Document Structure Analysis with Syntactic Model and Parsers: Application to Legal Judgments

Hirokazu Igari, Akira Shimazu, and Koichiro Ochimizu

School of Information Science
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa, 923-1292, Japan
{higari,shimazu,ochimizu}@jaist.ac.jp
http://www.jaist.ac.jp/

Abstract. The structure of a type of documents described in a common format like legal judgments can be expressed by and extracted by using syntax rules. In this paper, we propose a novel method for *document structure analysis*, based on a method to describe syntactic structure of documents with an *abstract document model*, and a method to implement a document structure parser by a combination of *syntactic parsers*. The parser implemented with this method has high generality and extensibility, thus it works well for a variety of document types with common description format, especially for legal documents such as judgments and legislations, while achieving high accuracy.

Keywords: document structure analysis, document model, document structure parser.

1 Introduction

The advances of information processing technologies and the high speed network in recent years have made a huge number of electronic documents, which are available for retrieval. To help readers quickly find documents and information in them depending upon purposes and interests, new technology is awaited, which automates extraction of relevant information from documents.

As an approach for realizing such technology, we are proceeding with research on methods for extracting structure and relevant information from documents. According to our empirical observations, for a type of documents described in a common format, the structures often have high similarity to each other, and the same type of information is often included in similar parts. For example, *legal judgments* (or simply called as *judgments*), written by the courts for describing the decision based on the facts, have almost the same structure if the court and the case type are the same. In addition, relevant information in them, such as claims by the parties or decision by the court, appears in similar parts in almost the same order, and similar section titles are assigned to them. With these facts,

M. Okumura, D. Bekki, and K. Satoh (Eds.): JSAI-isAI 2011, LNAI 7258, pp. 126–140, 2012.

it is expected that, by using structure information, a new method for extracting relevant information from documents with high accuracy can be realized.

As the first step, we are researching methods for *document structure analysis*, which is a task for extracting structure information from documents. In this paper, we propose a novel method for document structure analysis. While most of existing document structure analysis methods make use of layout information for each document page [6,7,9,12,14,16], our method instead uses syntactic information for the entire document text. This method performs very well for types of documents with a common description format, and for legal documents such as judgments and legislations in particular. Until now, we have designed and implemented a *document structure parser* for legal judgments made by a Japanese court[1].

To make the document structure parser general and extensible to support a variety of document types, we have invented two novel and effective methods; one is an abstract document model called *block structure model,* and the other is a method to implement syntactic parsers for block structure called *block breaker*s. These methods work very well for increasing generality and extensibility of the parser, as well as for achieving high accuracy of document structure extraction. Fig. 1 illustrates an overview of the document structure parser implemented with these methods. A judgment text is broken by block breakers into hierarchical text segments according to the block structure denition, then the leaf text segments are parsed by syntactic parsers to extract the document structure.

In the remaining of this paper, related works on document structure analysis are discussed in Section 2, document structure expressions of judgments with syntax rules and document structure model are explained in Section 3, and implementation of document structure parser with syntactic parsers and block breakers are explained in Section 4, followed by performance evaluation and conclusion in Section 5 and 6.

2 Related Works on Document Structure Analysis

Methods for document structure analysis have been widely researched since the 1990s [9], when many documents started to be transformed to electronic and structured formats such as SGML and XML, so that the contents can be processed by computers. The information on the structure of a document includes; *physical structure* which consists of visual elements in each document page and their layout information; *logical structure* which consists of logical elements in the entire document text and their order and inclusive relation; *semantic structure* which consists of topics in the document and their semantic relation[2].

The main target of document structure analysis has been logical structure [6,7,12,14,16], which consists of information such as metadata and section structure relevant for structuring and transforming documents to electronic format.

[1] Important judgments made by Japanese courts are available as PDF files with text from the Courts in Japan web site (`http://www.courts.go.jp/`).

[2] Names and definitions for document structure information categories vary between researches. These categories are mentioned in researches such as [3] and [9].

