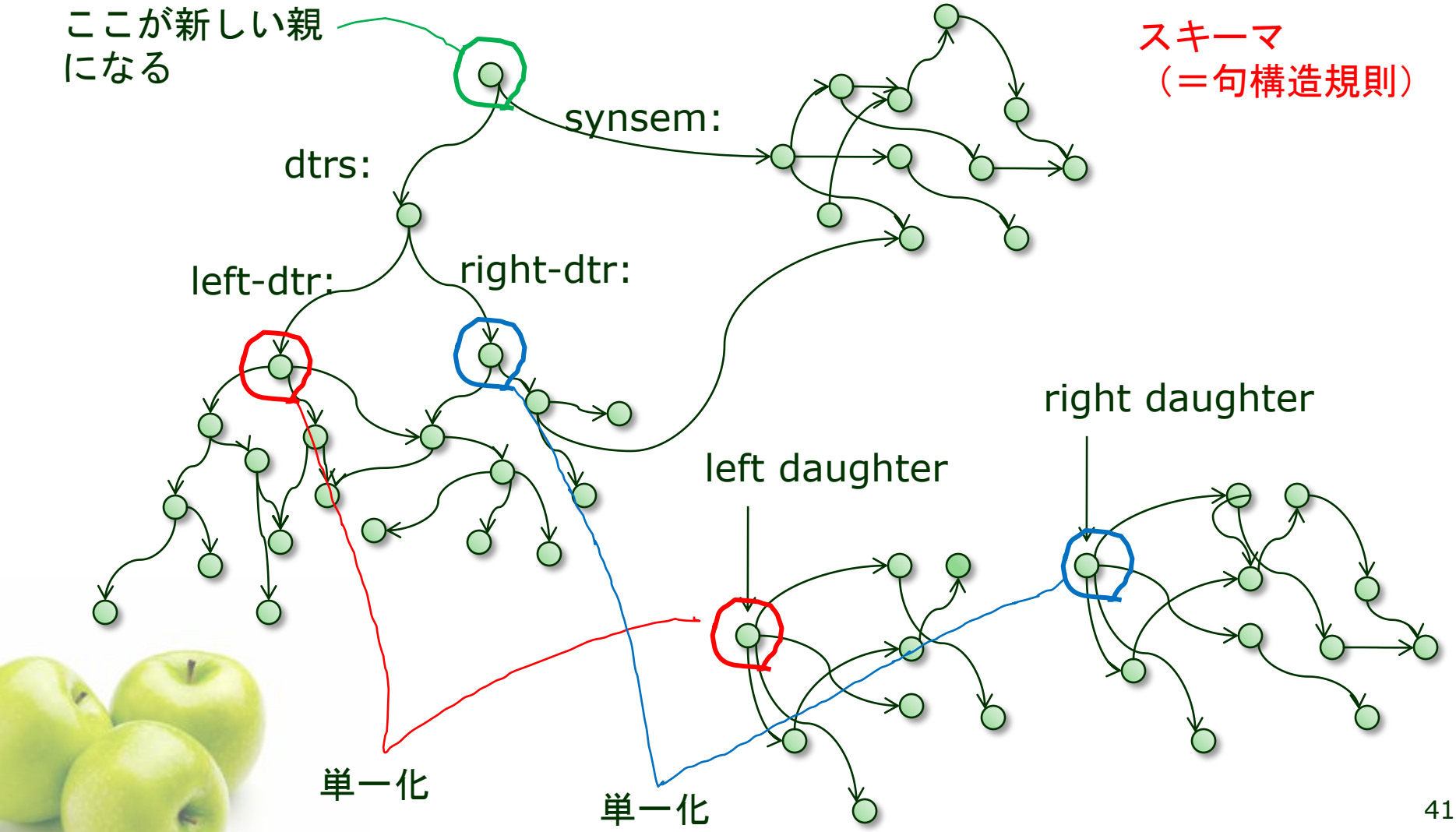


HPSGのスキーマの単一化

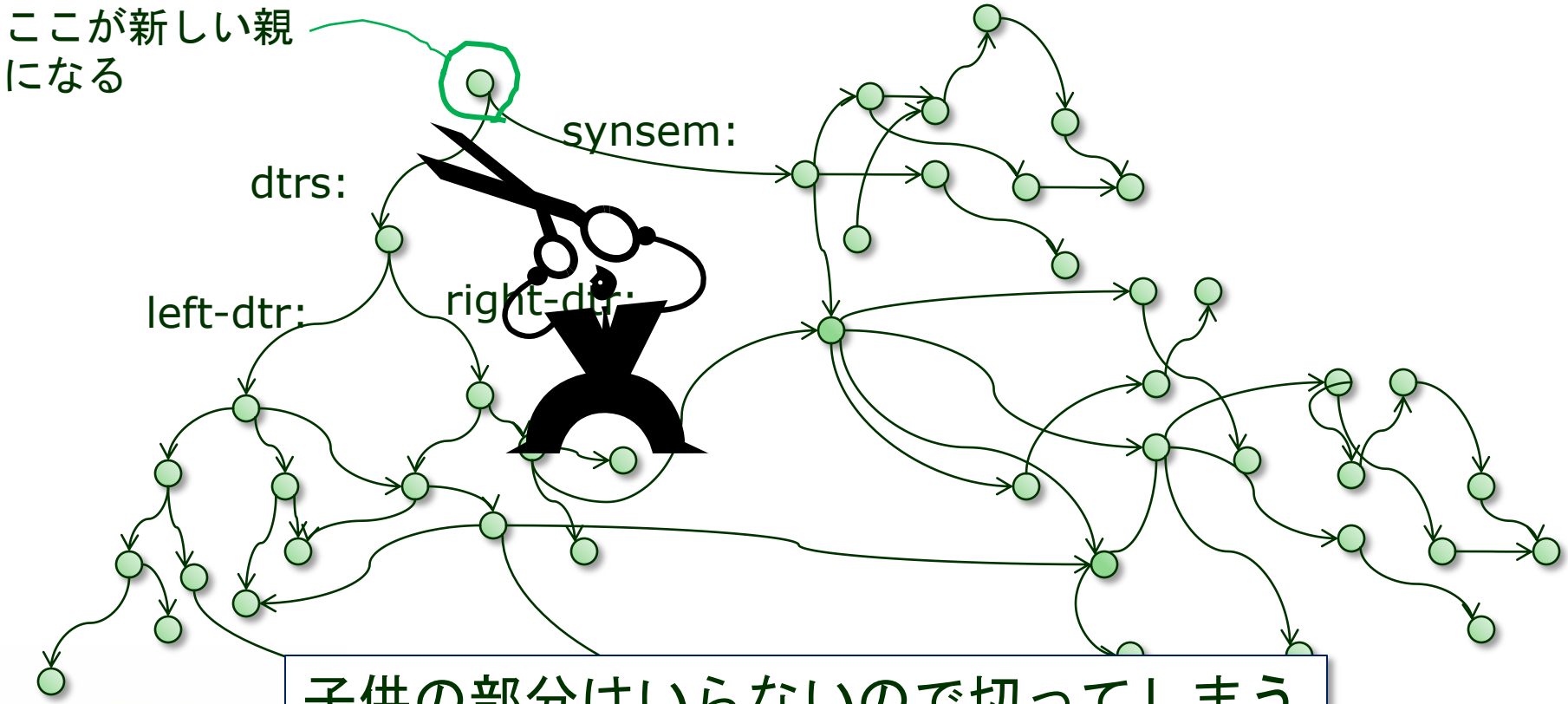


スキーマ
（＝句構造規則）

HPSGのスキーマの単一化



Reduce Sign



子供の部分はいらないので切ってしまう

Reduce Sign

- 子供(daughters)の部分をカット
  - 子供がまるまる残っていると、構文木のルートノードに全ての展開された構文木集合が格納されてしまう（簡単に数百億以上の構文木になってしまう）
  - そこで、子供の部分をカットしてCKYチャートに格納してやればよい

CKYアルゴリズム

- CFGのCKYアルゴリズムとの違い
  - 非終端記号のかわりに素性構造が格納される
  - 子供をカットする操作 (reduce sign)が必要
  - 単一化は破壊的操作なので、実行前にコピーをして元の素性構造を残しておく
  - ルール規則の適用
    - CFG: 非終端記号X, Yに対し、G(X, Y)は親の非終端記号集合を返す
    - HPSG: 素性構造X, Yに対し、G(X, Y)は親の素性構造集合を返す
  - ファクタリング
    - CFG: 等価な非終端記号
    - HPSG: 等価な素性構造 (X = Y iff X ⊑Y and X ⊒ Y)
  - デコーディングの時も同様

CKY法 for HPSG

for j = 1 to n
    $S_{j-1,j}$:= L(w_j)　## Lは単語wに対する素性構造の集合を返す関数
for l = 2 to n
