特征结构与合一运算

常宝宝
北京大学计算语言学研究所
chbb@pku.edu.cn

上下文无关文法

- 上下文无关文法为句法知识的形式化提供了一个有效的工具。
- 同时，对于上下文无关文法，存在像Earley算法、广义LR算法等一系列有效的算法，进行句法分析。
- 然而，利用上下文无关文法描写自然语言，不但可以生成自然语言中的合法句子，也可以产生大量自然语言中不合法的句子，存在所谓的过度生成问题(over-generation)。
  - A dog bark in the yard (×)
  - Those dog bark in the yard (×)
  - He just vanished an apple (×)

一致性问题

- 英语中限定词和名词之间在数方面要保持一致。例如：
  those dogs (√) the dog (√)
  the dogs (×)
- 英语中主语和谓语在人称和数方面要保持一致。
  (1) 复数主语后面要跟复数的谓语，例如
  those dogs bark in the yard. (√)
  a dog bark in the yard. (×)
  (2) 单数第三人称的主语 后面的谓语动词要变化，如：
  a dog barks in the yard. (√)
  a dog bark in the yard. (×)

动词的次范畴化框架

- 动词可以根据其所要求的搭配成分形成不同的框架，这种框架被称为动词的次范畴化框架(subcategorization frame)。例如:

disappear *==> Verb*

serve *lunch ==> Verb NP*

sent *James a letter ==> Verb NP NP*

sent *a letter to James ==> Verb NP PP*

think *that you will come ==> Verb* S_{fin}

tell *us where we are ==> Verb NP* S_{wh}

上下文无关文法

- 可以采用对句法范畴进行分类的方式解决过度生成问题。例如：
  - 按照单复数把*Det*、*NP*、*VP*进一步分成*Det_sg* 、*Det_pl*、*NP_sg*、*NP_pl*、*VP_sg*、*VP_pl*，则

    S → *NP_sg VP_sg*
    S → *NP_pl VP_pl*
    NP_sg → *Det_sg Noun_sg*
    NP_pl → *Det_pl Noun_pl*
    ……
  - 按照人称，把*NP_sg*、*VP_sg*还可以进一步分成*NP_sg_3rd*、*NP_sg_non3rd*、*VP_sg_3rd*以及*VP_sg_non3rd*，则

    S → *NP_sg_3rd VP_sg_3rd*
    S → *NP_sg_non3rd VP_sg_non3rd*
    ……
- 句法范畴的数量迅速增加，导致重写规则的数量爆炸性增加。

上下文无关文法

- 在上下文无关文法中，只使用了单一的语法范畴标记，无法表示更加细致的语言学特征。
- 由于没有细致的语言学特征，成分之间是否可以组合缺乏判别依据。
- 解决的办法是引入更多的语言特征，并允许在成分和成分组合时进行某种测试。

  $S \rightarrow NP\ VP$

  Only if the number of the NP is equal to the number of VP.
- 需要 特征描述手段 以及 测试手段

语言中的特征继承

- 语言中成分和成分组合形成一个更大成分时，如何确定这个组合成分的特征。
- 语言中的很多结构属于一种向心结构，组成组合成分的不同成分地位并不相同，组合成分的特征往往从其中心组成成分(head)那里继承特征。例如

NP → *Det* ***Nom*** (Nom 是NP的中心成分，NP的特征继承自Nom)
VP → ***Verb*** *NP* (Verb 是VP的中心成分，VP的特征继承自Verb)
VP → ***VP***$_1$ *PP* (VP_1 是VP的中心成分，VP的特征继承自VP_1)

serves **SG 3rd**
serves lunch **SG 3rd**
serves lunch in the restaurant **SG 3rd**

特征结构(Feature Structure)

- 将上下文无关文法中的简单句法范畴扩展为带若干特征的复杂句法范畴(特征结构)。
- 应用重写规则时，要首先通过特征结构的检验。
- 特征结构是有限个“特征-值”对的集合。特征结构形式上可写成如下形式。

$$\begin{bmatrix} \text{Feature}_1 & \text{Value}_1 \\ \text{Feature}_2 & \text{Value}_2 \\ \vdots & \vdots \\ \text{Feature}_n & \text{Value}_n \end{bmatrix}$$

- 特征结构也称复杂特征集(complex features set)或属性值矩阵(attribute-value matrix)。

特征结构

- 为了描述成分的范畴属性、人称属性以及单复数属性，可以设定特征CAT、NUMBER以及PERSON
  - CAT的值可以是NP、VP、Verb、Noun等
  - NUMBER的值可以是SG和PL。
  - PERSON的值可以是1、2和3。
- 例如一个单数第三人称的名词短语(*NP_sg_3rd*)可用下面的特征结构描述：

CAT	NP
NUMBER	SG
PERSON	3

特征结构

- 在特征结构中，特征的值不仅可以是简单的原子值(不再可分解)，也可以是一个特征结构。
- 例如：可以把NUMBER和PERSON两个特征放在一起，用一个新的特征AGREEMENT来描述。

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SG} \\ \text{PERSON} & 3 \end{bmatrix} \end{bmatrix}$$

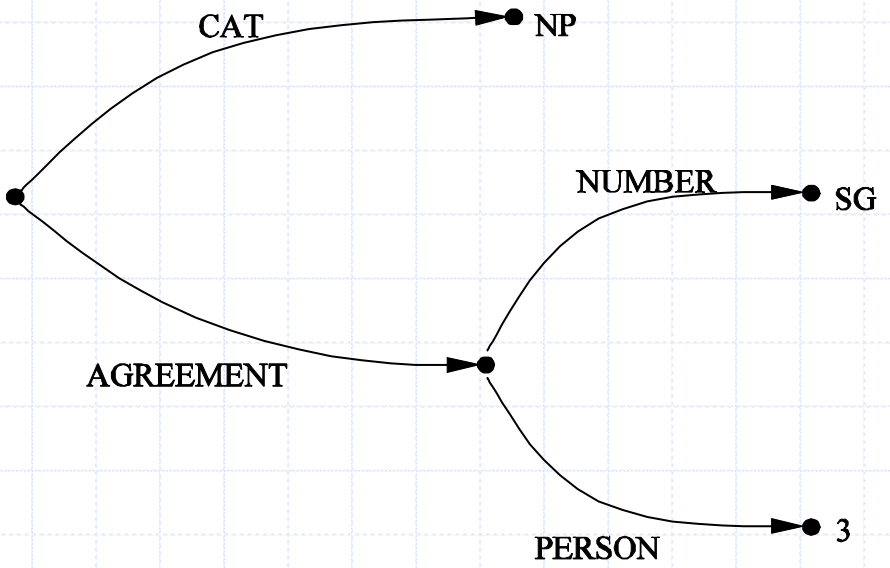
特征结构的图形表示

- 特征路径(feature path)：特征结构中，按照特征-值关系，形成的形如<$Feature_1$ $Feature_2$ … $Feature_m$>的特征序列。例如：

