10. СИНТАКТИЧЕН АНАЛИЗ С БЕЗКОНТЕКСТНИ ГРАМАТИКИ

Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth...

Robert Frost *The Road Not Taken*

Тази глава въвежда алгоритми за синтактичен анализ. Ние определихме синтактичния анализ в Глава 3 като комбинация от разпознаване на входящ стринг и определянето на няколко структури за него. Синтактичният анализ в такъв случай е разпознаването на изречението и записването на синтактичната му структура. Ние сме най-вече заинтересувани от вида на структурите назначени от безконтекстните граматики на Глава 9. Безконтекстните свободни граматики са декларативен формализъм, те не определят как точно се прави граматичен разбор на дървото за дадено изречение. Тази глава има за цел да ви покаже някои от многото възможни алгоритми за автоматично записване на контекстно-свободно дърво на вложено изречение. Правенето на граматичен разбор на дървета е полезен за приложенията както например, граматична проверка в системите на текстообработката ; изречението, на което не може да бъде направен граматичен разбор би трябвало да има граматични грешки (или поне да бъде трудно за четене). В допълнение, синтактичният анализ може да бъде важен междинен етап на представяне за семантичен анализ (което ще видим в Глава 15), и това ще ни предостави важна роля в приложенията както машинен превод и отговаряне на въпроси. Синтактичните анализатори са също така използвани в лексикографските приложения за създаване на on-line версии на речници. Накрая, наскоро бяха включени стохастичните версии на синтактичен анализ на алгоритми за речевите разпознаватели и за езикови модели (Ney, 1991) и за неопределено състояние на акустично и фонетично моделиране на (Lari и Young, 1991).

Основният синтактичен анализ на алгоритъм представен в тази глава е “Earley” алгоритъм (Earley, 1970) най-лесния за клас на без-контекстен синтактичен анализ на алгоритмите основани на *динамични методи за програмиране* (други включват Cocke-Kasami- Younger (CKY) ()? и Graham - Harrison-Ruzzo (GHR) (Graham и други, 1980) алгоритми). Ние започваме, въпреки мотивиращите различни основни синтактични анализационни идеи, които съставляват алгоритъма. Първо отново преглеждаме ‘търсещата метафора’ за синтактичен анализ и опознаване, която ние въведохме за крайно състояние на автомат в Глава 2, и разисквахме низходящата и възходящата стратегии за търсене. Ние тогава развиваме сравнително ефективна низходяща, връщаща се в изходно състояние реализация, която се опитва да се възползва от силите на тези стратегии.

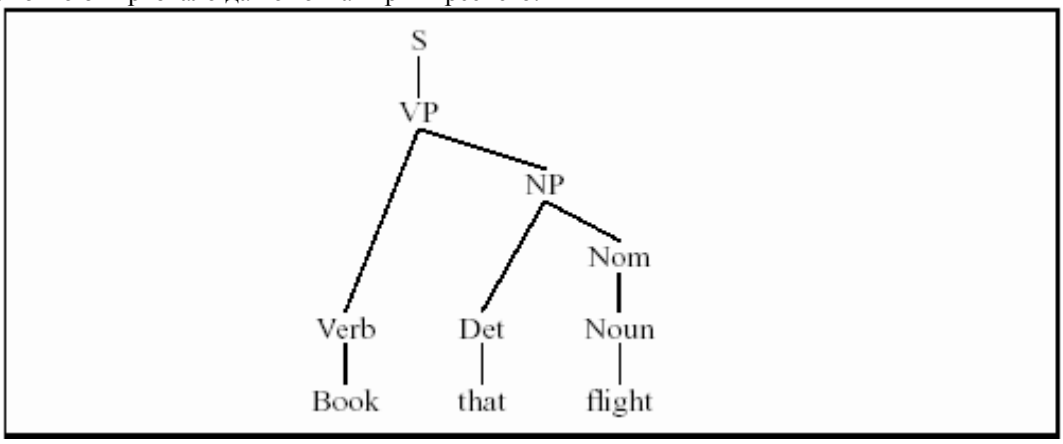
10.1 СИНТАКТИЧЕН АНАЛИЗ КАТО ТЪРСЕНЕ

Главите 2 и 3 показаха това откриване на правилния път през определен-автомат или откривайки правилното преобразуване за вход, можещ да бъде разгледан като търсения проблем. За Федерални агенции за сигурност, например, синтактичният анализатор претърсва пространството на всички възможни пътища през автомата. В синтактичния анализ, синтактичния анализатор може да се разгледа като претърсване на пространството на всички възможни граматични дървета за да се намери правилното граматично дърво за изречението. Точно така, както търсенето пространство на възможни пътища определи от структурата на ФЕДЕРАЛНАТА АГЕНЦИЯ ЗА СИГУРНОСТ, така търсенето пространство на възможните граматични дърветата е определено от граматиката. Например, разгледайте следващия ATIS изречение:

1) Летящата книга.

Използване на миниатюрната граматика и речникът на фигура 10.2, който съдържа на някой от CFG правилата за Английски въведени в Глава 9, правилното граматично дърво е записано на примера показан на фигура 10.1.

Как можем да използваме граматиката на фигура 10.2 да назначим граматичен разбор на дървото на фигура 10.1 на пример (1)? (В този случай има само един правилен разбор на граматичното дърво, но това е възможно за там, където е повече от едно). Целта граматичното търсене е да се намерят всички дървета чиито корен е началният символ S, коя покривка точно въведените думи. Въпреки това за търсенето на алгоритъма ние избираме, двата вида действия по принуда които би трябвало да помогнат при търсенето.



Фигура 10.1 правилния граматичен разбор на дърво за изречението "Летящата книга" (Book that flight), съгласуван с граматиката на фигура 10.2

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	
$Nominal \rightarrow Noun\ Nominal$	$Prep \rightarrow from \mid to \mid on$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	

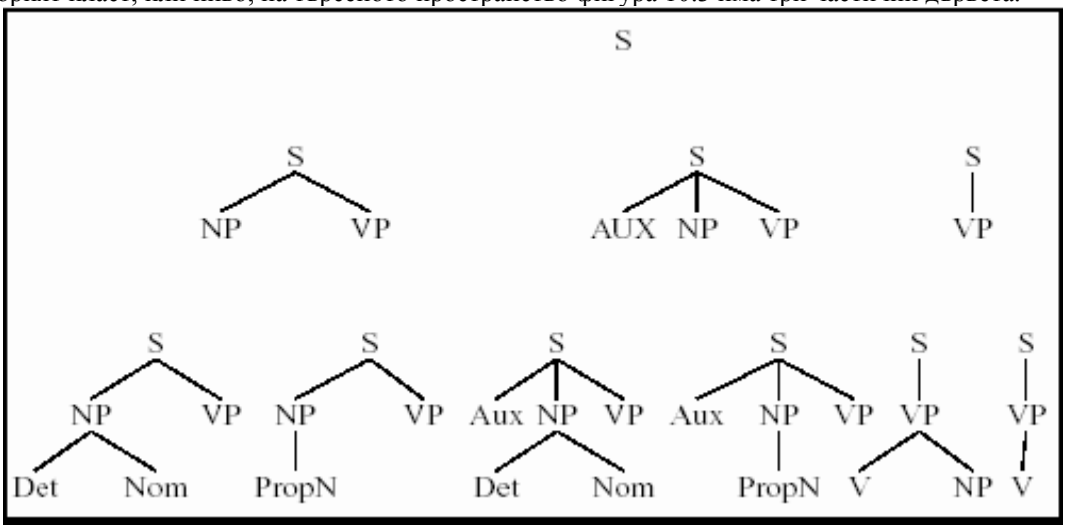
Фигура 10.2 миниатюрна Английска граматика и речник.

Един вид принудата произхожда от данните, например входът осъжда себе си. Каквото още е вярно на края на разбора на граматичното дърво, ние знаем, че трябва да има три изхода и те трябва да бъдат книгата за думите: "книжен" и "полет". Второто изискване произхожда от граматиката. Ние знаем, че всичко, което е вярно на накрая на разбора на граматичното дърво, би трябвало да има корен, който трябва да бъде стартовият символ S.

Тези две изисквания положени в основата подобряват двете основни търсещи стратегии: низходяща или "целево-ориентирана" и възходяща или "ориентирана към данните". Тези изисквания са повече стратегии за търсене. Те отразяват две важни същности в западната философска традиция: емпиричната традиция (фокусирайте върху данните), и рационалната традиция (фокусирайте на по-ранното знание). Ние ще видим тези същности, например, в моделите за изучаване в Глава 13.

Низходящ синтактичен анализ

Низходящия синтактичен анализатор започва чрез приемането на входа, който може да бъде извлечен от определящия стартов символ S. Това, следователно, произвежда върховете на всичките възможни дървета, които започват с S от търсене на всичките граматични правила с S на ляво ориентираната страна. В граматиката на фигура 10.2, има три правила, които разширяват S, ето защо вторият пласт, или ниво, на търсеното пространство фигура 10.3 има три частични дървета.



Фигурата 10.3 разширение на низходящо търсещо пространство.

За в бъдеще ние ще разширим изискванията за тези нови дървета, точно така, както за нас първоначално бе разширено S. Първото дърво ни казва да очакваме NP следван от VP, вторият очаква Aux следван от NP и VP, и третият очаква VP самостоятелно. За да съответствувате търсенето пространство на страницата, ние имаме показани в третият слой на Фигура 10.3 само дърветата, които дават резултат на разширението най-вече на изходите от ляво на всяко дърво. Във всеки слой на пространството на търсенето ние използваме дясната страна на правилата за да се осигури новата серия на очаквания за синтактичния анализатор, които тогава са използвани за рекурсивно генериране на останалите на дърветата. Дърветата водят надолу, докато евентуално достигнат част от речта категоризираща се като дъното на дървото. В тази точка дърветата, чиито изходи не успяват да съчетаят всичките думи на входа може да се отхвърлят, напускайки тези, които се оказват сполучливи за граматични разбори.