  for i = 0 to n – l
    j := i + l;
    for k = i+1 to j - 1
      forall X∈$S_{i,k}$
        forall Y∈$S_{k,j}$
          forall b ∈ BS　　# BSはバイナリースキーマの集合
            X′ := copy(X); Y′ := copy(Y); b′ := copy(b)
            Si,j := Si,j ∪ Reduce(b′(X′, Y′))
      $S_{i,j}$:= $S_{i,j}$ ∪ U($S_{i,j}$)　## Uはユーナリールールの適用手続き

等価性チェックとコピー

- 構造共有があるため素性構造の等価性チェックやコピーはそんなに簡単ではない
  - 破壊的でない操作
    - すでに辿ったセルのアドレスをハッシュにいれておいて構造共有をチェック
  - 破壊的操作＋バックトラック
    - すでに辿ったセル上にマーキングタグと対応する相手側のアドレスを書き込む
    - 変更履歴をたどることによって元のデータに戻す（バックトラック）

不完全ながら高速な等価性チェック

- セルの列とセルの列を単純なforループで簡単にチェック
  - コピーする際に、ポインタの作り方や部分構造の作り方を工夫すれば、等価な素性構造はまったく同じセルの列になる
  - ただし、VARタグを展開していない場合とVARタグが展開されてSTRになっている場合では正しいチェックができないので注意

まとめ

- 単一化アルゴリズム
- HPSGフルパージング