  <AGREEMENT NUMBER>

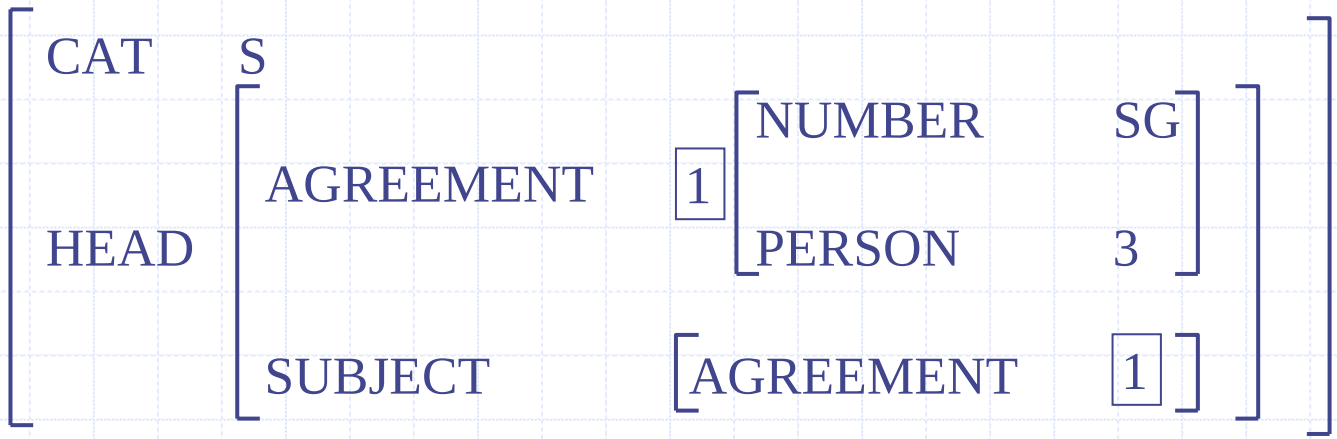
  <AGREEMENT PERSON>

- 特征结构也可以表示为一个有向无环图(DAG)：
  - 特征表示为边
  - 值表示为结点
- 特征结构的图形表示提供了实现特征结构的数据结构。

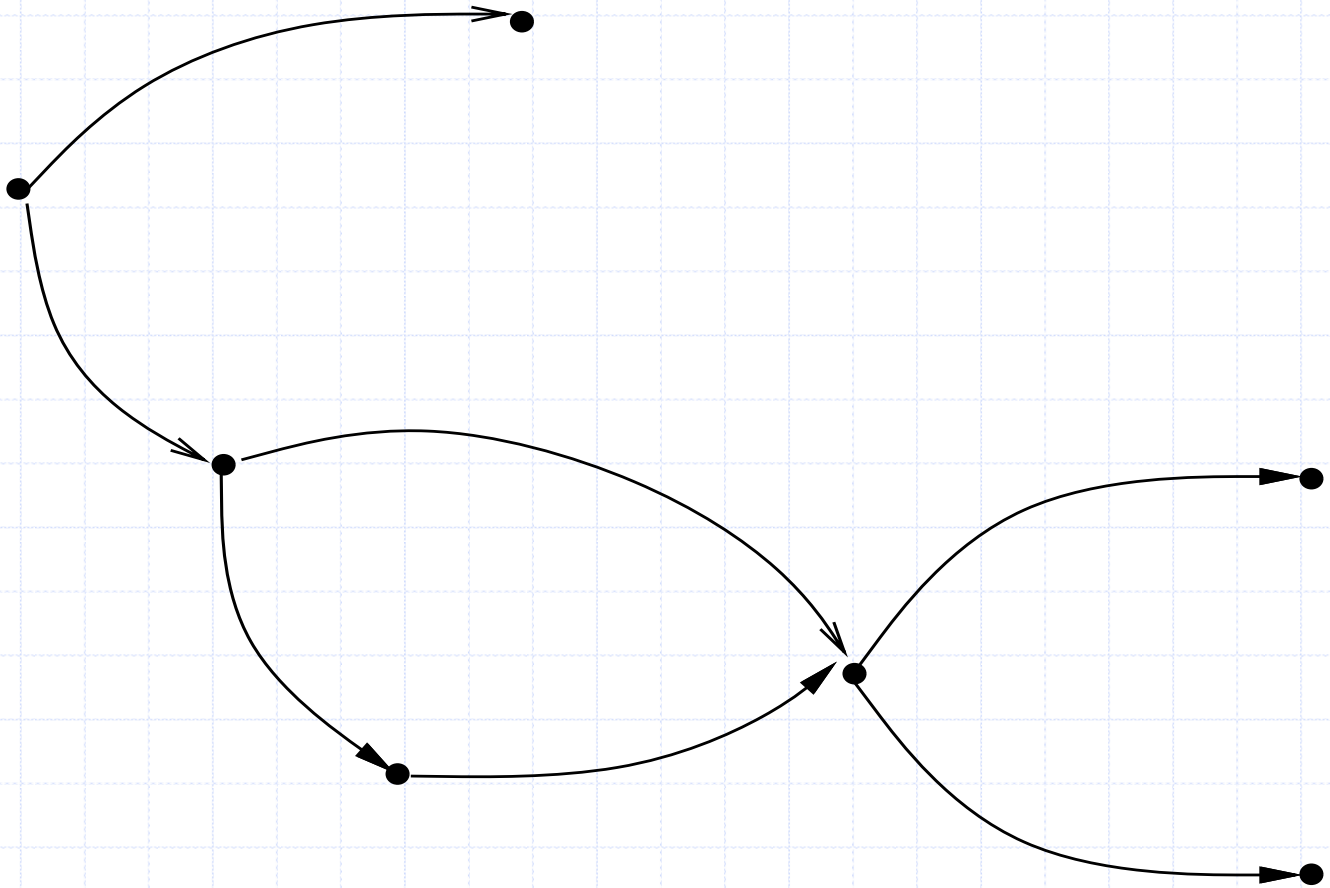


重入式特征结构(Reentrant Structure)

- 在一个特征结构中，不同的特征可以共享同样的特征值。这种特征结构被称为重入式特征结构。(注意：不同的特征取相同的特征值不属于重入式特征结构)
- 在重入式特征结构中，共享特征值用数字索引标示。
- 在下列特征结构中，特征<HEAD AGREEMENT>以及<HEAD SUBJECT AGREEMENT>共享同样的特征值。