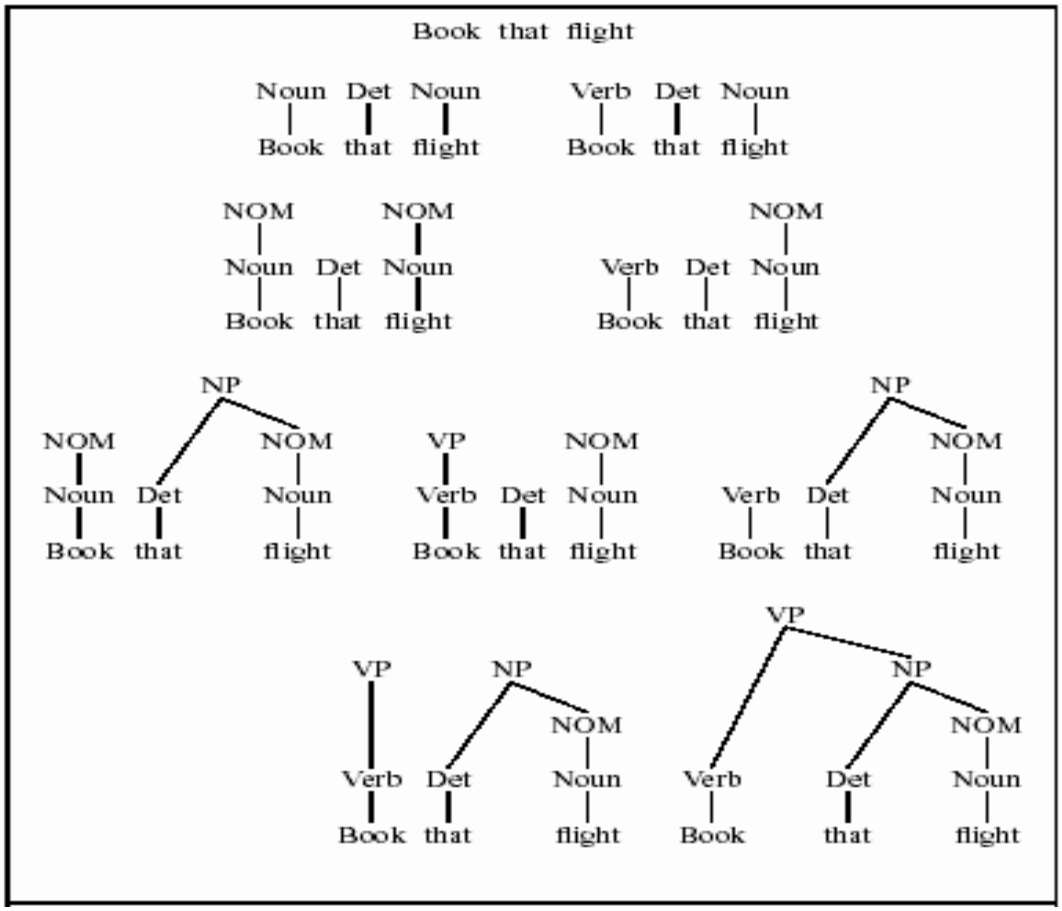
На фигура 10.3, само 5-тото граматично дърво (това което е разширило правилото VP -> глагол NP) евентуално ще съчетае входящото изречение “Книжният полет”.

Възходящ синтактичен анализ

Възходящият синтактичен анализ е най-рано известения алгоритъм за синтактичен (първи го е предложил Yngve (1955) и е използван в намаляващите промяната синтактични анализатори общи за компютърните езици Aho и Ullman (1972)). Във възходящия синтактичен анализ, синтактичен анализатор започва с думите на входа, и опитват да построи нагоре дървета от думи, отново чрез прилагане на правила от граматиката. Граматичният разбор е сполучлив ако синтактичния анализатор успее да изгради дърво вкоренено в началния символния S, който покрива всички входове. Синтактичният анализатор преглежда всяка дума (книга, която, и лети) в речника и изгражда три частични дървета с частта от речта за всяка дума. Но думата “книга” е двусмислена; това може да бъде съществително или глагол. Така синтактичният анализатор трябва да разгледа две възможни групи дървета. Първия два слоя на фигура 10.4 показват това начално раздвоение на търсенето пространство. Всяко от дърветата във вторият слой е разширен. В първия граматичен разбор наляво (това е “книга” неправилно разгледано съществително), правилото на НОМИНАЛНОТО Съществително е отнесено и към двете съществителни (книжен и полет). Това същото правило е също така отнесено към единственото съществително (полет) отдясно, производство на дърветата за третият слой. Всеки слой се разширява на следващ в един от два начина. Първо, синтактичният анализатор може да разпознае места в прогреса на правят граматичен разбор, където един от най-горните нетерминални е дясната страна на някое правило в граматиката. Освен това, последователността на съседен най-горен нетерминален би могла да бъде правилната страна на някое правило. Така в четвъртият слой, в първият и третият граматичен разбор, последователността Det NOM е разпозната като дясно-страничната страна на NP Det NOM правилото. Забележете това, че нашия по-ранен низходящ синтактичен анализатор за разширени дървета, чрез прилагането на правила, когато тяхната лявата страна съчетана не разширения не терминал Възходящият синтактичен анализатор вместо това прилага правила, когато тяхната дясна страна попълни не терминала в различните дървета в процеса на работа. В петият слой, интерпретацията на “книга” като съществително е била окастрена от търсещото пространство. Това е така, защото този граматичен разбор не може да продължи: там няма правило в граматиката с дясно ориентирано NOM NP. Финалният слой на търсенето пространство (не показан на фигура 10.4) е правилният разбор на граматичното дърво (погледнете Фигура 10.1). Убедете се, че разбирате, кой от двата граматични разбора на предпоследните слоеве повишиха този граматичен разбор.

Сравнение между низходящ и възходящ синтактичен анализ

Всяки от тези две архитектури имат своите собствени преимущества и недостатъци. Низходящите синтактични анализатори никога не прахосват време докато изучават дървета това, не довеждат до



Фигурата 10.4 разширение възходящото търсещо пространство за изречението "Летящата книга". Тази фигура не показва крайната редица на търсенето с правилния разбор на граматично дърво (погледнете Фигура 10.1). Убедете се, че разбирате края на граматичният разбор на синтактичното дърво в тази фигура.

S, оттогава те започват с генериране точно на тези дървета. Това означава, че те също така никога няма да се разраснат с поддървета, които не могат да намерят място в някое S-вкоренено дърво. Разликата във възходяща стратегия е това, че нямат надежда за водене до S, или побирайки в всеки техен съсед, са генерирани с напускания, Например левият клон на пространството на търсенето на фигура 10.4 е напълно излишно усилие; това се основава на интерпретиране на "книга" като съществително в началото на изречението въпреки факта,че няма дърво, което да може да доведе до S при тази граматика. Низходящият подход има неговите собствени недостатъци. Докато при него не се губи време с дървета които не водят до S, в този случай се харчат значителни усилия за S дървета, които не съответстват на входа. Забележете, че това са първите четири от шестте дървета на фигура 10.3 всички са оставили клони които не могат да съчетаят думата "книга". Нито едно от тези дървета не може да бъде използвано в синтактичният анализ на това изречение. Тази слабост в низходящите синтактични анализатори възниква от факта, че те могат да изхарчат значителна енергия за да генерират дървета преди да са проверили входа. Възходящите синтактични анализатори, от друга страна, никога не предлагат дървета, които не са поне временни в действителният вход.

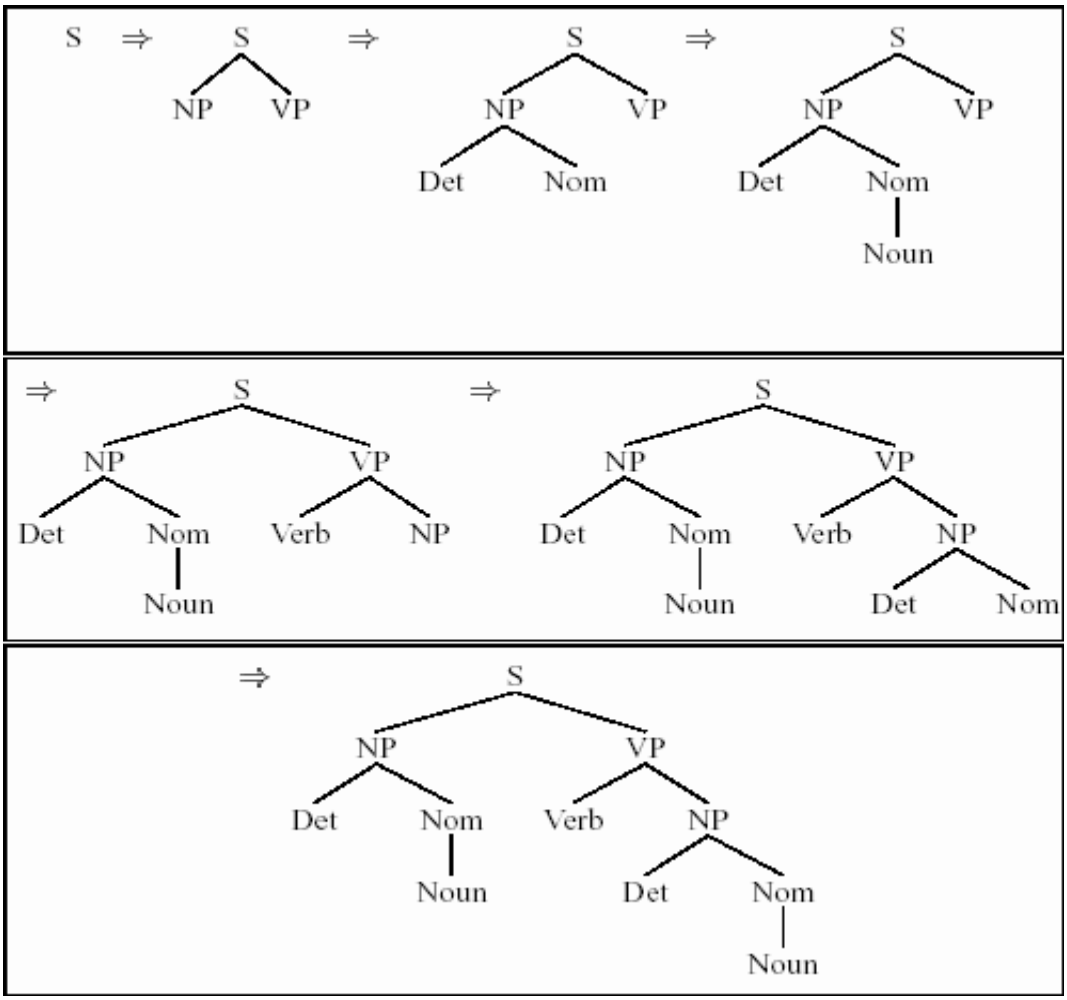
В следващата секция, ние представихме алгоритъмът, който се стреми по-добре да покрие най-доброто от двете дървета на синтактичният анализ.