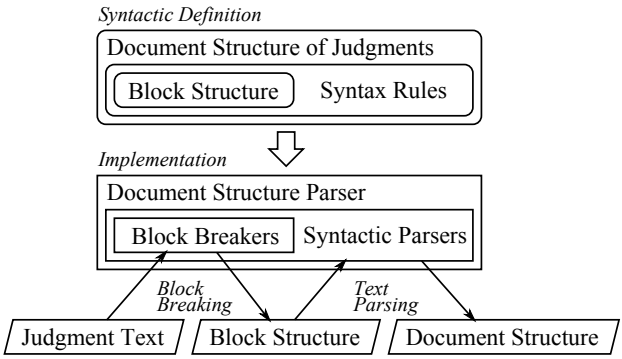

Fig. 1. Overview of Document Structure Parser

For most of the logical document structure analysis methods, physical structure is extracted first from document data such as OCR result or PDF file, and then logical structure is extracted by assigning logical roles to the physical elements and by analyzing their relations with layout information.

However, when documents are described in a common format, it is possible to express the logical structure by syntax rules for the document text. By taking advantage of this characteristic of such document types, the method we propose uses physical structure only for obtaining the document text, and extracts logical structure according to the syntactic document structure definition.

There have also been researches on document structure analysis for legal documents [1,10]. These methods are based on text grammar, similarly to our method, but the implemented parsers are specific to the target document types. Our method is more general and extensible based on an abstract document model and syntactic parsers, thus it works for a variety of document types.

Regarding semantic document structure, probabilistic methods for extracting topics in documents and analyzing their relation, such as *topic models* [3,8,15] and *content models* [2], have been proposed since the early 2000s. Most of these methods extract only document level topics, and few method considers logical document structure for extracting topics and their relation.

In this paper, we focus on logical document structure, and unless specifically mentioned, the term "document structure" refers logical document structure.

3 Document Structure of Legal Judgments

For legal judgments written by the same court for the same type of cases, their description formats have high commonality. For example, judgments made by the Japanese Intellectual Property High Court for patent cases are described in a format as Fig. 2. A judgment starts with case metadata consisting of judgment date, case number and name, and last debate date, followed by judgment type and parties. Sentence, fact and reason are described in a hierarchical section structure, and then court and judges are listed with any supporting papers.

(Judgment Date)	平成２１年１月２８日　判決言渡
(Case Numebr and Name)	平成１９年(行ケ)第１０２８９号　審決取消請求事件
(Last Debate Date)	平成２１年１月２８日　口頭弁論終結
(Judgment Type)	判　決
(Parties)	原告　レキシンジャパン株式会社
	被告　シコー株式会社
(Sentence)	主　文
(Section)	１　原告の請求を棄却する。
(Section)	２　訴訟費用は原告の負担とする。
(Fact and Reason)	事実及び理由
(Section)	第１　請求
	特許庁が…した審決を取り消す。
(Section)	第２　事案の概要及び判断
(Section)	１　被告らは，…。
	原告は，…。
(Section)	２　当裁判所の判断
	…よって，主文のとおり判決する。
(Court)	知的財産高等裁判所第３部
(Judges)	裁判長裁判官　飯村　敏明
	裁判官　　中平　健

Fig. 2. Description Format of Legal Judgments by a Japanese Court

In addition, each element has unique prefix text pattern as Table 1, described by PEG parsing expressions which will be explained in the next section, thus the boundaries of elements can be recognized by text pattern matching. We take advantage of these characteristics of legal judgments for the document structure analysis method proposed in this paper.