重入式特征结构的图形表示



特征结构间的关系

- 蕴涵关系(subsumption)
  - 特征结构F_1蕴涵特征结构F_2，记作$F_1 \dot{\subseteq} F_2$，当且仅当
    (1)若特征$f \in F_1$，则$f \in F_2$，并且
    (2)若f的值是特征结构，则$value_{F1}(f) \dot{\subseteq} value_{F2}(f)$
    (3)若f的值是简单的原子值，则$value_{F1}(f) = value_{F2}(f)$
- 空特征结构
  - 不含任何特征的特征结构，记作[]。
  - 空特征结构蕴涵任何特征结构。

$$\begin{bmatrix} \text{NUMBER} & \text{SG} \end{bmatrix} \dot{\subseteq} \begin{bmatrix} \text{NUMBER} & \text{SG} \\ \text{PERSON} & 3 \end{bmatrix} \dot{\subseteq} \begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUMBER} & \text{SG} \\ \text{PERSON} & 3 \end{bmatrix}$$

$$\begin{bmatrix} \text{NUMBER} & \text{SG} \end{bmatrix} \not\dot{\subseteq} \begin{bmatrix} \text{PERSON} & 3 \end{bmatrix}$$

合一运算(Unification)

- 特征结构的相容性(compatibility)
  - 特征结构F_1和F_2是相容的，当且仅当若$f \in F_1$且$f \in F_2$，则
    (1)若f的值是特征结构，则$value_{F1}(f)$和$value_{F2}(f)$是相容的。
    (2)若f的值是简单的原子值，则$value_{F1}(f) = value_{F2}(f)$
- 特征结构间的合一运算
  - 特征结构F_1和F_2的合一运算记作$F_1 \dot{\cup} F_2$
  - 若F_1和F_2相容，则合一成功，合一结果为F_1和F_2中所有特征组成的新的特征结构。
  - 若F_1和F_2不相容，则合一失败，合一结果是ϕ。
- 合一运算
  - 检查参与合一的特征结构是否相容
  - 归并了参与合一运算的特征结构中的特征信息

合一运算举例

[]和任何特征值相容

$$\begin{bmatrix}\text{NUMBER} & \text{SG}\end{bmatrix} \sqcup \begin{bmatrix}\text{NUMBER} & \text{SG}\end{bmatrix} = \begin{bmatrix}\text{NUMBER} & \text{SG}\end{bmatrix}$$

$$\begin{bmatrix}\text{NUMBER} & \text{SG}\end{bmatrix} \sqcup \begin{bmatrix}\text{NUMBER} & \text{PL}\end{bmatrix} = \phi\text{，合一失败}$$

$$\begin{bmatrix}\text{NUMBER} & \text{SG}\end{bmatrix} \sqcup \begin{bmatrix}\text{NUMBER} & [\,]\end{bmatrix} = \begin{bmatrix}\text{NUMBER} & \text{SG}\end{bmatrix}$$

$$\begin{bmatrix}\text{NUMBER} & \text{SG}\end{bmatrix} \sqcup \begin{bmatrix}\text{PERSON} & 3\end{bmatrix} = \begin{bmatrix}\text{NUMBER} & \text{SG} \\ \text{PERSON} & 3\end{bmatrix}$$

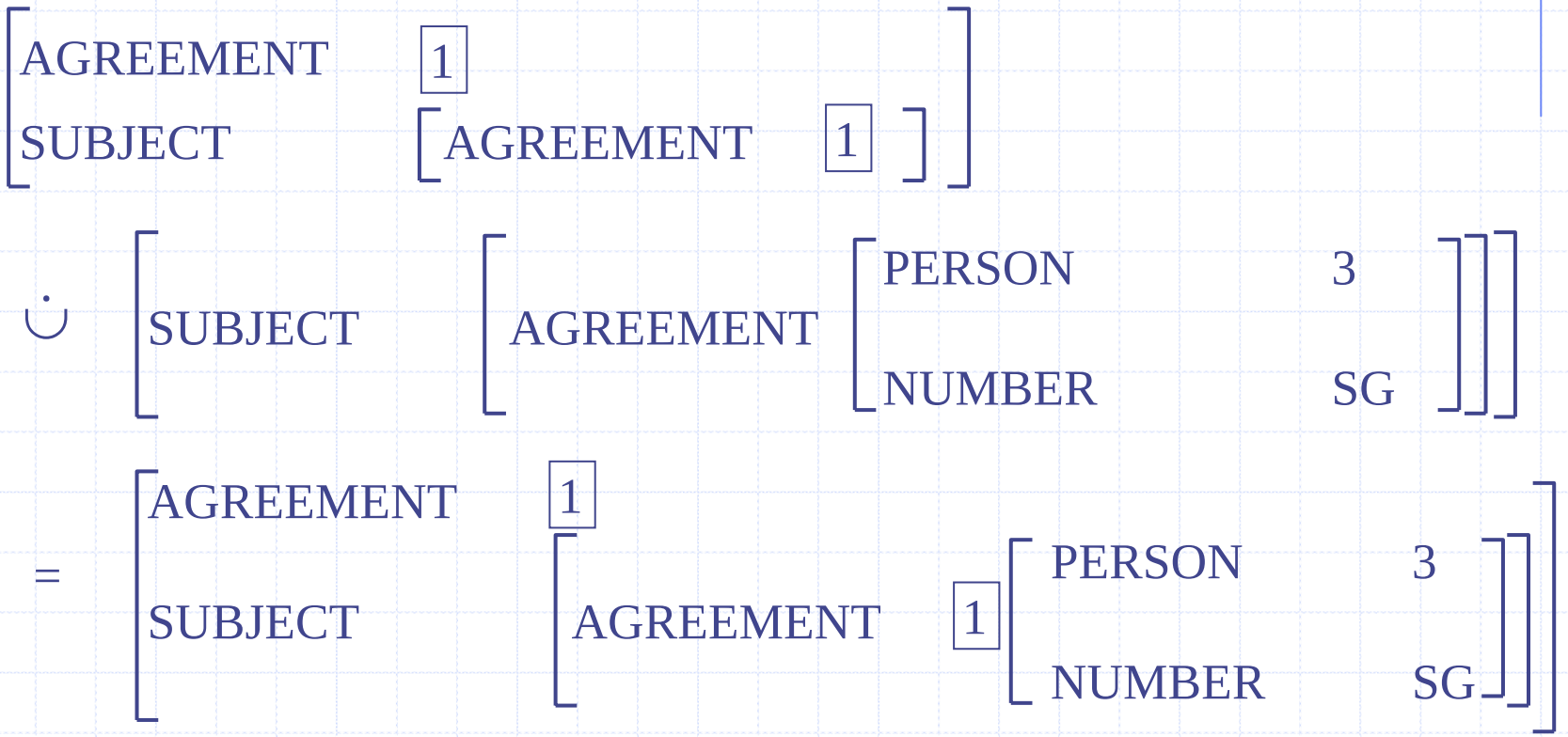
合一运算举例

$$\begin{bmatrix} \text{AGREEMENT} & \boxed{1}\begin{bmatrix} \text{NUMBER} & \text{SG} \\ \text{PERSON} & 3 \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

$$\dot{\cup} \quad \begin{bmatrix} \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{PERSON} & 3 \\ \text{NUMBER} & \text{SG} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \text{AGREEMENT} & \boxed{1}\begin{bmatrix} \text{NUMBER} & \text{SG} \\ \text{PERSON} & 3 \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