10.2 НИЗХОДЯЩ СИНТАКТИЧЕН АНАЛИЗ С ВЪЗХОДЯЩА ФИЛТРАЦИЯ

Има което и да било множество направления на обединяване на най-добрите отличителни черти на низходящ и възходящ синтактичен анализ в единствен алгоритъм. Един доста директен подход е приемането на една техника като стратегия за първично управление, използвана да генерира дървета и тогава да принуждава друга техника да филтрира изходи, неподходящи за граматични разбори на слоя. Синтактичния анализатор, който развиваме в тази секция използва стратегия на низходящото управление с увеличаващ се възходящ филтриращ механизъм. Нашата първа стъпка е да развием конкретна реализация на низходящата стратегия указана в последната секция. Способността да се филтрират лоши граматични разбори се базира на възходящото принуждаване, от входа тогава ще бъдат присадени върху този низходящ синтактичен анализатор.

В нашите дискусии за низходящ и възходящ синтактичен анализ ние приехме, че ние бихме изучили всички възможни граматични дървета в паралел. Така всеки слой на търсенето на фигура 10.3 и фигурирайте 10.4 показват всички възможни разширения на граматичните дърветата на предшествуващите слоеве. Макар, че сигурно е възможно да осъществим този метод директно, това влече след себе си типичното използване на недействително количество памет за запас, във времето докато се построяват дърветата.

Това е изключително вярно и затова се използват реалистични граматики, които имат много по-голяма двусмисленост, отколкото миниатюрната граматика на фигура 10.2.



Фигурата 10.5 низходящ дълбочинен произход с граматиката на фигура 10.2.

По-добрият подход трябва да използува дълбочинна стратегия както например, различните определени държавни машини в Главата 2 и Глава 3. Дълбочинният подход разширява пространството на търсенето разширявайки систематично изучаваните нива по едно и също време. Нивото избрано за разширение е най-скоро генерираното. Когато тази стратегия се прилага за дърво, това е противоречиво с входа, търсенето продължава от връщане при скоро генерирано и все още неизучено дърво. Чистия ефект на тази стратегия е синтактичен анализатор, който единствен търси дървета, докато успее или пропадне, преди да се завърне на работа с дървета генерирани по-рано в процеса.

Фигура 10.5. илюстрира такава низходящ, дълбочинен произход използващ граматика на 10.2.

Забележете, че този произход не е напълно решен от спецификацията на

```
function TOP-DOWN-PARSE(input, grammar) returns a parse tree

  agenda ← (Initial S tree, Beginning of input)
  current-search-state ← POP(agenda)
  loop
   if SUCCESSFUL-PARSE?(current-search-state) then
     return TREE(current-search-state)
   else
       if CATEGORY(NODE-TO-EXPAND(current-search-state)) is a POS
then
        if CATEGORY(node-to-expand)
           ⊂
           POS(CURRENT-INPUT(current-search-state)) then
          PUSH(APPLY-LEXICAL-RULE(current-search-state), agenda)
        else
          return reject
      else
        PUSH(APPLY-RULES(current-search-state, grammar), agenda)
   if agenda is empty then
      return reject
   else
      current-search-state ← NEXT(agenda)
  end
```

Фигура 10.6 низходящият, дълбочинен от ляво на дясно синтактичен анализатор

низходяща, дълбочинна стратегия. Има два вида на избор това, че имат ляв неспецифиран който може да доведе до различни произходи: изборът от кой краен възел на дърво да разшири и заяви в коя приложима граматика да приложи правилата. В този произход, левия не разширен краен възел на текущото се разширява първо, и приложимите правила на граматиката да са приложени според тяхната текстова поръчка в граматиката. Решението за разширение на левия не разширен възел на дървото е важен, от него се определя поръчката към кои входящи думи се обърне докато дървото се конструира. Особено, това резултира върху входящите думи на дървото. Втория избор от прилагане на правилата в тяхната текстова поръчка има последици които ще бъдат обсъдени по-късно.

Фигура 10.6 представя алгоритъм на граматичен разбор, който се инициализира като: низходяща, дълбочинна, от ляво на дясно стратегия. Този алгоритъм поддържа дневен ред на търсено състояния. Всяко търсено ниво се състои от частични дървета заедно с указател към следващата въведена дума в изречението. Основната бримка на синтактичния анализатор взема ниво от предната част на дневния ред и произвежда нов комплект от състояния чрез прилагане всички приложими граматични правила, докато дървото се асоциира с това ниво. Този комплектът от нови състояния е добавен към предната част на дневния ред в съответствие с текстовата поръчка на граматичните правила, които бяха използвани за да ги произведе. Този процес продължава докато се получи правилен граматичен разбор на дървото е или дневния ред е изчерпан показвайки, че на входа не може да се направи граматичен разбор.

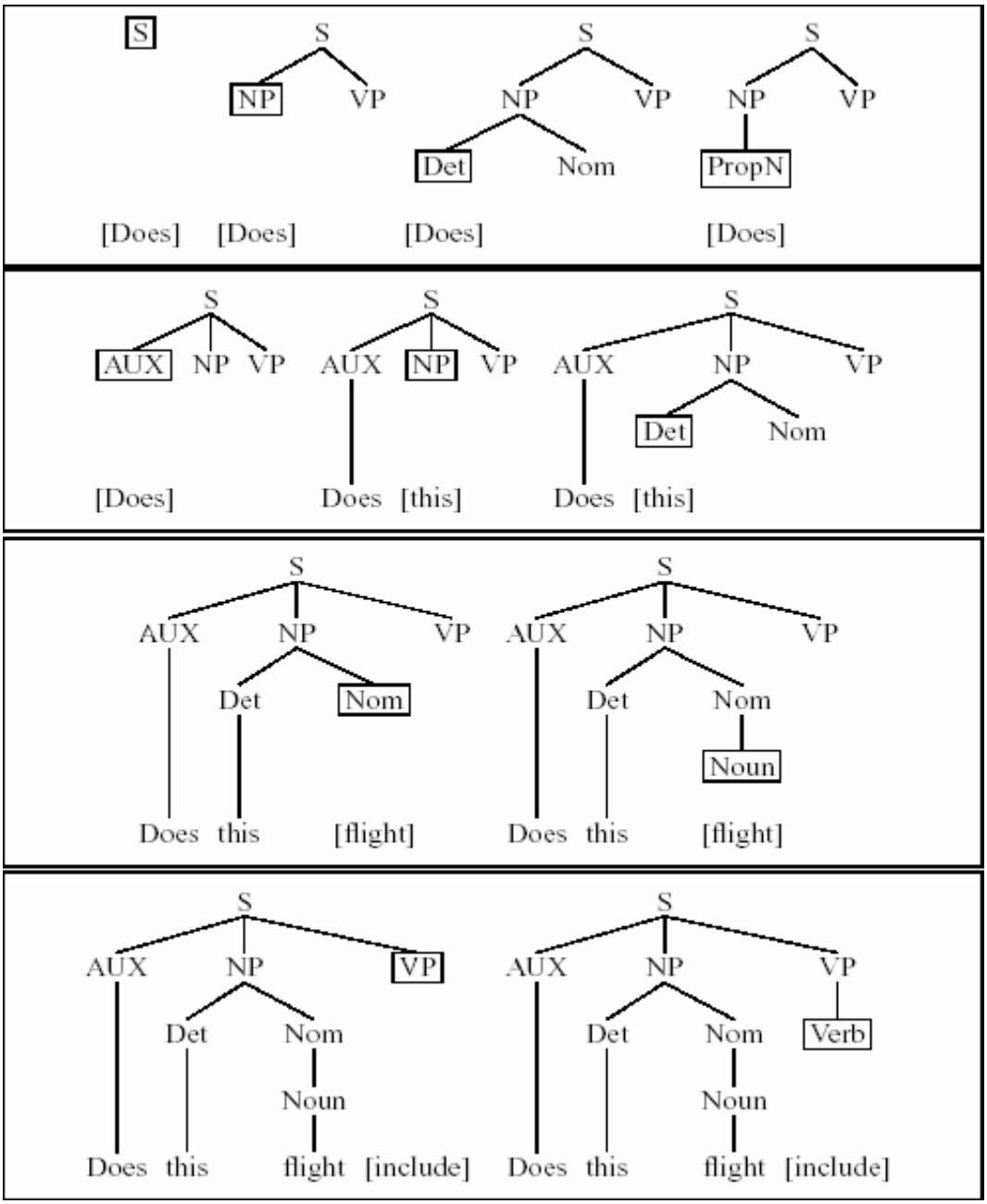
Фигура 10.7 показва последователността на състояния изпитани от този алгоритъм в пътя на синтактичен анализ на следващото изречение:

(2) Този полет включва ли храна?