3.1 Syntactic Document Structure Expressions

When a type of documents has a common description format, structure of documents can be formally expressed by syntax rules. For text parsing, syntax rules can be described by PEG (Parsing Expression Grammar) [4]. PEG is a derivative of CFG (context-free grammar), which is described by notations such as EBNF (Extended Backus-Naur Form), but PEG is different from CFG in that it is designed for text recognition and does not describe ambiguous syntax. Also, PEG is a formal description of *recursive descent parser* which performs top down syntax parsing. A PEG grammar G is defined as $G = (V_N, V_T, R, e_S)$, where:

- V_N is a finite set of *nonterminal symbols*
- V_T is a finite set of *terminal symbols*
- R is a finite set of *parsing rules*
- e_S is a *parsing expression* termed the start expression
- $V_N \cap V_T = \emptyset$
- Each parsing rule $r \in R$ is written as $A \leftarrow e$, where $A \in V_N$ and e is a parsing expression.
- For any nonterminal symbol A, there is exactly one parsing expression e for the parsing rule $A \leftarrow e \in R$.

Table 1. Prefix Text Patterns of Legal Judgments by a Japanese Court

Element	Prefix Text Pattern
Judgment Date	`Date "判決言渡"`
Case Number and Name	`CaseNumber CaseName`
Last Debate Date	`Date "口頭弁論終結"`
Judgment Type	`"判決"`
Parties	`(PartyTitle PartyName) | PartyAddress`
Sentence	`"主文"`
Fact and Reason	`"事実及び理由"`
Court	`CourtName CourtDivision`
Judges	`"裁判長"? "裁判官" JudgeName`

Terminal symbols are defined by the following expressions.

- '*abc*' or "*abc*", string literal
- [*abc*] or [*a* − *c*], character class
- ., any single character

Parsing expressions are defined as follows, where e, e_1 and e_2 are parsing expressions.

- ε, the empty string
- a, any terminal, where $a \in V_T$
- A, any nonterminal, where $A \in V_N$
- $e_1\, e_2$, a sequence
- $e_1 \,|\, e_2$, a prioritized choice
- $e*$, zero-or-more repetitions
- $e+$, one-or-more repetitions
- $e?$, an option
- $\&e$, an and-predicate - succeeds if e succeeds without consuming input
- $!e$, a not-predicate - succeeds if e fails without consuming input

A syntax rule for the top level document structure of the judgment presented in Fig. 2 can be described by PEG as follows, where nonterminals such as `JudgmentDate` and `CaseNumberNames` are defined separately.

```
Judgment <-
  (JudgmentDate | CaseNumberNames | LastDebateDate)+
  JudgmentType
  Parties
  Sentence
  ((Fact Reason) | FactReason)?
  CourtJudges
  AttachedPaper?
```

For example, syntax rules for `CaseNumberNames` can be described as follows.

```
CaseNumberNames <- CaseNumberName+
CaseNumberName  <-
  (CaseAbbName | CaseNumber | CaseName)+ |
  (LPar (CaseAbbNames | OriginalSentences) RPar)
```

At the bottom level of the document structure, syntax rules for `CaseNumber` can be described as follows, where nonterminals such as `CourtName` and `Year` are defined separately.

```
CaseNumber   <- CourtName? (Year Symbol? | '同')? SeqNumber
SeqNumber    <- '第' HyphenNumber '号'? | HyphenNumber '号'
HyphenNumber <- Number (Hyphen Number)*
Symbol       <- SymbolPrefix? LPar SymbolChar+ RPar
SymbolChar   <- SymbolPrefix | SymbolType | Kana
SymbolPrefix <- [刑合特]
SymbolType   <- [受許行家医収少人秩手日分甲]
```

By a combination of these syntax rules, the structure of the entire document can be defined. Note that document structure can be dened arbitrarily for its purpose. That is, multiple document structures can be dened for a single document type depending on the information to be extracted. For example, to extract only metadata, dening the document body as a single element is sucient.

3.2 Block Structure Model

Although it is possible to implement a document structure parser according to syntax rules for the actual document elements, such implementation works only for a specific document type. To make the document structure parser implementation general and extensible, we have invented a novel method to describe document structure with an abstract document model, called *block structure model*, which expresses document structure with abstract document elements, called *blocks* and *nodes*. Blocks are hierarchical text segments corresponding to the actual document elements, and nodes are leaf text segments to be parsed for extracting information. There are two types of blocks, *composite block* and *node block*. A composite block contains a sequence of child blocks, and a node block contains a node. The boundaries of blocks are recognized by the prefix text pattern of nodes. To express block structure by syntax rules, the following nonterminal symbols are defined for block structure elements.