合一运算举例


AGREEMENT 1
SUBJECT AGREEMENT 1
∪
SUBJECT AGREEMENT PERSON 3
NUMBER SG
=
AGREEMENT 1
SUBJECT AGREEMENT 1 PERSON 3
NUMBER SG

合一运算举例

$$\begin{bmatrix} \text{AGREEMENT} & \boxed{1}\begin{bmatrix} \text{NUMBER} & \text{SG} \\ \text{PERSON} & 3 \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

$$\cup \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SG} \\ \text{PERSON} & 3 \end{bmatrix} \\ \text{SUBJECT} & \begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{PL} \\ \text{PERSON} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$= \phi$，合一失败

合一运算

- 合一运算的性质
  - 单调性(monotonic)：如果某个特征结构具有某种特征，在同其他特征结构合一后，合一结果仍然具有某种特征。
  - 顺序无关性(order independent)：如果多个特征结构进行合一运算，合一结果与合一运算进行的顺序没有关系。

- 若$F_1 \cup F_2 = H$，则 $F_1 \dot{\subseteq} H$，$F_2 \dot{\subseteq} H$，若存在特征结构X，使得$F_1 \dot{\subseteq} X$，$F_2 \dot{\subseteq} X$，则必有$H \dot{\subseteq} X$，合一结果是包含参与合一的特征结构的所有特征的最小特征结构

扩充的上下文无关文法

- 任何句法成分均可被视作一个带有特征结构描述的语言对象。在词典中，为每一个词汇条目增加特征结构描述。
- 较大成分的特征结构描述可通过其组成成分的特征结构描述组合产生。
- 两个较小的成分是否可以组合成较大的成分，取决于他们之间的约束关系是否满足，这可以通过检查特征结构中的特征是否相容而进行。

扩充的上下文无关文法

- 在扩充的上下文无关文法中，重写规则被扩展成如下形式的规则：

  $\beta_0 \rightarrow \beta_1 \cdots \beta_n$

  { 约束集 }

- 其中，每一个约束形式如下

  (1) < β_i feature path > = atomic value

  (2) < β_i feature path > = < β_k feature path >

  < β_i feature path > 表示成分 β_i 的特征结构中的某个特征路径。

- 若约束集中的约束未能满足，则表示重写规则右面的成分不能结合成重写规则左面的成分

处理一致性

- *S* → *NP VP*

  <*NP* AGREEMENT> = <*VP* AGREEMENT>
- *S* → *Aux NP VP*

  <*Aux* AGREEMENT> = <*NP* AGREEMENT>
- *NP* → *Det Nominal*

  <*Det* AGREEMENT> = <*Nominal* AGREEMENT>

  <*NP* AGREEMENT> = <*Nominal* AGREEMENT>
- *VP* → *Verb NP*

  <*VP* AGREEMENT> = <*Verb* AGREEMENT>
- *Aux* → *do*

  <*Aux* AGREEMENT NUMBER> = PL

  <*Aux* AGREEMENT PERSON> = 3

处理一致性

- *Aux* → *does*

  <*Aux* AGREEMENT NUMBER> = SG

  <*Aux* AGREEMENT PERSON> = 3
- *Det* → *this*

  <*Det* AGREEMENT NUMBER> = SG
- *Det* → *these*

  <*Det* AGREEMENT NUMBER> = PL
- *Noun* → *flight*

  <*Noun* AGREEMENT NUMBER> = SG
- *Noun* → *flights*

  <Noun AGREEMENT NUMBER> = PL
- *Verb* → *Serve*

  <*Verb* AGREEMENT NUMBER> = PL

处理一致性

- *Verb* → *serves*

  <*Verb* AGREEMENT NUMBER> = SG

  <*Verb* AGREEMENT PERSON> = 3

- 上述扩充的上下文无关文法可有效处理下面的句子
  - *This flight serves breakfast*
  - *Does this flight serve breakfast*
  - *Do these flights serve breakfast*
  - *This flight serve breakfast* (×)
  - *Do this flight serve breakfast* (×)
  - *Does these flights serve breakfast* (×)

中心特征的继承

- 特征继承的表达
  - *VP* → *Verb NP*

    <*VP* AGREEMENT> = <*Verb* AGREEMENT>
  - *NP* → *Det Nominal*

    <*Det* AGREEMENT> = <*Nominal* AGREEMENT>

    <*NP* AGREEMENT> = <*Nominal* AGREEMENT>
  - *Nominal* → *Noun*

    <*Nominal* AGREEMENT> = <*Noun* AGREEMENT>

中心特征的继承

- 继承自中心成分的特征成为中心特征，引入特征head描述中心特征
  - *VP* → *Verb NP*

    <*VP* HEAD> = <*Verb* HEAD>
  - *NP* → *Det Nominal*

    <*Det* HEAD AGREEMENT> = <*Nominal* HEAD AGREEMENT>

    <*NP* HEAD> = <*Nominal* HEAD>
  - *Nominal* → *Noun*

    <*Nominal* HEAD> = <*Noun* HEAD>
  - *Noun* → *flights*

    <*Noun* HEAD AGREEMENT NUMBER> = PL

处理次范畴化框架

disappear an apple (×)

serves lunch (√)

- 引入次范畴化特征SUBCAT
  - *Verb* → *disappears*

    <*Verb* HEAD AGREEMENT NUMBER> = SG
    <*Verb* HEAD SUBCAT FIRST> = END
  - *Verb* → *serves*

    <*Verb* HEAD AGREEMENT NUMBER> = SG
    <*Verb* HEAD SUBCAT FIRST CAT> = NP
    <*Verb* HEAD SUBCAT SECOND> = END
  - *Verb* → *leaves*