В тази фигура, възела обикновено бидейки разширен е показан в кутията, докато думата на текущия вход е затворена в скобки. Думи от ляво на затворената в скобки дума вече са били включени в дървото. Синтактичният анализатор започва с безплодно изследване на S NP VP правилото, което в края на краищата пропада тъй като думата "включва" не може да бъде производна на някоя от частите на речта които може да започнат в NP. Синтактичният анализатор, който елиминира S-> NP VP правилото. Следващото ниво на търсене в дневния ред отговаря на S -> Aux NP VP правилото. Веднъж намерено в това ниво, търсенето продължава в дълбочина, отляво надясно оформяйки директно останалата част от произхода. Фигурата 10.7 показва важен качествен аспект на низходящият синтактичен анализатор. Започвайки от корена на граматичното дърво, синтактичния анализатор разширява не-терминалните символи покрай левия ръб на дървото, надолу на думата в основният ляв ъгъл на дървото. Колкото по скоро като думата се включи в дърво, толкова по-бързо указателя на входа ще се придвижи, и синтактичният анализатор ще разшири новия, следващ, ляв, отворен, не-терминален символ, слизайки надолу до новия ляв ъгъл.

Така във всеки сполучлив граматичен разбор текущия вход на дума трябва да служи като първа дума в произхода на не разширения възел, който синтактичният анализатор обработва. Това води до важна последица кое ще бъде полезно в прибавянето на възходяща филтрация. Синтактичният анализатор не би трябвало да разглежда никакви граматични правила, ако текущия вход не може да служи на като първа дума на левия ръб на някой произход от това правило. Ние викаме първата дума покрай левия ръб на произхода на левия-ъгъл на дървото.

Разглеждайки граматичното дърво, VP е показан на фигура 10.9. Дали ще дадем ясна зрителна представа на граматичното дърво за това VP, като триъгълник с думи покрай дъното, думата "предпочита" лежи в долния ляв ъгъл на дървото.

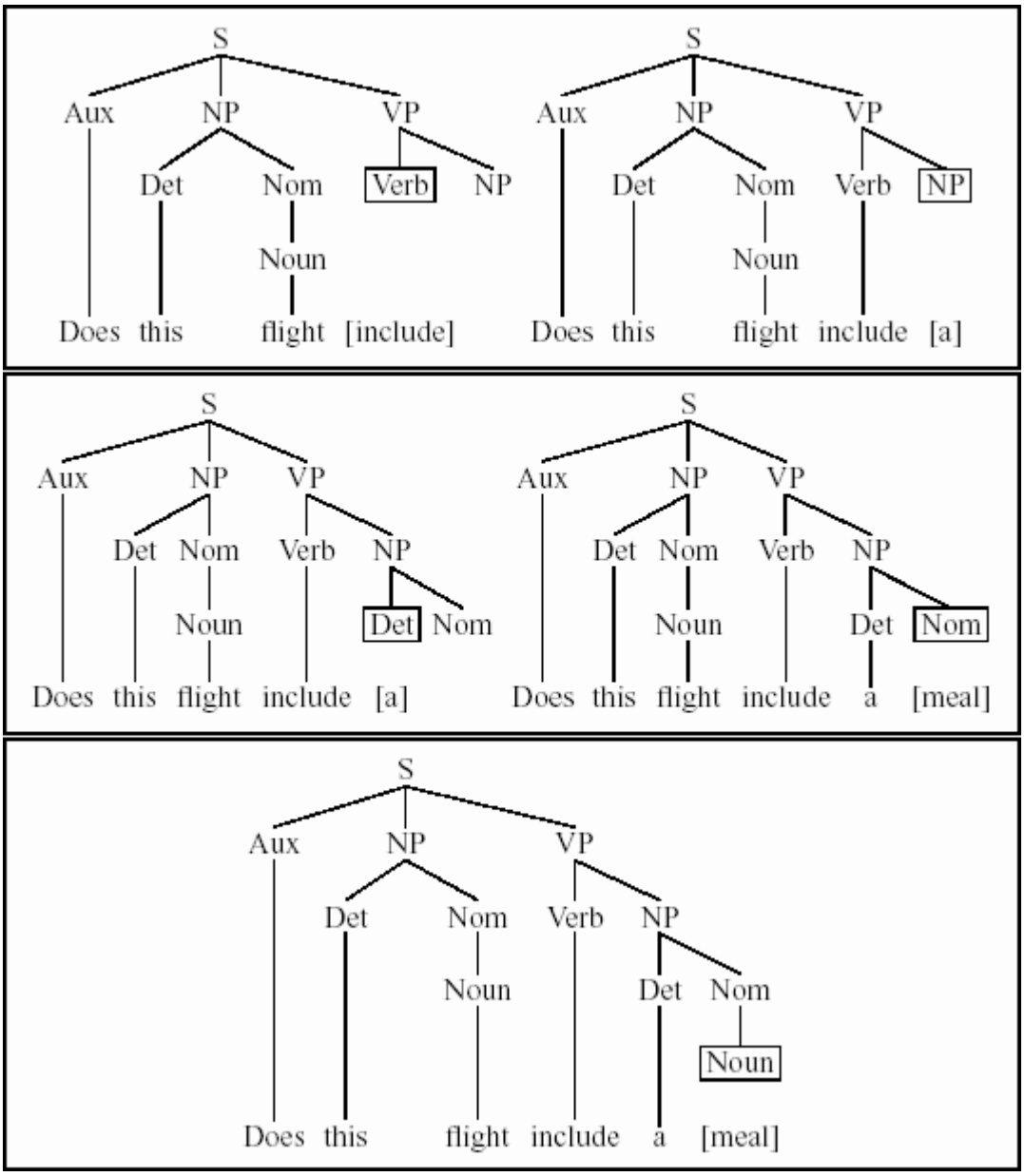


Фигура 10.7 низходящ, дълбочинен, произход от дясно на отляво

Формално, ние можем да кажем, че за не-терминали A и B, B е ляв ъгъл на A, ако следва отношението съдържа:

$$A \xrightarrow{*} B * a$$

С други думи, B може да бъде ляв ъгъл на A, ако съществува произход на A, който започва от B.



Фигура 10.8 низходящ, дълбочинен произхода, отляво надясно продължение

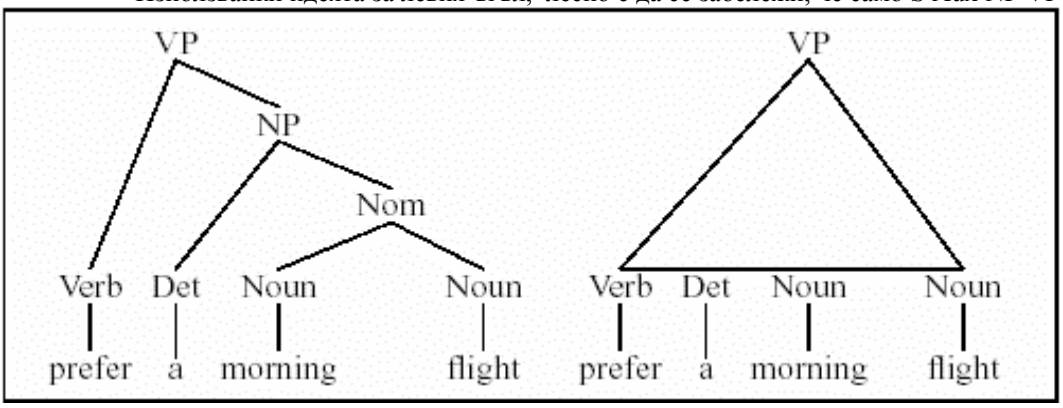
Да се върнем на нашия пример “Този полет включва ли храна”?.Граматиката на фигура 10.2. ни осигурява три правила, с които можем да разширим категорията S:

S -> NP VP

S -> Aux NP VP

S -> VP

Използвайки идеята за левия-ъгъл, лесно е да се забележи, че само S Aux NP VP



Фигура 10.9 илюстрация на идеята за левия-ъгъл.

правилото е добър кандидат понеже думата "Does" не може да служи като ляв-ъгъл, на който и да е NP или VP изискани от други две S правила. Знаейки това, синтактичният анализатор би трябвало да се концентрира върху theAux NP VPrule, без първо конструиране и връщане в изходно състояние извън другите, както това направи с не филтриращия пример показан на фигура 10.7. Информацията има нужда ефективен инструмент като филтър можещ да събере в таблица, която да съдържа верни категории за ляв-ъгъл за всеки не-терминал в граматиката. Когато правилото е разгледано, входа на таблицата за категории започващи от дясната страна на правилото е обърнато. Дали това не съдържа всяка част на речта асоциирана с текущия вход, тогава правилото е отстранено от разглеждане. Следващата таблица е за ляв ъгъл на граматика 10.2.

Category	Left Corners
S	Det, Proper-Noun, Aux, Verb
NP	Det, Proper-Noun
Nominal	Noun
VP	Verb

Търсене в безкрайните пространства

Присъщ проблем с използването на дълбочинното търсене в безкрайното търсещо пространството е, че то е склонно да се потапя надолу по безкрайни пътища, които не осигуряват възможности за да се върнете при по-рано произведени не разширени състояния. Този проблем проявява себе си в низходящите, дълбочинни, от ляво на дясно синтактични анализатори, когато ляво рекурсивните граматики са използвани. С други думи, граматиката е ляво рекурсивна ако съдържа не-терминална категория, която има произход включващ себе си навсякъде покрай неговия най-ляв клон. Примерът на тази ситуация се среща с правилата представени в Главата 9 да овладее собственическа NPs както аерогарата на Атланта.