- `Block`, a text segment for a document element
- `CompositeBlock`, a block which contains a sequence of child blocks
- `NodeBlock`, a block which contains a node
- `Node`, a leaf text segment to be parsed

Since many documents including judgments have hierarchical section structure, extended block structure elements to describe the section structure are also defined.

- `SectionBlock`, a composite block for a section
- `SectionNodeBlock`, a node block for a section node
- `SectionNode`, a node which consists of section prefix and text to be parsed

The block structure model is expressed by syntax rules as follows.

```
Document         <- CompositeBlock
CompositBlock    <- Block+
Block            <- SectionBlock | CompositeBlock | NodeBlock
NodeBlock        <- Node
Node             <- &NodePrefix NodeText
SectionBlock     <- SectionNodeBlock SectionBlock*
SectionNodeBlock <- SectionNode
SectionNode      <- &SectionPrefix SectionText
SectionPrefix    <- (SenctionNumber+ SectionTitle?) | SectionTitle
```

To describe document elements with abstract block structure elements, actual element names such as `Judgment` and `JudgmentDate` are appended with parentheses as the following example.

```
Document(Judgment)
CompositeBlock(Judgment)
NodeBlock(JudgmentDate)
SectionBlock(Sentence)
```

With the block structure model, the syntax rules for the top level document structure of the judgment presented in Fig. 2 can be rewritten as follows, and the judgment text is broken into blocks and nodes as Fig. 3.

```
Document(Judgment) <- CompositeBlock(Judgment)
CompositeBlock(Judgment) <-
  (NodeBlock(JudgmentDate) | NodeBlock(CaseNumberNames) |
   NodeBlock(LastDebateDate))+
  NodeBlock(JudgmentType)
  NodeBlock(Parties)
  SectionBlock(Sentence)
  (SectionBlock(Fact) SectionBlock(Reason)) |
   SectionBlock(FactReason))?
  NodeBlock(CourtJudges)
  NodeBlock(AttachedPaper)?
```

4 Document Structure Parser

A document structure parser extracts document structure from a given text of a document according to the syntactic document structure definition described as syntax rules with the block structure model. It consists of *syntactic parsers* implemented according to the syntax rules and extracts document structure by the following two steps of tasks.

- *Block Breaking* - Break a document text into blocks and nodes with *block breakers*, which are syntactic parsers for block structure.

Fig. 3. Block Structure of Legal Judgments by a Japanese Court

– *Text Parsing* - Parse nodes in the extracted block structure with syntactic parsers to extract information in them.

The extracted document structure is available as a block structure containing the text parsing results at its nodes. It can be traversed and transformed to any output format such as XML.

4.1 Syntactic Parsers

A parser which parses a given text according to syntax rules described by PEG can be implemented while preserving the structure of the syntax rules, with *parser combinators* [5,11]. We call such parsers as *syntactic parsers*. Parser combinators is a method to implement *recursive descent parsers*, and an implementation is provided as a standard library in *Scala* program language [13]. The structure of parser code written with parser combinators exactly matches the structure of syntax rules, thus implementing and understanding parsers is straightforward and easy. In addition, new parsers can be implemented by combinations of existing parsers, thus parsers can be easily and flexibly reused. Especially, parsers implemented with the Scala parser combinators have operators corresponding to PEG parsing expressions, thus any syntax rules can be directly implemented while preserving the structure as it is. To take this advantage, we developed the document structure parser with Scala. With the Scala parser combinators, syntactic parsers for `CaseNumberNames` presented above can be implemented as follows.

```
def caseNumberNames = (caseNumberName)+
def caseNumberName  =
  ((caseAbbName | caseNumber | caseName)+) |
  (lPar ~ (caseAbbNames | originalSentences) ~ rPar)
```

