# 空間解析器における効率的な探索手法

An Efficient Analysis Technique in Spatial Parsing

飯塚 和久 † 志築 文太郎 †† 田中 二郎 ††
Kazuhsia IIZUKA Buntarou SHIZUKI Jiro TANAKA

† 筑波大学大学院 工学研究科
Doctoral Program in Engineering, University of Tsukuba
†† 筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
{iizuka,shizuki,jiro}@iplab.is.tsukuba.ac.jp

フローチャート，E–R 図，オブジェクト図などの図式は，ある規則の基に，矩形や線などの基本図形要素を組み合わせて描かれる．この規則を,「図形文法」として定義することができる．また，図形文法に基づき，入力された図式を解析する「空間解析器」を作ることができる．図形文法の 1 つである CMG においては，Chok らの提案した解析手法が知られているが，解析は非常に遅いものであった．Balt はルール（規則）の制約条件の一部を利用することで，これを改善したが十分ではなかった．

我々は，これをさらに改良し，高速な解析を実現する．まず，Chok らの提案したルール適用法を見直し，解析におけるトークンの状態を利用して適用すべきルールを決定し，不要なルール適用を削減する．また，Balt の手法を拡張し，ルールの制約条件を利用して，あてはまる組合せを求め，利用されない要素を除いた探索を行う．これらにより，Chok や Balt らの解析手法に比べ高速な解析が実現された．

## 1 はじめに

フローチャート，E–R 図，オブジェクト図などの図式は，ある規則の基に，矩形や線などの基本図形要素（トークン）を組み合わせて描かれる．この規則は文法として定義することができ，この文法を，テキストのプログラミング言語の文法に対して「図形文法」と呼ぶ．図形文法としては，Positional Grammars[1]，Relational Grammars[2]，Constraint Multiset Grammars(CMG)[3, 4] などが提案されている．

テキストのプログラミング言語と同様に，図形文法に基づいて描かれた図式（図形言語）を解析する「空間解析器」を作ることができる．文法から空間解析器を自動生成するシステムも提案されており，VLCC[5]，SPARGEN[6]，Penguins[7]，恵比寿 [8] などがある．

恵比寿では，CMG に基づく空間解析器を用いることで，動的に入力・編集される図式を，インタラクティブに解析することができる．我々は，これを拡張し，空間解析器を用いた図形エディタの記述 [9, 10] や，ジェスチャによる図式の入力・編集 [11] を提案している．しかし，これまでの空間解析器では解析が非常に遅く，トークン数が多くなった場合にリアルタイムに処理を行うことが困難であった．本論文では，この問題を解決する，効率的な解析アルゴリズムを提案する．

## 2 CMG と既存の解析手法

CMG における生成規則（ルール）は以下のようになる．

T ::= $T_1$, $T_2$,...,$T_k$ where (
*Constraints*
) {
*AttributeAssignments*
}

これは，RHS の記号列 $T_1$, $T_2$,...,$T_k$ に対応するトークンが *Constraints*（制約条件）を満たしたときに，LHS で示されている記号 T に還元されることを表している．T の属性値は，*AttributeAssignments* で決定される．

図 1 のような「計算の木」を表す図形言語の例では，次に示す 2 つのルールで記述される．

```
<< ルール 1 >>
n:Node := c:Circle, t:Text where (
  c.mid == t.mid &&
  isValue(t.text)
```

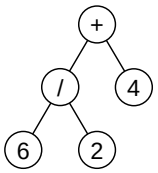


図 1: 図形言語の例（計算の木）

```
) {
  n.mid := c.mid
  n.mid_x := c.mid_x
}

<< ルール 2 >>
n:Node := c:Circle, t:Text, l1:Line,
     l2:Line, n1:Node, n2:Node where (
  c.mid == t.mid &&
  isOperand(t.text) &&
  c.mid == l1.start &&
  c.mid == l2.start &&
  l1.end == n1.mid &&
  l2.end == n2.mid &&
  n1.mid_x < n2.mid_x
) {
  n.mid := c.mid
  n.mid_x := c.mid_x
}
```

ルール 1 は，木の葉にあたる一つのノードを表している．円と文字列が存在し，その中心点が一致していて，テキストが数字であった場合は，ノードに還元されることを示している．ルール 2 は，木のノードを再帰的に定義するために，節にあたる円と演算子，2 つのノードとそれらを結ぶエッジから新しいノードを定義している．