NP -> Det Nominal

Det -> NP ` s

Тези правила въвеждат лява-рекурсия в граматиката понеже има произход за първия елемент на NP, Det, това има NP като неговата първа съставна част. По-очевидния и общ случай на лява-рекурсия в естествената граматика на езика, включващ незабавно ляво рекурсивни правила. Това са правила на формите $A \rightarrow A\beta$, където първата съставна част на правилната

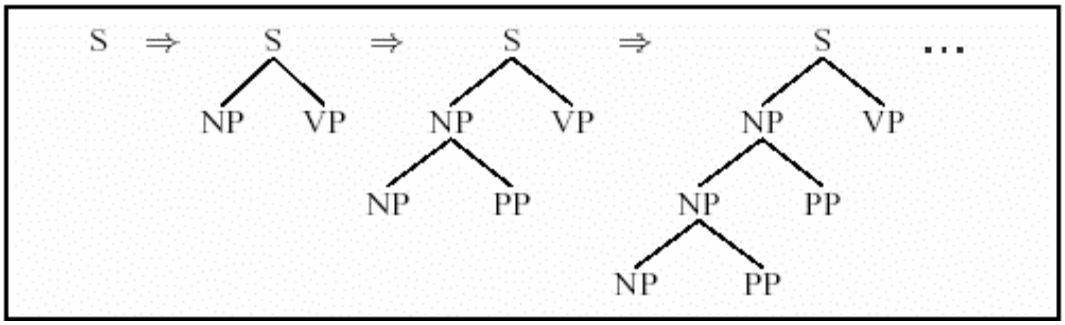
страна е идентична на лявата страна. Следат няколко незабавни ляво-рекурсивни правила, които представят английската граматика в обикновен вид.

NP -> NP PP

VP -> VP PP

S -> S and S

Интуитивно, не е трудно да се види това, че разширението на което и да било ляво-рекурсивен не-терминал би могъл евентуално да доведе низходящ, дълбочинен от ляво на дясно синтактичен анализатор до рекурсивно разширяване на самият не-терминал отново точно същият начин, водещ до безкрайно разширение на дървета. Фигура 10.10 показва вида на разширение, съпровождащ прибавянето на NP -> NP PP правило както първото NP правило в нашата малка граматика.



Фигура 10.10 началото на безкрайно търсене, причинено от ляво-рекурсивни правила

Има два разумни метода за занимаване с лява-рекурсия във връщащ се в изходно състояние низходящ синтактичен анализатор: презаписване на граматиката, и явно управление в дълбочината на търсене по време на синтактичния анализ. Обаче, преди да се обсъдят тези подходи, ние най-напред ще обсъдим много по-простия, очевидния, и за съжаление неправилният подход базиран на представата за “поръчващото правило”. Нашият низходящ синтактичен анализатор прилага правилата на граматиката в текстовият ред, в който те се появяват. Произхода е показан на фигура 10.10, следователно възниква от поставянето на NP -> NP PP правило начело на NP правилата. Разглеждайки какво се случва, когато направим граматичен разбор на NP като “полета от Бостън” примерно, с NP правила редът е както следва.

NP -> Det Nominal

NP -> Proper-Noun

NP -> NP PP

В този случай, след малко връщане, нашия синтактичен анализатор пристига на правилно направения граматичен разбор. По-точно, синтактичният анализатор ще мине през следната последователност:

1. Призовава се първото правило, което отчита “полетът”, то пропада, докато правилото напусне PP частта на неясният вход.
2. Призовава се второто правило и то пропада заради членуването на думата, която не е собствено съществително.
3. Призовава се третото правило, което води до следващата рекурсия (членуване) Призовава се първото правило отново, което отново прави граматичен разбор на думата “полета”.
4. Връща се от рекурсията и успява чрез синтактичен анализ на оставащия вход като PP.

Ключът за този успех се дължи на факта на правилото което поръчва, което винаги желае

да опита основните NP правила преди да призове рекурсивното правило. Всъщност, когато нашият низходящ синтактичен анализатор очаква NP това правило винаги изисква отбягване на

безкрайна рекурсия и връщане на правилния граматичен разбор когато е представено с действителния NP.

За съжаление, има две общи ситуации, където това решение не успява да отбегне безкраен регрес: прави се синтактичен анализ на не граматични входове, и прави граматичен разбор с неправилни очаквания. Като пример на този последен случай, разгледайте това, което се случва, когато се опитаме да направим граматичен разбор нашия любим вход “Летящата книга”. Първото S правило в нашата граматика, S -> NP VP, води до очакването, че би трябвало NP да е в началото на това изречение. За съжаление, когато основните NP правила се провалят за намирането на NP, рекурсивното правило се извиква да води безкрайният регрес. Второто S правило в граматиката, S -> VP, е никога не разглеждайте NP частта от пространството на търсене, което може да не бъде напълно изследвано. Оттогава ние бихме харесали нашите синтактични анализатори за сигурно връщане, ние трябва да намерим други начини за занимаване с рекурсивни правила.

Елиминация на лява рекурсия

Формалното основание за презаписващия подход бе въведено в Главата 9, където това бе показано като често възможен начин за пренаписване на правилата на граматиката в такъв случай гаранцията, която новата граматика приема за точно същият език е като оригинала. За такива граматики се казва, че са слабо еквивалентни, докато не приемат точно същия език, но могат да не назначат същите дървета за еднаквите стрингове в езика. За щастие, възможно е да се елиминира лявата рекурсия от тези общи класове от граматиките, чрез пренаписване на ляво рекурсивна граматика в слабо еквивалентна не-ляво рекурсивна такава. Ключът за отстраняването на лява рекурсия от граматика лежи на методът за елиминиране на преки ляво рекурсивни правила. Това е осъществено, чрез пренаписване на правила на формата $A \rightarrow A\beta$ според следната схема.

$$A \rightarrow A\beta | \alpha \quad \Rightarrow \quad \begin{array}{l} A \rightarrow \alpha A` \\ A` \rightarrow \beta A` | \varepsilon \end{array}$$

В този превод, символа A` представлява нов символ не използван в оригиналната граматика. Това преобразуване променя лявата рекурсия на дясна рекурсия и променя дърветата, като резултат от тези правила от ляв преход на структурите се преминава на правилно разклоняващи се такива. Разгледайте приложимостта на тази схема на рекурсивното правило NP -> NP PP обсъдено по-рано.

NP -> NP PP		NP Det Nominal NP`
NP -> Det Nominal	$\Rightarrow$	NP’ PP NP` | ε

Забележете как този пример приема, че NP -> Det Nominal правилото е единствената достъпна алтернатива за рекурсивното NP правило. В главният случай всички алтернативи за рекурсивното правило трябва да бъдат предвидени. Следващата обща схема може да бъде отнесена към подходящо елиминираме на всички леви рекурсии от даденият ни не терминал. В тази схема, ние групираме всичките презаписвания за A в единствено правило и приемаме, че това не е α начало с A-то.

$$\begin{array}{l} A \rightarrow A\,\beta_1 \mid A\,\beta_2 \mid \cdots \mid A\,\beta_i \mid \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_j \\ \Downarrow \\ A \rightarrow \alpha_1\,A' \mid \alpha_2\,A' \mid \cdots \mid \alpha_j\,A' \\ A' \rightarrow \beta_1\,A' \mid \beta_2\,A' \mid \cdots \mid \beta_i\,A' \mid \epsilon \end{array}$$

С тази схема като инструмент, ние можем да продължим на отстраняването на всички леви рекурсии от граматиката. Завръщането на леви рекурсии е скрито в граматики, които съдържат правила със следната структура:

A -> B C

B -> D E
D -> A F

Тук левите рекурсивни потоци от разширението на A до B и на B до D и обратно към A. По принцип, която и да било граматика с правила на следващата форма къде A може да бъде левият ъгъл на α, ще съдържа лява-рекурсия.

A -> α
.
.
.
B -> A β

Ключът за елиминиране на всички леви рекурсии е най-напред да се презапише граматиката в такъв ред, че да преобразува всички косвени леви рекурсии в непосредствени такива. Това е осъществено от последователно заменяне на избрания ляв ъгъл водещ до рекурсията с неговото възможно разширение. Прилагането на тази техника върху граматика дадена над първите замени на A в D -> A F правило с разширението дадено от A правилото. Това води до следната граматика.

A -> B C
B -> D E
D -> BC F

Заменяйки B в новото правило с неговите разширителни добива следната граматиката с лява рекурсия направена изрична от последното правило.

A -> B C
B -> D E
D -> D E C F

Веднъж такива преобразувания са били прилагани, схемата за непосредствена елимининация на лява рекурсия може да бъде използвана за освобождаване на незащитени Ляво рекурсивни правила. Фигурата 10.11 дава алгоритъм, който систематично наема този двустепенен процес за елиминиране на лява-рекурсия от граматика. То завършва това, чрез единствено помитане през нареденият списък от не терминали в граматиката. На всяка стъпка, правилата се асоциирани с текущия не терминал протичат първи, така че която и да било латентна лява рекурсия е презаписана като непосредствена лява рекурсия; схемата за елиминиране на непосредствена лява рекурсия, тогава е приложена в следните правила.

```
function ELIM-LEFT-RECURSION(grammar, non-terminal-list) returns
grammar
  n ← LENGTH(non-terminal-list)
  for i ← from 1 to n do
    for j ←from 1 to i-1 do
      for each rule (A_i → A_j γ) in grammar do
        TRANSFORM( (A_i → A_j γ) , grammar)
      end
    for each (A_i → A_i δ) in grammar do
        ELIM-IMMEDIATE-LEFT-RECURSION-FROM(A_i, grammar)
    end
  end
  return(grammar)
 procedure TRANSFORM((A → B γ), grammar)
    DELETE((A → B γ), grammar)
    for each rule (B → β) in grammar do
      ADD((A → β γ), grammar)
    end
```