Similarly, syntactic parsers for `CaseNumber` can be implemented as follows.

```
def caseNumber =
  (courtName?) ~ (((year ~ (symbol?)) | '同')?) ~ seqNumber)


def seqNumber =
  ('第' ~ hyphenNumber ~ (('同')?)) | (hyphenNumber ~ '号')
def hyphenNumber = number ~ ((hyphen ~ number)*)
def symbol = (symbolPrefix?) ~ lPar ~ (symbolChar+) ~ rPar
def symbolChar = symbolType | symbolPrefix | kana
def symbolPrefix = among("刑合特")
def symbolType = among("受許行家医収少人秩手日分甲")
```

4.2 Block Breakers

Since the end of a block is determined by the prefix of the subsequent block, it is not possible to implement a parser for block breaking by a combination of independent syntactic parsers. To resolve this issue, we have invented a novel method to implement syntactic parsers for block breaking, which extract block structure while recognizing the boundaries of blocks, called *block breakers*. A block breaker generates two syntactic parsers, called *first* and *blocks,* which are used for implementing a new block breaker by a combination of existing block breakers. The notations `first(next)` and `blocks(next)` denote syntactic parsers, which are dynamically generated depending on the parameter `next`.

- `BlockBreaker`, a block breaker
  - `first(next)`, a syntactic parser which recognizes the prefix of the first existing node depending on `next`
  - `blocks(next)`, a syntactic parser which extracts the target block structure depending on `next`
  - `next`, one of the following parsers is used
    - `first` of a subsequent block breaker and used when there is a subsequent block breaker
    - `EOF`, a parser which recognizes the end of input and used when there is no subsequent block breaker
    - `NEVER`, a parser always fails and used when there is no need to consider subsequent block breaker
  - `blocks(EOF)`, a standalone syntactic parser which extracts the target block structure

The prefix of the first node in the target block is recognized by `first`, but the target block can be optional, thus the first existing node may be contained in

a subsequent block. To handle such cases, `next` is used to recognize the first existing node. As `next`, `first` of the subsequent block breaker is used when there is a subsequent block breaker. Otherwise, `EOF` is used when there is no subsequent block, or `NEVER` is used when there is no need to consider subsequent blocks. As a special case, `block(EOF)` generates a standalone syntactic parser which extracts the target block structure from a given text.

To describe the structure of block breakers by syntax rules, block breakers for block types and the syntax rules for `first` and `blocks` are defined as follows. Nodes are recognized and parsed in terms of *node types* associated with two parsers, one recognizes node prex and the other parses node text respectively. In addition, since sibling sections have sequential section numbers with the same indent, section nodes are recognized in terms of *section context* containing node type and other context information. The notations such as `b.first(next)`, `nodeType.prefix` and `context.nodeType` denote members of block breaker, node type and section context respectively.

- `CompositeBlockBreaker(b)`, extracts a composite block where `b` is a block breaker for child blocks
  - `first(next)  <- b.first(next)`
  - `blocks(next) <- b.blocks(next)`
- `NodeBlockBreaker(nodeType)`, extracts a node block where `nodeType` is the target node type
  - `first(next)  <- nodeType.prefix`
  - `blocks(next) <- nodeType.prefix (!next textLine)*`
- `SectionBlockBreaker(context)`, extract a section block where `context` is the section context
  - `SectionBlockBreaker(context) <- SectionNodeBlockBreaker(context) SectionBlockBreaker(context)*`
- `SectionNodeBlockBreaker(context)`, extract a section node block where `context` is the section context
  - `SectionNodeBlockBreaker(context) <- NodeBlockBreaker(context.nodeType)`

For any block breaker `b`, `b1` and `b2`, a new block breaker can be created by using the same syntax expression as the target block structure.