CMG に基づく解析は，適用すべきトークンの集合（トークンデータベース）に対して，ルールに基づいてトークンを還元することにより行われる．

ルールのうち 1 つが選択され，その RHS の各記号に対応する「トークンの組合せ」をトークンデータベースにあるトークンから生成し，それらすべてに対して制約条件をチェックする．この過程を「ルール適用」と呼ぶ．ルール適用において，制約条件を満たすトークンの組合せが見つかった場合，そのトークン列は LHS の記号に還元される．また，どのトークンの組合せも制約条件に合致しなかった場合は，ルール適用は失敗となる．ルール適用は，図形言語を定義するすべてのルールに対して繰り返し行われ，トークンが還元できなくなった時点で，解析が終了する．

ルールが複数存在した場合，ルール適用の順序が解析の高速化に重要になってくる．Chok らは，ルール間の依存関係を利用したルールの適用順序を提案している [12]．ルール間には，LHS の記号と RHS の記号により依存関係がある．すなわち，終端記号を非終端記号に還元し，さらに別のルールで別の非終端記号に還元するといった関連である．彼らは，このような依存関係に基づいてルールをトポロジカルソートして階層化し，各階層の逆順序にルールを適用することにより，ルール適用を効率化している．

また，ルール適用におけるトークンの組合せの選び方によっても高速化が図れる．Chok らは，トークンの組合せすべてを用いて解析を行っていたのに対し，Balt はこれを改良し，制約条件の一部を利用して探索する組合せを減少させている [13]．また，我々は [14] で，制約条件間の関係に基づくグラフを利用して，トークンの組合せを求める手法についても提案している．

## 3 トークンに着目したルール適用

Chok らの手法は，ルール間の関係のみ着目しており，現在解析中のトークンがどのような状態であっても，順序どおりルールを適用する．このため，無用なルール適用を生じていた．そこで，我々はトークンの状態に着目したルール適用を提案する．

第 2 節で示した図形言語の場合，Node トークンが存在しない場合には，還元すべき RHS のトークンがないので，ルール 1 を適用しても意味がない．つまり，トークンの状況を把握し，解析すべきトークンに対して，必要なルールを適用すればよい．あるトークンに対して，適用すべきルールは，そのトークンが RHS に現れるようなすべてのルールである．

一度ルール適用が失敗したトークンについては，さらに同じルール適用しても無駄である．よって，このようなトークンを区別し，ルール適用を判断するためにトークンを管理する．まず，解析の最初で，すべてのトークンを「未処理トークン集合」に入れておき，ルールを適用した後にトークンを未処理トークン集合から取り除くようにする．ルール適用を行うトークンを未処理トークン集合から選ぶことにより，重複してルールを適用することがなくなる．なお，インタラクティブに追加された図形に対応するトークンや，ルール適用により還元された LHS の記号は，ルール適用を行っていないため，未処理トークン集合に追加される．適用すべきルールがなくなっ

た場合，すなわち未処理トークン集合が空になった場合に解析が終了する．

提案したルール適用と，未処理トークン集合の扱いを含めた解析アルゴリズムは次のようになる．

```
routine Parse()
  while U is not empty
    T := one of U
    changed := false
    for each P in DependRule(T)
      if EvaluateRule(P) then
        changed := true
        break
      endif
    end
    if !changed then
      U := U\HasSameNameOf(T, U)
    endif
  end
end

routine InsertNewToken(A, P)
  N := NewNonTerminal(A, P)
  D := (D\A)∪{N}
  U := (U\A)∪{N}
end
```

ここで，U と D は大域変数であり，それぞれ未処理トークン集合とトークンデータベースを表す．*Parse*() は解析のトップレベルであり，D に対してルールを適用し解析を行う．最初に U から 1 つのトークン T を選び，そのトークン T に対して，*DependRule*(T) で T の記号名が RHS に現れるようなルールの列を得てルールを適用する．このルールすべてに対してトークンの還元が行われなかった場合は，T は U から取り除く．ここでは，*HasSameNameOf*(T, U) を用いて，U に含まれる T と同じ記号名を持つすべてのトークンを取得し，U から削除している．これは，T の記号名にのみ着目して解析を行っており，T と同じ記号名を持つ他のトークンについてもルール適用を行っているため，これらトークンについてもルール適用が失敗しているからである．

*EvaluateRule*(P) は，トークンデータベース D に対してルール P を適用する．トークンの還元が行われる場合は，*EvaluateRule*() 内部で *InsertNewToken*(A, P) が呼ばれ，RHS のトークン列 A がルール P に基づいて還元され，*EvaluateRule*() は *true* を返す．*InsertNewToken*() は，LHS の記号を *NewNonTerminal*() で生成する．生成したトークン N は，D とともに U に追加する．また，RHS のトークン列 A を，D から取り除くとともに，U からも取り除く．