Фигура 10.11 алгоритъм за елиминиране на лява рекурсия от граматика

Управление дълбочината на търсене

За съжаление, това се обръщение има недостатъците на предотвратените от нас, използвани най-естествено представяне на синтактичния феномен във въпрос (и така ние ще видим в главата 15, това може да направи доста трудна семантична интерпретация. Алтернативен подход е да се остави граматиката сама и да се промени стратегията за граматичен разбор. По-точно, ние можем да приложим техника за пряко търсене, която позволява дълбочинните стратегии да отбягнат безкрайна рекурсия породена от ляво рекурсивни правила. Ключът за този подход е прибавянето на дълбочинна граница за търсената процедура. Когато дълбочинната граница е достигнатата, връщане в изходно състояние е спуснато, синтактичния анализатор оставя безрезултатни рекурсиите и се разглеждат алтернативи, които могат да доведат до граматичен разбор. Под такава схема всеки граматичен разбор, който съществува вътре за търсеният периметър определен от границата ще бъде намерен въпреки рекурсивната същност на правилата. Освен това, синтактичният анализатор подходящо ще прекрати работата си, когато се изиска направи граматичен разбор на стрингове, които не са в езика. Разбира се проблемът с използване на фиксирана дълбочинна граница е такъв, че всяко дърво чиито клони са по-дълги отколкото е заложено в границата няма да бъде открито. Синтактичният анализатор специално ще търси цялото пространство вътре в границата и ще отхвърля входящите, докато пространството се изчерпи. Макар, че този отказ може да покаже стринговете, които не са в езика, а също така би могло да се получи случая, в който дърво, което се проверява на входа да лежи всъщност извън периметъра на дефинираната дълбочинна граница. Единно решение на този проблем е да се избере дълбочинна граница основана на анализ на използвана специфична граматика и дължината на даден вход, който е достатъчно дълбок за да позволи синтактичния анализатор да намери, което и да било допустимо дърво. Горният лимит на дълбочинна граница подходяща за която и да било без контекстна граматика K(N+1), където K е числото на не терминали в граматиката и N е дължината на входа. Ако има дърво за дадения вход, тогава най-дългия клон за такова дърво не може да превиши тази дълбочина. Погледнете упражнението?? за
обсъждане на този случай. За съжаление, този граница обикновено е толкова голяма, че ще позволи на синтактичния анализатор да се върти около до голяма степен изчерпаните непроизводителни области на търсенето пространството преди да се придвижи към по- плодородни области. Просто изменение на този подход, наречен итеративно задълбочаване, решава този проблем като позволява на синтактичния анализатор да изучи първо по-плитки части на търсенето пространство. Итеративното задълбочено търсене започва с дълбочинна граница 1 и я увеличава по време на следващите многократни дълбочинни търсения, докато или решението е намерено или максималната дълбочинна граница е достигната, отчитайки това не може да се намери граматичен разбор.

Удивително, работата направена посредством итеративно задълбочаване търсене в дълбочина на N не е много повече отколкото работата направена от единствено дълбочинно търсене същата дълбочина. Това е така, заради задълбоченото търсене, времето прекарано, в което и да било от нивата е доминиращо над броя на дърветата, които са били изследвани на границата на това ниво. Броят на алтернативите, обаче в тази граница се увеличава експоненциално със средния коефициент за разклоняване в граматиката, свършената работата на ниво N + 1 е по-голяма отколкото сумата на работата на търсенето от 1 до N. Упражнения??и?? изучавате темата за итеративни-задълбочаващи се синтактични анализатори по-подробно.

Двусмисленост, неяснота

Една сутрин застрелях един слон в моята пижама. Как той е влязъл в нея аз никога няма да разбера.

Groucho Marx

Низходящият синтактичен анализатор даден на фигура 10.6 произволно връща първото граматично дърво намерено за даденият вход. Обаче, двусмислената същност на езика и ненасочената същност на нашите синтактични анализатори, има незначително основание за вярване, че първият открит граматичен разбор ще бъде подходящ за поставената задача. Възможно решение на този проблем е завръщането на всички възможни граматични разбори за дадения вход. Изменянето нашите основни алгоритми за връщане на всички възможни граматични разбори е просто начин за свързване на граматични разбори, които са намерени и продължаване с обработка като че ли не са намерени. Когато търсенето пространство е било изчерпано, списъка на всичките намерени дървета просто се връща. Това е до следваща обработка или до решението на човешки аналитик, че върнатите граматични разбори са правилни. За съжаление, ние почти сигурно не искаме всички възможни граматични разбори от смисъла, много двусмислен, граматики с широк-обсег са използвани в практическите риложения. Причината за това се дължи в потенциално експоненциалното число на граматични разбори, които са възможни за даденият вход. Разгледайте следващия ATIS пример:

(3)Покажете ми храната на полета UA 386 от Сан-Франциско на Денвър.

Когато нашата извънредно малка граматика е увеличена с рекурсивния VP -> VP PP и NP -> NP PP правила представени по-горе, трите предложени фрази накрая на това изречение заговорничат да дадат общата сума на 14 граматични дървета за това изречение.

Church и Patil (1982) показаха това, че броят на граматични разбори на изречения на този тип израстъци е на същото ниво като поставените в скоби аритметични изрази. Такива проблеми поставени в скоби, по ред, са познати като разрастващи се експоненциално в съответствие с извиканите Каталанови числа:

$$C(n) = \frac{1}{n+1}\binom{2n}{n}$$

Следващата таблица показва броя на граматичните разбори за проста съществителна фраза като функция от броя на проследяване на предложени фрази. Когато може да бъде видяна, този вид двусмисленост може много бързо да направи неблагоразумието да съхрани всеки възможен граматичен разбор наоколо.

Number of PPs	Number of NP Parses
2	2
3	5
4	14
5	132
6	469
7	1430
8	4867

Има два основни начина извън тази дилема: използвайки динамично програмиране за да се експлоатират правила в пространството на търсенето така, че общите под части да са произведен само по веднъж, така се намаляват някои от разходите се асоциирани с двусмисленост, и увеличаване стратегията за търсене на синтактичният анализатор с евристика, което доведе до

вероятни граматични разбори. Динамичното програмиране на подход ще бъде изследвано в следващата секция, докато евристичните търсещи стратегии ще се разгледат в глава 12.

10.3. ДИНАМИЧНИ ПРОГРАМИРАЩИ ПОДХОДИ КЪМ СИНТАКТИЧЕН АНАЛИЗ

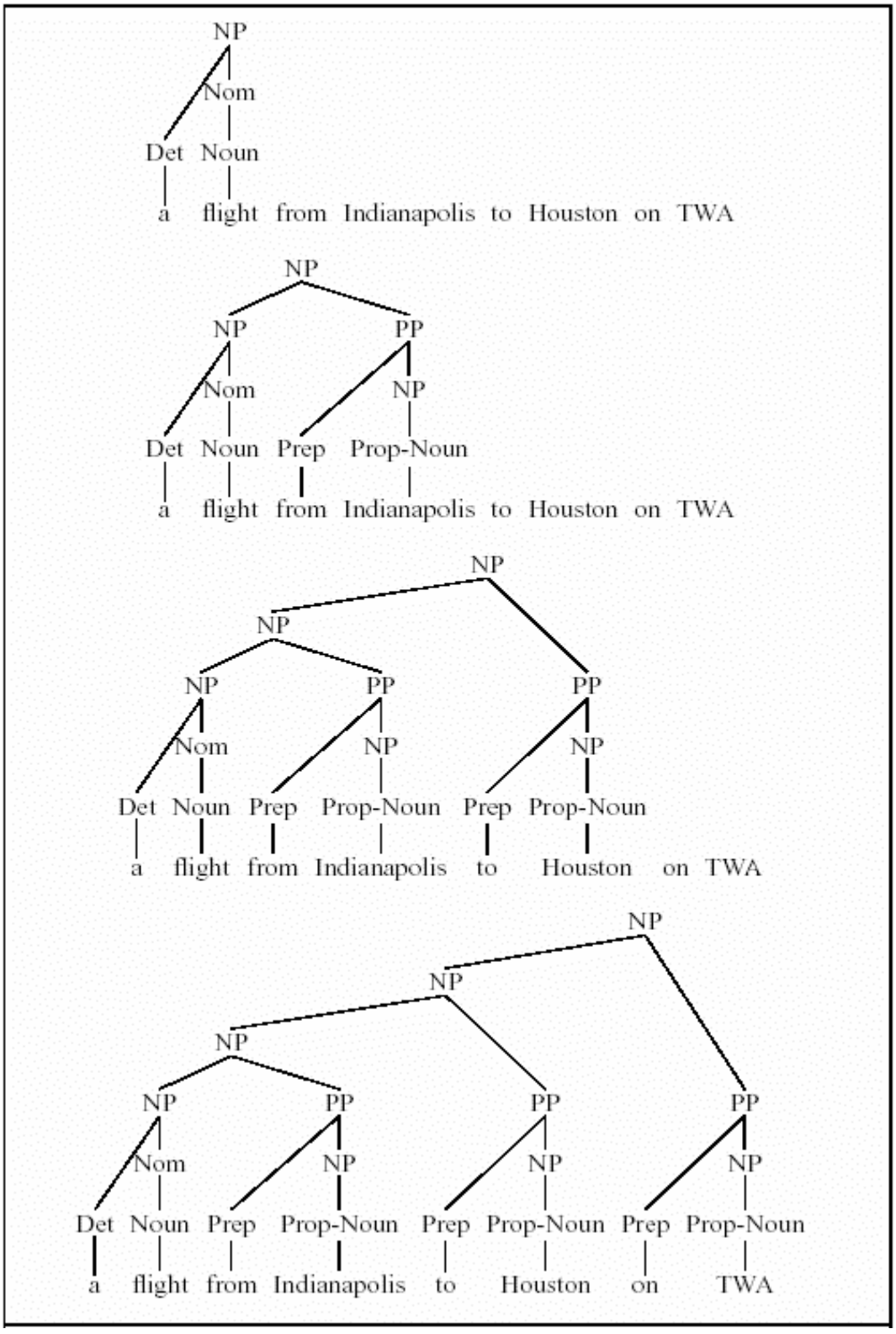
Въпреки опитите ни да се възползуваме от принудителните действия осигурени от низходящите граматични очаквания и от входа, алгоритъма зададен ни в секцията 10.2 е все още е напълно неефикасен в много случаи. Тази неспособност възниква първоначално от факта, че синтактичният анализатор често строи действителни дървета за части от входът тогава ги отхвърля, докато ги връща в изходно състояние само да се намерят и възстанови отново. Разгледайте процеса включен в откриването на граматичен разбор за следващото NP.