    <*Verb* HEAD AGREEMENT NUMBER> = SG
    <*Verb* HEAD SUBCAT FIRST CAT> = NP
    <*Verb* HEAD SUBCAT SECOND CAT> = PP
    <*Verb* HEAD SUBCAT THIRD> = END
  - *VP* → *Verb NP*

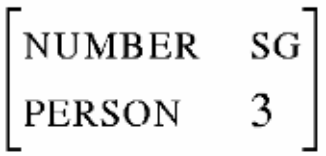
    <*VP* HEAD> = <*Verb* HEAD>
    <*VP* HEAD SUBCAT FIRST CAT> = <NP CAT>
    <*VP* HEAD SUBCAT SECOND> = END

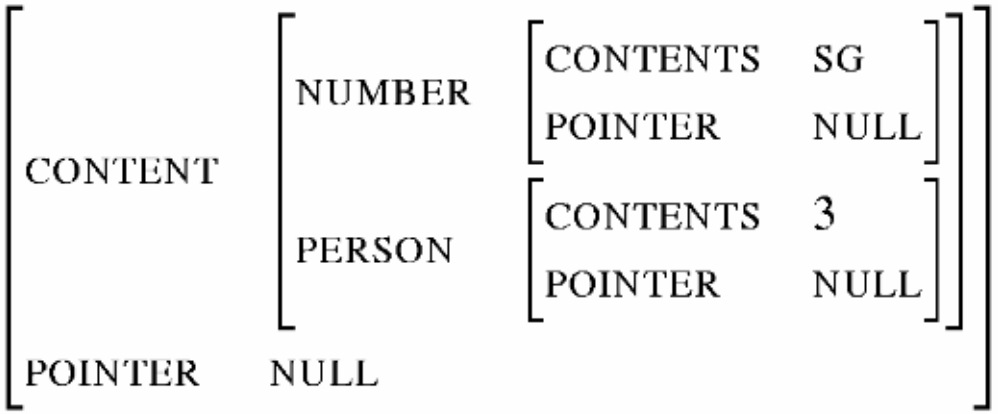
合一运算的计算机实现

- 若将参与合一的特征结构表示为DAG
- 合一实际上是递归的图匹配算法
  若所有特征都匹配，合一成功
  若存在特征不匹配，合一失败
- 目前提出的合一运算的实现多为破坏性的
  合一结束后，无论成功还是失败，参与合一的特征结构遭到破坏。
- 为了易于实现，在实现时，对特征结构进行扩展