- `b1 b2`, a sequence
  - `first(next)  <- b1.first(b2.first(next))`
  - `blocks(next) <- b1.blocks(b2.first(next)) b2.blocks(next)`
- `b1 | b2`, a prioritized choice
  - `first(next)  <- b1.first(NEVER) | b2.first(next)`
  - `blocks(next) <- b1.blocks(next) | b2.blocks(next)`

- `b*`, zero-or-more repetitions
  - `first(next)  <- b.first(NEVER) | next`
  - `blocks(next) <- b.blocks(first(next))*`
- `b+`, one-or-more repetitions
  - `b+ <- b b*`
- `b?`, an option
  - `first(next)  <- b.first(NEVER) | next`
  - `blocks(next) <- b.blocks(next)?`

Fig. 4 shows how a new block breaker is implemented by a combination of existing block breakers. In this example, a sequence of block breakers `b1` and `b2`, described as `b1 b2`, is implemented by using `first` and `blocks` of `b1` and `b2`. The parser `next` used by `b1 b2` is determined depending on whether it has subsequent block breaker. If it has subsequent block breaker `b3`, `first` of `b3` is used as `next`. Otherwise, `EOF` or `NEVER` is used depending on the context in which `b1 b2` is used in the syntax rules.

Structure of block breakers can be expressed by the same syntax rules as the target block structure as follows.

```
DocumentBreaker      <- CompositeBlockBreaker
CompositBlockBreaker <- BlockBreaker+
BlockBreaker         <-
  SectionBlockBreaker | CompositeBlockBreaker | NodeBlockBreaker
SectionBlockBreaker  <- SectionNodeBlockBreaker SectionBlockBreaker*
```

As a result, the top level block breaker structure of the judgment presented in Fig. 2 can be expressed by syntax rules as follows.

```
DocumentBreaker(Judgment) <- CompositeBlockBreaker(Judgment)
CompositeBlockBreaker(Judgment) <-
  (NodeBlockBreaker(JudgmentDate) | NodeBlockBreaker(CaseNumberNames) |
   NodeBlockBreaker(LastDebateDate))+
  NodeBlockBreaker(judgmentType)
  NodeBlockBreaker(Parties)
  SectionBlockBreaker(Sentence)
  (SectionBlockBreaker(Fact) SectionBlockBreaker(Reason)) |
   SectionBlockBreaker(FactReason))?
  NodeBlockBreaker(CourtJudges)
  NodeBlockBreaker(AttachedPaper)?
```

Block breakers for these syntax rules can be implemented by a combination of block breakers as it is, where `compositeBlockBreaker`, `sectionBlockBreaker` and `nodeBlockBreaker` denote methods to create the corresponding types of block breakers.

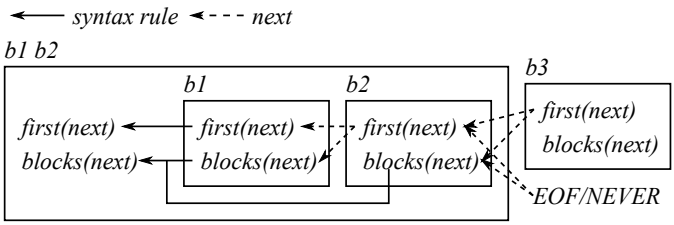

Fig. 4. Implementation of a Sequence of Block Breakers

```
def judgmentDocumentBreaker = judgmentCompositeBlockBreaker
def judgmentCompositeBlockBreaker = compositeBlockBreaker(
  ((nodeBlockBreaker(JudgmentDate) | nodeBlockBreaker(CaseNumberNames) |
    nodeBlockBreaker(LastDebateDate)+) ~
  nodeBlockBreaker(JudgmentType) ~
  nodeBlockBreaker(Parties) ~
  nodeBlockBreaker(Sentence) ~
  (((sectionBlockBreaker(Fact) ~ sectionBlockBreaker(Reason)) |
    sectionBlockBreaker(FactReason))?) ~
  nodeBlockBreaker(CourtJudges) ~
  (nodeBlockBreaker(AttachedPaper)?)
```

With these block breakers and the syntactic parsers for the nodes, the document structure parser extracts the document structure from a given text of a judgment.