## 4 組合せ数の削減

*EvaluateRule*() におけるルール適用では，トークンデータベース中のトークンの組合せに対して，制約条件のテストが行われる．探索されるトークンの組合せは，トークンデータベースに含まれるトークンから生成されるすべての可能な組合せとなる．一般に，トークンデータベースのサイズを $N$ とし，ルールにおける RHS の記号数を $k$ とすると，トークンの組合せは $O(N^k)$ となる．解析が遅い一番の原因は，ここにある．

Balt が組合せを求める際に利用したのは，対象となるトークンが 1 つのトークンとなる制約条件である．これを単一トークンに関する制約と呼ぶことにする．第 2 節で示した例では `isText(t.text)` と `isOperand(t.text)` の 2 つが該当する．この制約条件を満たすようなトークンを前処理で求め，このようなトークンで構成される組合せのみ探索することにより，すべての組合せを探索することなく解析が実行できる．

ルールにおける制約条件は，単一トークンに関する制約以外にも様々なものが記述できる．我々が着目するのは，2 つのトークンに関する制約である．これは，対象となるトークンが 2 つのトークンになる制約条件である．また，単一トークンに関する制約とともに，非常に良く使われる制約条件である．2 つのトークンに関する制約の例としては，直線の始点が円の中心と一致している，2 つの矩形の横幅が等しいといったようなものがある．第 2 節で示した例では，単一トークンに関する制約を除いたすべての制約条件が該当する．

トークンの組合せを求める際に，単一トークンに関する制約を利用し，さらに，2 つのトークンに関する制約を利用することで，対象となるトークンを減らし，組合せを削減することができる．ルール適用の際は，まず，トークンデータベースから，RHS の記号に対応するトークンを取得し，初期状態のトークンとする．そして，単一トークンに関する制約を用いて，この制約条件を満たすことのできないトークンを取り除く．さらに，2 つのトークンに関する制約を満たすトークンの組を探索し，この組に現れないようなトークンを除外する．このような前処理を施すことにより，探索するトークンの組合せ数の削減を図ることができる．

探索する組合せ数を削減させたルール適用を行う *EvaluateRule*() は，次のようになる．

```
routine EvaluateRule(P)
  M := GetTokenList(D, P)
  C := constraint list in P
  for each L in SinglesOf(C)
    M := CheckConstraint(L, M)
  end
  for each L in DoublesOf(C)
    M := CheckConstraint(L, M)
  end
  for each A in Combination(M)
    if SatisfiesConstraints(C, A) then
      InsertNewToken(A, P)
      return true
    endif
  end
  return false
end
```

*GetTokenList*(D, P) は，トークンデータベース D に含まれるトークンのうち，ルール P における RHS の各記号に合致するトークンを，各記号別にリストにしたものとして取得する．これが，初期状態のトークンとなり M に設定される．C にはルール P における制約条件が設定され，*SinglesOf*(C) と *DoublesOf*(C) により，C における単一トークンに関する制約と，2 つのトークンに関する制約を得る．*CheckConstraint*(L, M) では，制約条件 L を満たすトークンのみを M から求め，制約条件を満たさないトークンを取り除いたものを返す．最初の for ループにおいて，単一トークンに関する制約を用いたトークンの限定を行い，M を更新する．その後に，同様にして 2 つのトークンに関する制約を利用してトークンを限定する．これら前処理を行った後，求められたトークンの列で構成されるすべての組合せを *Combination*() で求め，これら 1 つ 1 つに対して，*SatisfiesConstraints*() を用いて制約条件が成り立つか探索している．

## 5 実験

提案手法の効果を確認するため，Chok らの手法と Balt の手法との比較実験を行った．第 2 節に示した図形言語を用い，トークン数が 50 となるような図形に対し，トークンをランダムな順序で 1 つずつ解析器に与え，ルール適用の数，探索した組合せの数，すべてを解析するのに要した時間（秒）について 20 回の平均値を計測した．表 1 に実験結果を示す．ルール適用数に関しては，Chok らの手法や，同様の手法を用いている Balt と比べ約 20 %削減され，一定の効果が得られていることがわかる．また，組合せの数では，Chok らの手法に比べ 94 分の 1，Balt の手法に比べても 78 分の 1 となり，大きく削減されている．解析時間においては，Balt や Chok らの手法では 10 秒程度かかっているが，提案手法で解析した場合，0.15 秒と高速に解析できている．これにより，特に組合せ数の削減の効果により，解析の高速化が実現されていることがわかる．

表 1: 実験結果

| | ルール適用 | 組合せ | 解析時間 |
|---|---|---|---|
| Chok | 120.0 | 2368717.1 | 11.822 |
| Balt | 120.0 | 1955030.3 | 9.647 |
| 提案手法 | 95.8 | 25092.2 | 0.153 |