合一运算的计算机实现

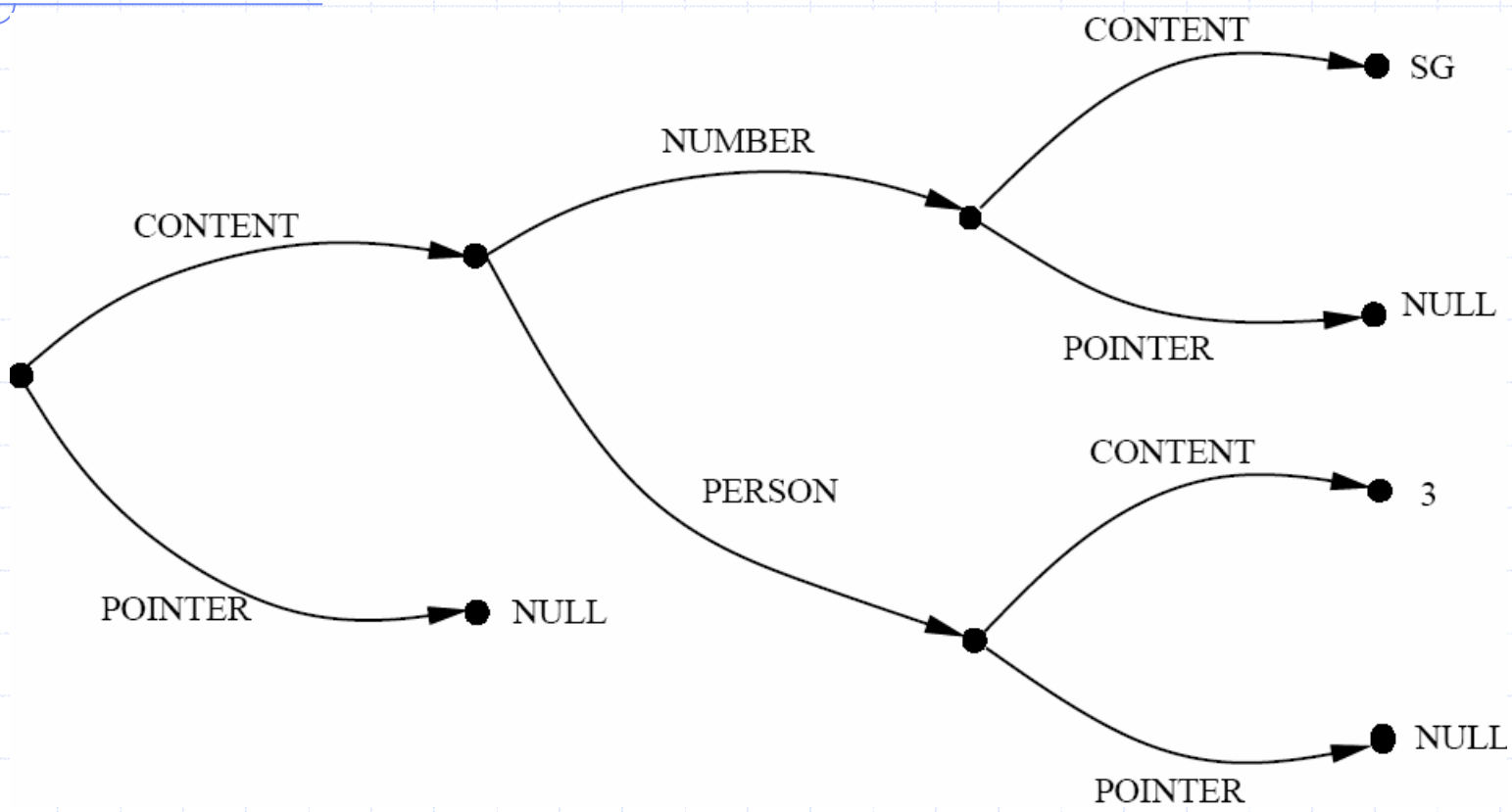
- 为每个特征结构增加POINTER特征和CONTENT特征





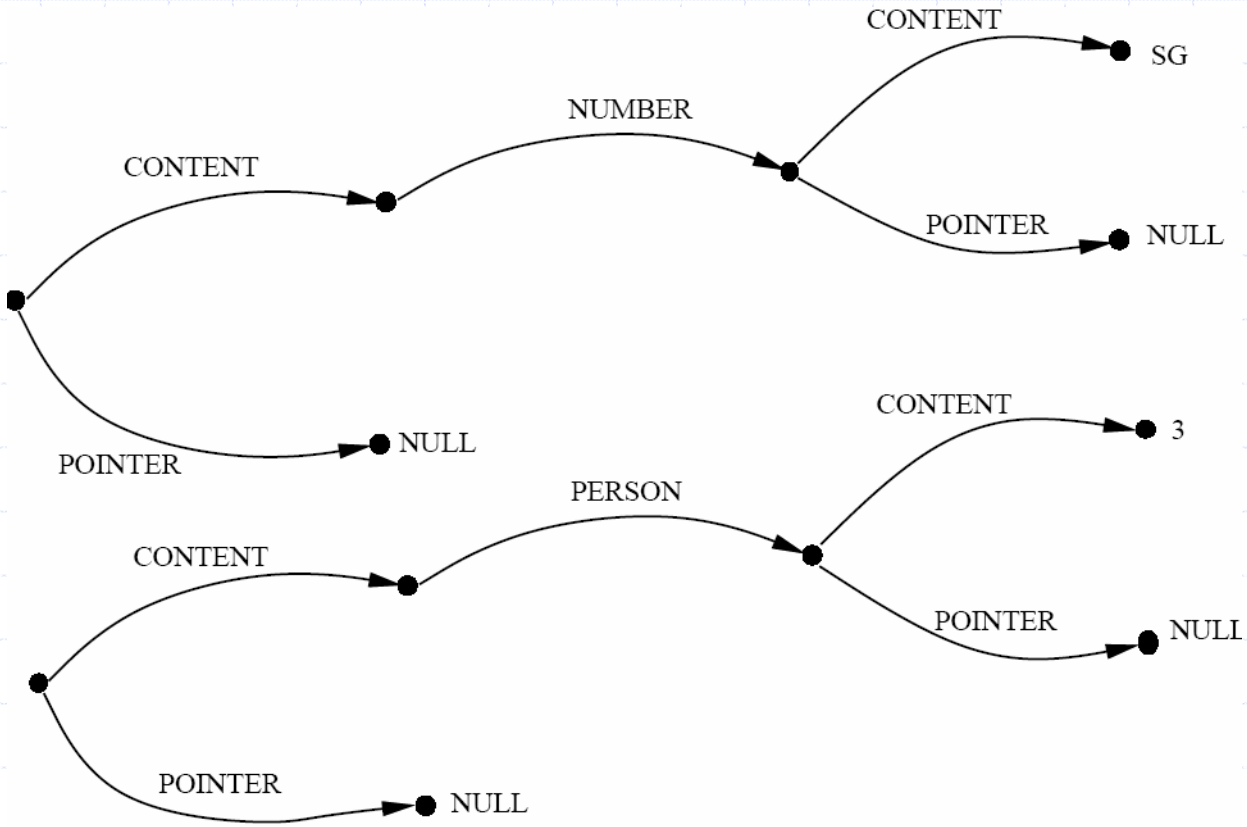
- **if** pointer == null **then** content指向有效的特征结构
  if pointer != null **then** pointer 指向有效的特征结构