5 Performance Evaluation

To evaluate the document structure parser implemented with the proposed method, performances are calculated for the current version of the parser and the baseline system. As a baseline system, a document structure parser based on non-syntactic method is used, which is actually used for creating legal case data for a Japanese legal information service. It extracts block structure by a program logic specific to judgments according to predefined rules. For 245 judgments made by the Japanese Intellectual Property Court for patent cases in 2009, extracted block structures are compared in terms of the node paths with the manually extracted block structures consisting of 25,213 nodes. The result is summarized in Table 2, which consists of three subtables.

The numbers of extracted nodes, their successes and errors, followed by precision, recall and f-score are listed in the first subtable. Errors are categorized in terms of whether nodes are extracted by only the parser (Type I) or by only manually (Type II). The current parser recognizes 24,922 nodes with 1,403 Type I and 1,694 Type II errors, and precision, recall and f-score are 0.944, 0.933 and 0.938. On the other hand, the baseline system recognizes 27,899 nodes with 7,368 Type I and 4,682 Type II errors, and precision, recall and f-score are 0.736, 0.814 and 0.773. This result shows significant improvement by the current method.

Table 2. Performance Evaluation of Document Structure Parser

Performance: Judgments=245 Manual Nodes=25,213

	nodes	success	TypeI err	TypeII err	precision	recall	f-score
Current	24,922	23,519	1,403	1,694	0.944	0.933	0.938
Baseline	27,899	20,531	7,368	4,682	0.736	0.814	0.773

F-Score Distribution: Judgments=245

	judgments		percentage		accumulated		percentage	
f-score	curr	base	curr	base	curr	base	curr	base
1.0	70	19	28.6%	7.8%	70	19	28.6%	7.8%
0.9 - 1.0	127	86	51.8%	35.1%	197	105	80.4%	42.9%
0.8 - 0.9	28	43	11.4%	17.6%	225	148	91.8%	60.4%
0.7 - 0.8	9	29	3.7%	11.8%	234	177	95.5%	72.2%
0.6 - 0.7	9	27	3.7%	11.0%	243	204	99.2%	83.3%
0.5 - 0.6	2	27	0.8%	11.0%	245	231	100.0%	94.3%
0.2 - 0.5	0	14	0.0%	5.7%	245	245	100.0%	100.0%

Error Detail: Judgments=10 Manual/Parser Nodes=1,736/1,599

	Type I				Type II			
Type	number	title	prev.err	total	number	indent	prev.err	total
Errors	18	11	131	157	23	24	244	294
Ratio	11.5%	7.0%	83.4%	100%	7.8%	8.2%	83.0%	100%

The numbers and percentages of judgments for f-score ranges, and their accumulated numbers and percentages are listed in the second subtable. The current parser recognizes nodes perfectly for 70 judgments (28.6%), with greater than 0.9 f-score for 197 judgments (80.4%), and with greater than 0.8 f-score for 225 judgments (91.8%). Although the structures and the node prefix patterns of the judgments should have only a small variance, since they are all made for patent cases within a year, this result still can be considered as fairly good, taking account of the size of judgments consisting of 102 nodes in average with hierarchical section structure. However, some judgments have a low f-score. According to the analysis, most of these judgments have quoted text of other judgments, which are not considered by the current document structure parser.

Error types, their numbers and ratios for 10 sampled judgments by the current parser are listed in the third subtable. For 1,736 manually extracted nodes, 1,599 nodes are extracted by the parser, and all errors are for section prefix recognition. Among 157 Type I errors, 18 (11.5%) section numbers and 11 (7.0%) titles which are not actually section prefixes are misrecognized. Among 294 Type II errors, 23 (7.8%) are by unexpected section number patterns, and 24 (8.2%) are by illegal indent greater than the predefined threshold. Other errors, 131 (83.4%) Type I and 244 (83.0%) Type II errors, are caused by previous prefix recognition errors. According to this result, it is expected that the parser performance can be further improved by adding more section prefix patterns and introducing new methods to improve section prex recognition.