## 6 議論

提案手法では，組合せ数を削減させるため，トークンを限定させる前処理を行っている．単一トークンに関する制約をチェックするには，該当するすべてのトークンを調べるため，O($N$) の計算量が必要になる．また，2 つのトークンに関する制約をチェックするには，トークンの組をすべて調べるため，O($N^2$) の計算量が必要となる．しかし，これら前処理は，最終的に探索する O($N^k$) の組合せに比べて少ないため問題とはならない．実験の結果からもわかるように，前処理を行うことで組合せ数が効率的に削減できており，総合的に高速化に寄与している．

我々が [14] で提案している手法は，本論文で提案した手法や，Chok や Balt の手法とも異なったアプローチをとっている．制約条件と RHS の記号の関係に着目した,「制約条件グラフ」を利用し解析を行っている．トークンの組合せは，グラフの訪問に対応させて探索している．本論文で提案した手法は，前処理でトークンの組合せを削減してから組合せを探索するといった違いがある．ルール適用では，ほぼ同様の処理を用いているが，ルール適用に失敗した際の処理が一部異なっている．

## 7 まとめと今後の課題

空間解析器における処理を高速化するための 2 つの手法について提案を行った．トークンの状態に着目し，未処理トークン集合を用いてトークンを管理することにより効率的なルール適用を実現した．また，2 つのトークンに関する制約を利用して前処理を行うことにより，探索される組合せ数の削減させ

た．これら手法による高速化の効果を確かめるため実験を行い，Chok らや Balt の解析手法に比べて高速な解析が実現できることを示した．

今後の課題としては，解析手法の更なる改善と，実験の追加が挙げられる．トークンに着目したルール適用については，さらに効率化できる可能性があると考えている．また，他の例題を用いた実験をしたいと考えている．さらに，提案した手法を恵比寿などのシステムに組み込み，テストを行う予定である．

## 参考文献


[1] Costagliola, G., Lucia, A. D., Orefice, S., and Tortora, G.: Positional Grammars: a Formalism for LR-like Parsing of Visual Languages, *Visual Languages Theory*(Marriott, K. and Meyer, B.(eds.)), Springer, 1998, pp. 171–191.

[2] Ferrucci, F., Tortora, G., Tucci, M., and Vitiello, G.: A Predictive Parser for Visual Languages Specified by Relation Grammars, *Proceedings of IEEE Symposium on Visual Languages*, 1994, pp. 245–252.

[3] Helm, R., Marriott, K., and Odersky, M.: Building Visual Language Parsers, *Conference proceedings on Human Factors in Computing Systems*, 1991, pp. 105–112.

[4] Marriott, K.: Constraint Multiset Grammars, *Proceedings of IEEE Symposium on Visual Languages*, 1994, pp. 118–125.

[5] Costagliola, G., Tortora, G., Orefice, S., and Lucia, A. D.: Automatic Generation of Visual Programming Environments, *IEEE Computer*, Vol. 28,No. 3(1995), pp. 56–66.

[6] Golin, E. J. and Maglierry, T.: A Compiler Generator for Visual Languages, *Proceedings of IEEE Symposium on Visual Languages*, 1993, pp. 314–321.

[7] Chok, S. S. and Marriott, K.: Automatic Construction of Intelligent Diagram Editors, *Proceedings of the ACM Symposium on User Interface Software and Technology*, 1998, pp. 185–194.

[8] 馬場昭宏, 田中二郎: Spatial Parser Generator を持ったビジュアルシステム，情報処理学会論文誌, Vol. 39,No. 5(1998), pp. 1385–1394.

[9] 飯塚和久, 志築文太郎, 田中二郎: 図形言語処理システムにおける図形エディタと空間解析器 の統合, 日本ソフトウェア科学会第 18 回大会, 2001.

[10] Iizuka, K., Tanaka, J., and Shizuki, B.: Describing a Drawing Editor by Using Constraint Multiset Grammars, *Proceedings of the International Symposium on Future Software Technology*, 2001, pp. 119–124.

[11] 山田英仁, 飯塚和久, 田中二郎: ビジュアルシステム生成系「恵比寿」におけるジェスチャ の実現, 日本ソフトウェア科学会第 19 回大会, 2002.

[12] Chok, S. S. and Marriott, K.: Parsing Visual Languages, *Proceedings of the 18th Australasian Computer Science Conference*, 1995, pp. 90–98.

[13] Balt, L. R.: Full CMG parsing, Master's thesis, Leiden University, The Netherlands, 1996.

[14] 飯塚和久, 亀山裕亮, 志築文太郎, 田中二郎: インクリメンタルな解析による空間解析器の高速化, 情報処理学会論文誌：プログラミング. 掲載予定.