合一运算的计算机实现

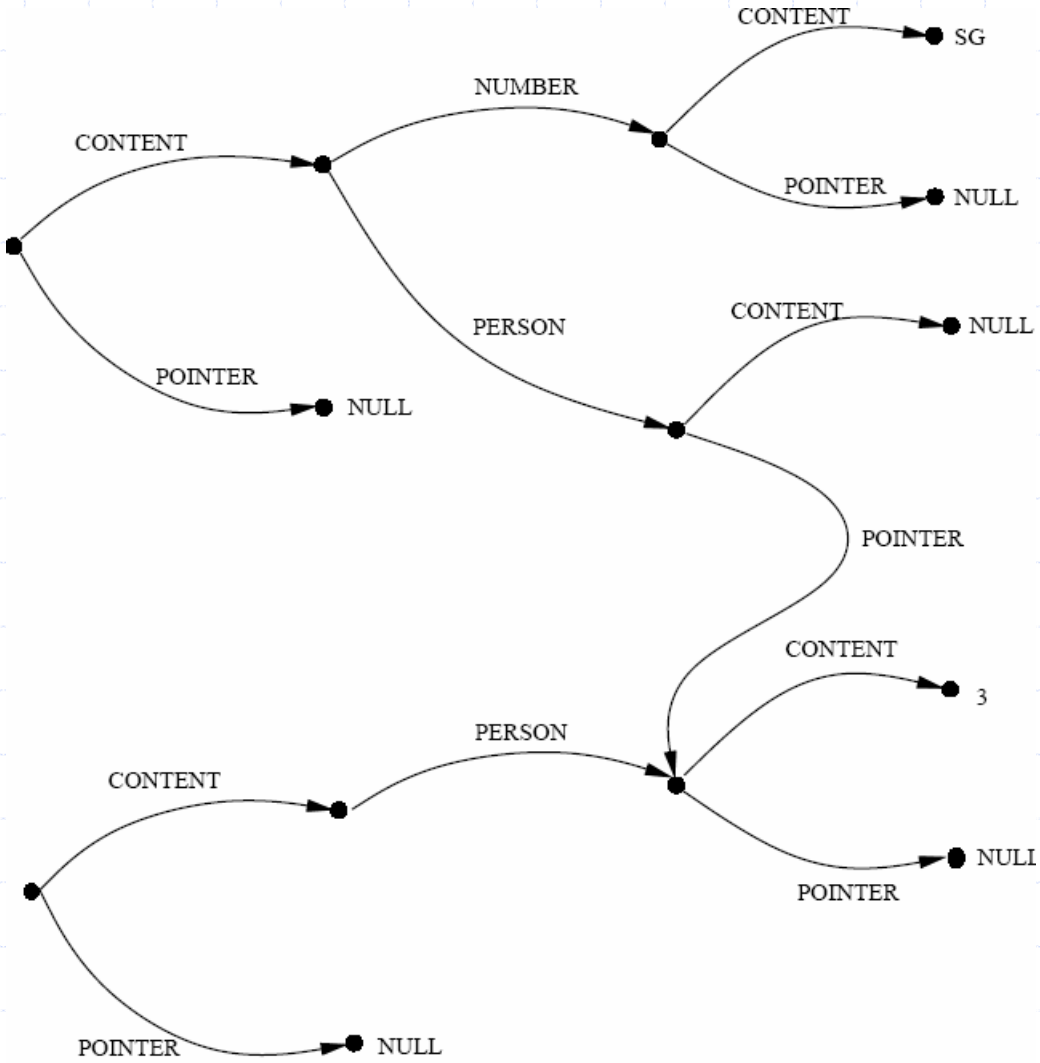

CONTENT
SG
NUMBER
CONTENT
NULL
POINTER
CONTENT
PERSON
3
POINTER
NULL
NULL
POINTER

合一运算的计算机实现

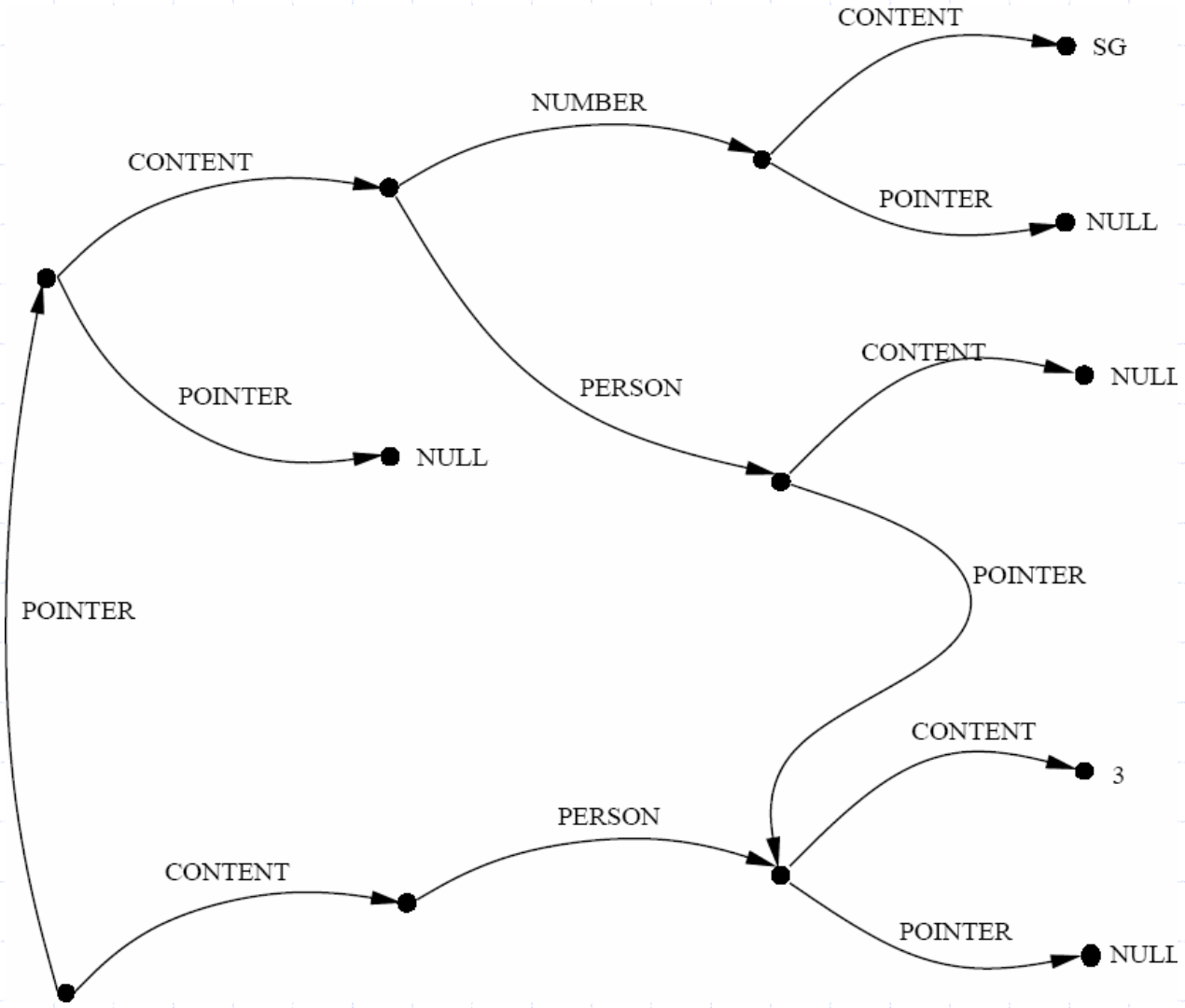
$$\begin{bmatrix}\text{NUMBER} & \text{SG}\end{bmatrix} \sqcup \begin{bmatrix}\text{PERSON} & 3\end{bmatrix} = \begin{bmatrix}\text{NUMBER} & \text{SG} \\ \text{PERSON} & 3\end{bmatrix}$$



合一运算的计算机实现


CONTENT
SG
NUMBER
CONTENT
POINTER
NULL
PERSON
CONTENT
NULL
POINTER
NULL
POINTER
CONTENT
3
PERSON
CONTENT
NULL
POINTER
POINTER
NULL

合一运算的计算机实现


CONTENT
SG
NUMBER
CONTENT
POINTER
NULL
POINTER
NULL
CONTENT
NULL
PERSON
POINTER
POINTER
CONTENT
3
PERSON
CONTENT
POINTER
NULL

合一算法

```
function UNIFY(f1, f2) returns fstructure or failure

 f1-real ← Real contents of f1
 f2-real ← Real contents of f2

 if f1-real is null then
   f1.pointer ← f2
   return f2
 else if f2-real is null then
   f2.pointer ← f1
   return f1
 else if f1-real and f2-real are identical then
   f1.pointer ← f2
   return f2
 else if both f1-real and f2-real are complex feature structures then
   f2.pointer ← f1
   for each feature in f2-real do
     other-feature ← Find or create
                        a feature corresponding to feature in f1-real
     if UNIFY(feature.value, other-feature.value) returns failure then
       return failure
   return f1
 else return failure
```

基于合一的句法分析

- 如何应用扩充的上下文无关文法进行句法分析？
  - 排除约束条件不成立的归约或推导，排除不合理的句法分析结果
  - 为句法成分提供更为丰富的信息表示
- 如何改进基于上下文无关文法的句法分析算法，使之可以使用特征结构及合一运算进行句法分析？
  - Earley算法
  - 广义LR算法
- 对Earley算法进行改进

  （1）根据重写规后所附的约束为规则生成特征结构

  （2）对chart中的状态进行改进

  （3）对COMPLETER()操作进行改进

  （4）状态蕴涵关系检查

基于合一的Earley分析算法

- 将重写规则中所附的约束转写为特征结构

  S → *NP VP*

  <*NP* HEAD AGREEMENT> = <*VP* HEAD AGREEMENT>

  <*S* HEAD > = <*VP* HEAD>

$$\begin{bmatrix} S & [\ \text{HEAD} & \boxed{1}\ ] \\ NP & [\ \text{HEAD} & [\ \text{AGREEMENT} & \boxed{2}\ ]\ ] \\ VP & [\ \text{HEAD} & \boxed{1}\ [\ \text{AGREEMENT} & \boxed{2}\ ]\ ] \end{bmatrix}$$