6 Conclusion

In this paper, we proposed a method for document structure analysis, which is based on a method to describe syntactic document structure with an abstract model, and a method to implement a document structure parser by a combination of syntactic parsers. The parser implemented with this method extracts the document structure with high accuracy from documents described in a common format and the structure can be expressed by syntax rules. In addition, the parser has high generality and extensibility, thus works well for a variety of document types, especially for legal documents such as judgments and legislations.

However, there are still some elements which can not be recognized correctly by the current method. For example, it is difficult to determine by the deterministic syntactic rules if the text in the first line of a section is title or part of a paragraph. In the current document structure parser implementation, this is determined by a program logic considering features in the line such as suffix pattern and length; however, many titles are still not recognized correctly.

To realize a new method for extracting relevant information from documents using structure information, it is essential that correct document structure is provided. Thus, to further improve the accuracy of document structure extraction, we are working to enhance the current method to support ambiguous syntax rules and to make use of machine learning methods. By selecting the most probable result among the alternatives returned for ambiguous syntax rules with machine learning methods, many elements not recognized by the current method are expected to be recognized more correctly.

References

1. Bacci, L., Spinosa, P., Marchetti, C., Battistoni, R., Senate, I.: Automatic mark-up of legislative documents and its application to parallel text generation. IDT, 45 (2009)
2. Barzilay, R., Lee, L.: Catching the drift: Probabilistic content models, with applications to generation and summarization. In: Proceedings of HLT-NAACL, vol. 2004 (2004)
3. Blei, D.M., Lafferty, J.D.: Topic models. Text Mining: Classification, Clustering, and Applications 10, 71 (2009)
4. Ford, B.: Parsing expression grammars: a recognition-based syntactic foundation. ACM SIGPLAN Notices 39, 111–122 (2004)
5. Hutton, G., Meijer, E.: Monadic parsing in haskell. Journal of Functional Programming 8(4), 437–444 (1998)
6. Klink, S., Dengel, A., Kieninger, T.: Document structure analysis based on layout and textual features. In: Proc. of International Workshop on Document Analysis Systems, DAS 2000, pp. 99–111. Citeseer (2000)
7. Lee, K.H., Choy, Y.C., Cho, S.B.: Logical structure analysis and generation for structured documents: a syntactic approach. IEEE Transactions on Knowledge and Data Engineering, 1277–1294 (2003)
8. Li, W., McCallum, A.: Pachinko allocation: Scalable mixture models of topic correlations. J. of Machine Learning Research (2008) (submitted)

9. Mao, S., Rosenfeld, A., Kanungo, T.: Document structure analysis algorithms: a literature survey. In: Proc. SPIE Electronic Imaging, vol. 5010, pp. 197–207. Citeseer (2003)
10. Moens, M.F., Uyttendaele, C.: Automatic text structuring and categorization as a first step in summarizing legal cases. Information Processing & Management 33(6), 727–737 (1997)
11. Moors, A., Piessens, F., Odersky, M.: Parser combinators in scala. CW Reports, vol. CW491. Department of Computer Science, KU Leuven (2008)
12. Namboodiri, A., Jain, A.: Document structure and layout analysis. In: Digital Document Processing, pp. 29–48 (2007)
13. Odersky, M., Altherr, P., Cremet, V., Emir, B., Maneth, S., Micheloud, S., Mihaylov, N., Schinz, M., Stenman, E., Zenger, M.: An overview of the scala programming language. Technical report. Citeseer (2004)
14. Rangoni, Y., Belaïd, A.: Document Logical Structure Analysis Based on Perceptive Cycles. In: Bunke, H., Spitz, A.L. (eds.) DAS 2006. LNCS, vol. 3872, pp. 117–128. Springer, Heidelberg (2006)
15. Steyvers, M., Griffiths, T.: Probabilistic topic models. In: Handbook of Latent Semantic Analysis, vol. 427(7), pp. 424–440 (2007)
16. Summers, K.: Automatic discovery of logical document structure (1998)