(4) a flight from Indianapolis to Houston on TWA - (полетът от Индианаполис до Хюстън на TWA)

Предпочетения граматичен разбор, кой е първият намерен първи от синтактичния анализатор присъствува в секция 10.2, е показан като дъното дърво на фигура 10.12. Докато има 5 ясни граматични разбора на тази фраза, ние ще се фокусираме върху нелепата сума на многократната работа замесена във възстановяване граматичният разбор. Поради начина правилата са обърнати към нашият низходящ, дълбочинен, от ляво на дясно подход, синтактичния анализатор ръководи най-напред малки граматични дървета, които пропадат, тъй като те покриват всички входове. Тези последващи неуспехи водят до връщащи се в изходно състояние събития, които пък водят до граматични разбори, покриващи все повече и повече на входове. Последователността на дървета предприети от нашия низходящ синтактичен анализатор е показан на фигура 10.12. Тази фигура ясно илюстрира вида на глупавото удвояване на работата, която възниква в резултата на връщането в изходно състояние на подходи. С изключение на неговия най-горен компонент, всяка част от крайното дърво е производна повече от веднъж. Следващата таблица илюстрира колко производни всеки от главните избиратели в крайното дърво има. Забележете, че пример е специфичен за низходящ синтактичен анализ, аналогични примери на непотребното усилие съществуват и за възходящ синтактичен анализ също.

a flight	4
from Indianapolis	3
to Houston	2
on TWA	1
a flight from Indianapolis	3
a flight from Indianapolis to Houston	2
a flight from Indianapolis to Houston on TWA	1

От какво се нуждае систематичния път да си спомня и използва отново действителни поддървета, където са открити на входа. За щастие, алгоритмичният подход известен като динамично програмиране ()? осигурява структура за решаване на такива проблеми. Динамичното програмиране се обръща към систематично попълване на таблиците на решения за субпроблеми. Когато завърши, таблиците съдържащи решението на всичките субпроблеми, нуждаещи се от решение на проблема като цяло. В случаят на синтактичен анализ, такава таблица се използвана за съхранение на поддървета за всеки от различните избиратели, които са открити на входа. Ефективността на печалбата възниква от факта, че тези поддървета са открити веднъж, съхранени, и след това използвани във всички граматични разбори изискани от избирателя.


NP
Nom
Det Noun
a flight from Indianapolis to Houston on TWA
NP
NP PP
Nom NP
Det Noun Prep Prop-Noun
a flight from Indianapolis to Houston on TWA
NP
NP
NP PP PP
Nom NP NP
Det Noun Prep Prop-Noun Prep Prop-Noun
a flight from Indianapolis to Houston on TWA
NP
NP
NP
NP PP PP PP
Nom NP NP NP
Det Noun Prep Prop-Noun Prep Prop-Noun Prep Prop-Noun
a flight from Indianapolis to Houston on TWA

Фигура 10.12 Удвоено усилие причинено от връщане в изходно състояние в низходящ синтактичен анализ

Алгоритъм на Earley

Алгоритъмът на Earley (Earley, 1970) използва динамичен програмиращ подход на ефективно изпълнени паралелни, низходящи търсения на вида обсъден в секция 1. Като с много динамични програмиращи решения, този алгоритъм намалява очевидно експоненциално-времевия проблем до многочленно-времеви такъв, чрез елиминиране на повтарящото се решение на субпроблеми присъщи на връщането в изходно състояние подходи. В този случай, динамичния програмиращ подход води до неблагоприятното поведение на $O(N^3)$, където N е броят думи на входа. Приличащ на Viterbi алгоритъм, обсъден в Главата 6, сърцевините на Earley алгоритъмът съдържат единствено преминаване от ляво на дясно, което запълва масив наречен диаграма, която има N+1входа. Елементите на диаграмата съдържат списъци на състоянията представляващи дървета произведени от низходящите и възходящите свидетелства достъпни при точка на входа. Когато завърши, масива компактно кодира всичките възможни граматични разбори на входа. Важно, всяко възможно под дърво е представено само веднъж и така може да се сподели от всичките граматични разбори, които се нуждаят от него. Индивидуалните състояния съдържани за всяко влизане на диаграмата се състоят от под дърво, отговарящо на единствено граматично правило, информация относно прогреса за завършването на това под дърво, и позицията на под дървото с отношение на входа. Нагледно, ние ще използуваме точка за правилната страна на състоянието на граматичното правило показващо напредъка направен в разпознаването му. Позицията на състоянието имайки предвид входа желае да представи две индикации, където състоянието започва и където точката и лежи. Разгледайте следващите три примерни състояния, кои бъдат между онези създадени от Earley алгоритъм в курса на синтактичен анализ на следващия пример:

(5)Летящата книга.

$$S \rightarrow \bullet\ VP,\ [0,0]$$
$$NP \rightarrow Det \bullet\ NOMINAL,\ [1,2]$$
$$VP \rightarrow V\ NP \bullet,\ [0,3]$$

Първата състояние с точката от ляво на неговия избирател, представлява низходящо предсказване за този подробен вид на S. Първите 0 показват, че избирателят предсказан от състоянието би трябвало да започне от началото на входа; втората 0 отразява факта това, че точката лежи на начало също. Второто състояние, създадено в по-късен етап от обработката на това изречение показва, че NP започва в позицията 1, това, че Det успешно е направил граматичен разбор и това, че Nominal очаква следващата позиция. Третото състояние с неговата точка на дясно от всички двамата си избиратели, представляват сполучливото откритие на дърво отговарящо на VP, което обхваща целия вход. Фундаменталното действие на синтактичен анализатор на Earley трябва да мине през N+1 комплекта на състояния в диаграмата на стила от ляво на дясно, обработка на състояния за всеки комплект в ред. На всяка стъпка, един от трите оператора описва Предишни приложения на всяко състояние, зависещо от неговия статус. Във всеки случай, това води до прибавянето на нови състояния до края, на което и да е или следващия комплект на състояния в диаграмата. Алгоритъмът винаги премества напред през диаграмата създаващите допълнения, както му отиват; състояния никога не са отстранявани и алгоритъма никога не се връща в изходно състояние на предшествуваща диаграма. Присъствието на състояние S ->α•,[0,N] в списъка на състояния в последната входяща диаграма показва сполучлив граматичен разбор. Фигурата 10.13 дава пълния алгоритъм.

Предсказател (Предиктор)

Както би могло да се предположи за името му, работата на ПРЕДСКАЗАТЕЛЯ е да създаде нови състоянията представящи низходящи очаквания произведени по време на процеса на синтактичен анализ. Предсказателят е отнесен към, което и да било състояние, което има не-терминал отдясно на точката, която не е част на речева категория. Това приложение резултира в създаването на едно ново състояние за всяко алтернативно разширение на този не-терминал осигурен от граматиката. Тези нови състояния са поставени в същата диагрмен вход като генериращото състояние. Те започват и завършват на този етап във входът, където генерирането на състояние завършва. Например прилагане на предиктора за състоянието S -> • VP, [0,0], води до прибавянето на състоянията VP -> • Verb,[0,0] и VP-> • Verb NP, [0,0] на първият диаграмен вход.

Скенер

Когато състоянието има части на речева категория на надясно от точката, СКЕНЕРА се извиква да изпита входа и да включи състоянието отговарящо на предсказаната отчасти на речева диаграмата. Това е осъществено чрез създаването на ново състояние от входящото състояние с точката предшестваща предсказаната входяща категория. Забележете, че в следващата схема само части на речта на дума, които са предсказани от някое състояние намерило своя път в диаграмата.

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

  ENQUEUE((γ → • S, [0,0]), 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)

procedure PREDICTOR((A → α • B β, [i,j]))
    for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
        ENQUEUE((B → • γ, [j,j]), chart[j])
    end

procedure SCANNER((A → α • B β, [i,j]))
    if B ⊂ PARTS-OF-SPEECH(word[j]) then
        ENQUEUE((B → word[j], [j,j+1]), chart[j])

procedure COMPLETER((B → γ •, [j,k]))
    for each (A → α • B β, [i,j]) in chart[j] do
        ENQUEUE((A → α B • β, [i,k]), chart[j])
    end

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

Фигура 10.13 Алгоритъм на Earley

Скенера тогава записва думата “книга” като глагол за да усети съчетаването на очакването определено от текущото състояние. Това води до създаване на ново състояние

VP -> Verb • _NP, [0,1].

Завършвач

Завършвачът се отнася за състояние когато неговата точка е достигнала правилният край на правилото. Интуитивно, присъствието на такова състояние представлява факта, че синтактичния анализатор има успешно открита подробна граматична категория над някой диапазон на входа. Целта на Завършвача е да намери и да придвижи напред всички по-рано създадени състояния това,

които са били разгледани за тази граматична категория в позицията на входа. Новите състояния са създадени, чрез копиране на старите състояния, напредването на точката върху очакваната категория и инсталиране новото състояние в текущият диаграмен вход. Например, когато състоянието NP -> Det NOMINAL •, [1,3], е протекло, завършвачът оглежда за завършени състояния в 1 очакващи NP-то. В конкретния пример ще намерите състояние VP -> Verb • NP, [0,1] създадени от скенера. Това води до добавяне на завършено състояние VP -> Verb NP •, [0,3].