- 特征结构可以被实现为有向无环图(DAG)

基于合一的Earley分析算法

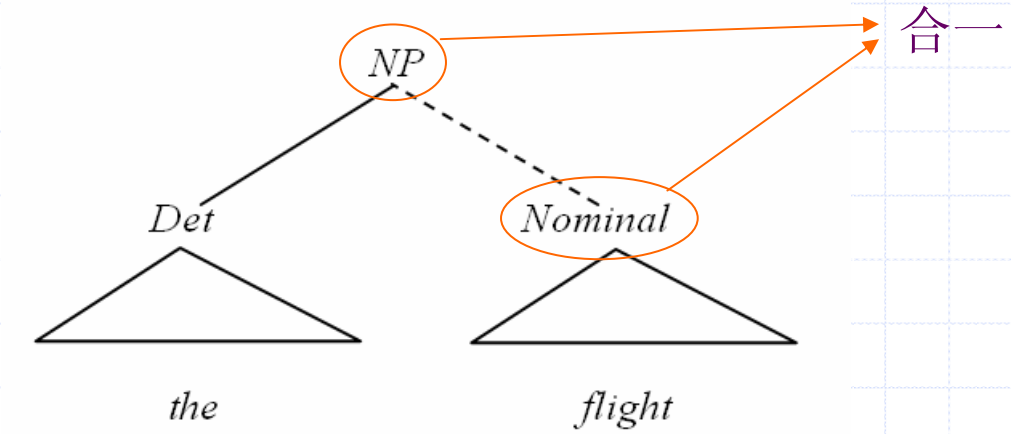
- 为状态增加特征结构字段(用DAG表示)
  - （1）重写规则，代表分析子树
  - （2）子树分析的完成状况，用点标记表示
  - （3）子树完成部分与输入中词的对应关系
  - （4）特征结构

$$S \rightarrow \cdot\, NP\ VP, [0,0], DAG$$

- 每当一个状态被PREDICTOR()加入状态表时，该规则对应的特征结构被作为状态的字段加入。

基于合一的Earley分析算法

- 改进COMPLETER()操作，进行合一运算
  - 每当一个子树(s_1)完成分析时，触发COMPLETER()操作
  - 检查其他状态集中是否有子树(s_2)等待该子树完成分析
  - 将s_1对应的特征结构与s_2对应的特征结构进行合一
  - 若合一失败，不产生新的状态
  - 若合一成功，产生新状态，并把合一结果作为新状态的特征结构



基于合一的Earley分析算法

- *NP* → *Det* ·*Nominal*, [0,1], DAG_1

$$\begin{bmatrix} \text{NP} & \begin{bmatrix} \text{HEAD} & \boxed{1} \end{bmatrix} \\ \text{DET} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{AGREEMENT} & \boxed{2}\begin{bmatrix} \text{NUMBER} & \text{SG} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{NOMINAL} & \begin{bmatrix} \text{HEAD} & \boxed{1}\begin{bmatrix} \text{AGREEMENT} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

- *Nominal* → *Noun* ·, [1,2], DAG_2

$$\begin{bmatrix} \text{NOMINAL} & \begin{bmatrix} \text{HEAD} & \boxed{1} \end{bmatrix} \\ \text{NOUN} & \begin{bmatrix} \text{HEAD} & \boxed{1}\begin{bmatrix} \text{AGREEMENT} & \begin{bmatrix} \text{NUMBER} & \text{SG} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

- 将DAG_1中的*Nominal*同DAG_2中的*Nomial*进行合一

基于合一的Earley分析算法

- 状态蕴涵关系检查
  - 为了保证不重复分析子树，在Earley算法中，如果新产生的状态和状态集合中的某个状态相同，新状态将不被加入状态集
  - 在基于合一的Earley算法中，同时还要检查状态所附的特征结构是否具有蕴涵关系，若新状态的特征结构被状态集中的状态的特征结构蕴涵，则不被加入状态集。

NP → · *Det NP*, [*i*,*i*], *DAG*

若状态集中的状态对*Det*没有约束

而新产生的状态要求*Det*必须是单数的

则新状态不必加入

基于合一的Earley分析算法

```
function EARLEY-PARSE(words, grammar) returns chart

  ENQUEUE((γ → • S, [0,0], dag_γ), chart[0])
  for i ← from 0 to LENGTH(words) do
    for each state in chart[i] do
      if INCOMPLETE?(state) and
                NEXT-CAT(state) is not a part of speech then
          PREDICTOR(state)
      elseif INCOMPLETE?(state) and
                NEXT-CAT(state) is a part of speech then
           SCANNER(state)
      else
          COMPLETER(state)
    end
  end
  return(chart)
```

基于合一的Earley分析算法

procedure PREDICTOR($(A \rightarrow \alpha \bullet B\ \beta,\ [i,j],\ dag_A)$)
  for each $(B \rightarrow \gamma)$ **in** GRAMMAR-RULES-FOR($B, grammar$) **do**
    ENQUEUE($(B \rightarrow \bullet\ \gamma,\ [j,j],\ dag_B), chart[j]$)
  end

procedure SCANNER($(A \rightarrow \alpha \bullet B\ \beta,\ [i,j],\ dag_A)$)
  if B $\subset$ PARTS-OF-SPEECH($word[j]$) **then**
    ENQUEUE($(B \rightarrow word[j],\ [j,j+1],\ dag_B), chart[j+1]$)

procedure COMPLETER($(B \rightarrow \gamma \bullet,\ [j,k],\ dag_B)$)
  for each $(A \rightarrow \alpha \bullet B\ \beta,\ [i,j],\ dag_A)$ **in** $chart[j]$ **do**
    if $new\text{-}dag \leftarrow$ UNIFY-STATES(dag_B, dag_A, B) $\neq$ Fails!
      ENQUEUE($(A \rightarrow \alpha\ B \bullet \beta,\ [i,k], new-dag), chart[k]$)
  end

基于合一的Earley 分析算法

```
procedure UNIFY-STATES(dag1, dag2, cat)
  dag1-cp ← COPYDAG(dag1)
  dag2-cp ← COPYDAG(dag2)
  UNIFY(FOLLOW-PATH(cat, dag1-cp), FOLLOW-PATH(cat, dag2-cp))

procedure ENQUEUE(state, chart-entry)
    if state is not subsumed by a state in chart-entry then
        PUSH(state, chart-entry)
    end
```