Примери

Фигурата 10.14 показва последователността на състояния създадени през пълната обработка на Пример 1. Алгоритъма започва от зародиша на диаграмата с низходящо очакване за S. Това е осъществено от допълване на бутафорното състояние γ -> • S, [0,0] за Диаграма 0. Когато това състояние се обработена, то е преминало за водена на ПРЕДСКАЗАТЕЛЯ до създаването на трите състояния представящи предиктори за всеки възможен тип на S, и транзитивно на състояния за всички от леви ъгли на тези дървета. Когато състоянието VP -> • V NP, [0,0] е обработено, СКЕНЕРА се извиква отново. Обаче, този път новото състояние не е прибавено понеже това би било идентично на друго състояние в диаграмата. Когато всичките състояния на Диаграма[0] са били обработени, алгоритъма се придвижва на Диаграма[1], където намира състояние представляващо глагол.Това е завършеното състояние с неговата точка на надясно от неговия избирател и следователно е преминато на Завършвача. Тогава Завършвачът намира двете по-ранни съществуващи състояния VP състояния очакващи глагола в тази точка на входа. Тези състояния са копирани с техните точки усъвършенствувани и добавени към Диаграма [1]. Завършеното състояние на непреходен VP води да създаването на повелително S състояние. Освен това, точката в транзитивната фраза на глагола води до създаването на двете състоянията предсказани от NP-то.

Chart[0]

$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
$NP \rightarrow \bullet Det\ NOMINAL$	[0,0]	Predictor
$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
$S \rightarrow \bullet VP$	[0,0]	Predictor
$VP \rightarrow \bullet Verb$	[0,0]	Predictor
$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor

Chart[1]

$Verb \rightarrow book \bullet$	[0,1]	Scanner
$VP \rightarrow Verb \bullet$	[0,1]	Completer
$S \rightarrow VP \bullet$	[0,1]	Completer
$VP \rightarrow Verb \bullet NP$	[0,1]	Completer
$NP \rightarrow \bullet Det\ NOMINAL$	[1,1]	Predictor
$NP \rightarrow \bullet Proper\text{-}Noun$	[1,1]	Predictor

Chart[2]

$Det \rightarrow that \bullet$	[1,2]	Scanner
$NP \rightarrow Det \bullet NOMINAL$	[1,2]	Completer
$NOMINAL \rightarrow \bullet Noun$	[2,2]	Predictor
$NOMINAL \rightarrow \bullet Noun\ NOMINAL$	[2,2]	Predictor

Chart[3]

$Noun \rightarrow flight \bullet$	[2,3]	Scanner
$NOMINAL \rightarrow Noun \bullet$	[2,3]	Completer
$NOMINAL \rightarrow Noun \bullet NOMINAL$	[2,3]	Completer
$NP \rightarrow Det\ NOMINAL \bullet$	[1,3]	Completer
$VP \rightarrow Verb\ NP \bullet$	[0,3]	Completer
$S \rightarrow VP \bullet$	[0,3]	Completer
$NOMINAL \rightarrow \bullet Noun$	[3,3]	Predictor
$NOMINAL \rightarrow \bullet Noun\ NOMINAL$	[3,3]	Predictor

Фигура 10.14 Последователности на състояния създадени в диаграмата докато синтактичен анализ на "Летящата книга". Всяко влизане показва състоянието, неговата начална и крайна точка, и функцията Earley, която го постави в диаграмата.

Възстановяване на граматични дървета от диаграма

Версията на алгоритъма на Earley точно описва в действителност не разпознаването на синтактичният анализатор. След преработването, действителното изречение ще напусне състоянието S -> α •, [0,N] в диаграмата. За съжаление и ние нямаме начин за възстановяване структурата на този S. Да обърнем този алгоритъм в синтактичен анализатор, ние трябва да сме способни да извлечем индивидуални граматични разбори от диаграмата. За да направим това, представянето на всяко състояние трябва да се увеличава с допълнително поле на запасена информация относно завършените състояния на тези генерирани избиратели. Тази информация

може да е събрана от създаване на проста промяна на Завършвача. Отново го извикваме, Завършвача създава нови състояния от придвижване много стари непълни състояния. Единствената промяна необходима Завършвача да изтласка указател на наскоро открито състояние на списъка на наследници на новото състояние. Възстановеното граматично дърво от диаграмата е в такъв случай просто рекурсивно възвръщане започващо със състояние (или състояния) представляващ пълен S в края на входящата диаграма. Фигура 10.15 показва диаграмата произведена от подходящо обновена от Завършвача. Забележете, че ако има експоненциален брой на дървета за A при изречение, Алгоритъма на Earley не може да ги върне всички в многочленното време. Най-доброто, което може да се направи е построяването на диаграма в многочленното време.

Chart[0]

S0	$\gamma \rightarrow \bullet S$	[0,0]	[]	Dummy start state
S1	$S \rightarrow \bullet NP\ VP$	[0,0]	[]	Predictor
S2	$NP \rightarrow \bullet Det\ NOMINAL$	[0,0]	[]	Predictor
S3	$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	[]	Predictor
S4	$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	[]	Predictor
S5	$S \rightarrow \bullet VP$	[0,0]	[]	Predictor
S6	$VP \rightarrow \bullet Verb$	[0,0]	[]	Predictor
S7	$VP \rightarrow \bullet Verb\ NP$	[0,0]	[]	Predictor

Chart[1]

S8	$Verb \rightarrow book \bullet$	[0,1]	[]	Scanner
S9	$VP \rightarrow Verb \bullet$	[0,1]	[S8]	Completer
S10	$S \rightarrow VP \bullet$	[0,1]	[S9]	Completer
S11	$VP \rightarrow Verb \bullet NP$	[0,1]	[S8]	Completer
S12	$NP \rightarrow \bullet Det\ NOMINAL$	[1,1]	[]	Predictor
S13	$NP \rightarrow \bullet Proper\text{-}Noun$	[1,1]	[]	Predictor

Chart[2]

S14	$Det \rightarrow that \bullet$	[1,2]	[]	Scanner
S15	$NP \rightarrow Det \bullet NOMINAL$	[1,2]	[S14]	Completer
S16	$NOMINAL \rightarrow \bullet Noun$	[2,2]	[]	Predictor
S17	$NOMINAL \rightarrow \bullet Noun\ NOMINAL$	[2,2]	[]	Predictor

Chart[3]

S18	$Noun \rightarrow flight \bullet$	[2,3]	[]	Scanner
S19	$NOMINAL \rightarrow Noun \bullet$	[2,3]	[S18]	Completer
S20	$NOMINAL \rightarrow Noun \bullet NOMINAL$	[2,3]	[S18]	Completer
S21	$NP \rightarrow Det\ NOMINAL \bullet$	[1,3]	[S14,S19]	Completer
S22	$VP \rightarrow Verb\ NP \bullet$	[0,3]	[S8,S21]	Completer
S23	$S \rightarrow VP \bullet$	[0,3]	[S22]	Completer
S24	$NOMINAL \rightarrow \bullet Noun$	[3,3]	[]	Predictor
S25	$NOMINAL \rightarrow \bullet Noun\ NOMINAL$	[3,3]	[]	Predictor

Фигура 10.15 Последователности на състояния създадени в диаграмата докато синтактичния анализ "Летящата книга" включва структурна информация

10.4. ОЦЕНЯВАНЕ НА СИНТАКТИЧНИ АНАЛИЗАТОРИ И ГРАМАТИКИ

Какво значи за входа да бъде приет или отхвърлен от синтактичен анализатор? Теоретично, това значи, че изречението което и да е, или не участник на езикът определен от граматиката, който синтактичният анализатор използва. На практично, обаче, има хубаво брой на алтернативни начини да се интерпретира резултата от синтактичен анализ. Тези алтернативи произлизат от характеристики на двата основни елемента, които определят който и да било резултат от синтактичен анализ: синтактичния анализатор или граматиката. За синтактични анализатори, ние определяме пълнота и правилност.... Пълнотата означава, че синтактичния анализатор ще намери действителният граматичен разбор за вход ако съществува. Правилността означава, че всички дървета върнати за входа ще бъдат правилни. Алгоритъмът на Earley е пълен и правилен. За граматики, обхват и коректност.... Има три начина да се категоризират оценените като не-граматични от синтактичният анализатор, който е завършил правилно:

1. Изреченията са не-граматични според някои стандарти.
2. Изречения които са, всъщност, граматични но наемат граматични конструкции, които са извън обхвата на граматиката използвана от синтактичния анализатор.
3. Изречения, които са граматични, но са оценени като не-граматични, заради недостатъци в граматиката използвана от синтактичния анализатор.

Аналогично, изречения, които са оценени като граматични от синтактичен анализатор могат да бъдат категоризирани в две категории:

1. Правилно, съгласно някой стандарт.
2. Не граматично, но оценено правилно заради грешки в граматиката.

Граматичните контрольори в системите на съвременната текстообработка работят, чрез опитване за да се направи граматичен разбор на всяко изречение в документа според граматика на езика, който е използван. Изреченията на които не може да са направи граматичен разбор са сигнализирани. В оценяването на граматични контрольори, различните под категории указани по-горе може да е комбинират в две прости метрики:

1. Лъжливата положителна норма – нормата в която правилни изречения са сигнализирани като не граматични.
2. Лъжливата отрицателна норма – нормата в която неправилни изречения са допуснати директно дори без да бъдат записани.

За съжаление на търгуващите отдели, тази две метрики може рядко да бъдат оптимизиран в едно и също време. Гъвкавата граматика, както например, S -> Word S | ε на ще доведе до 0% лъжлива положителна норма и 100% лъжлива негативна норма. От друга страна,, много ограничителна граматика ще хване повече не граматични изречения, близо до понижаване на лъжливата отрицателна норма, докато в същото време се определят много правилни изречения като не граматични. В граматиката контролната област, открива на правилното смесване на критики за да се удовлетворят потребителските потребности.