Simple PDF

Martín Goñi

2/2/2026

¿Que es **SPF**?

Simple PDF, es un DSL para generar PDFs, es decir un lenguaje que al ser interpretado produce PDFs. Es más este informe esta hecho con **SPF**.

Consideraciones de diseño

A la hora de diseñar **SPF** me inspire en **LaTeX**, pero sin mantenerme atado a sus convenciones o decisiones de diseño, lo trate como una fuente de inspiracion. En particular tome otra direccion en cuanto a la forma de los comandos, permitiendo solo un argumento; Al mismo tiempo las opciones deben tener asociada una clave indicando a que corresponde el valor. Finalmente simplifique la forma en la que estructura el documento, ya no es necesario separar el titulo y documento de forma explicita.

Los comandos implementados son un subconjunto de los disponibles en **LaTeX**, elegidos para permitir generar documentos (mayormente)completos, manteniendo al mismo tiempo la cantidad total de comandos baja.

Sintaxis

Para representar la sintaxis se usan ciertas convenciones; El simbolo ***** indica cero o mas ocurrencias de lo que encierre mientras que el **+** indica una o mas ocurrencias. Los comentarios no estan incluidos pues no son parte de la sintaxis en si, son añadidos en el parser para la conveniencia del usuario. A continuación se presenta la sintaxis concreta de **SPF**:

```
 digit           ::= '0' | '1' | ... | '9'
 integer         ::= <digit> (<digit>)*
 float           ::= <integer> '.' <integer>
 letter          ::= 'a' | ... | 'z' | 'A' | ... | 'Z'
 letterStr       ::= <letter> (<letter>)*
 specialChars    ::= '\' | '{', '}' | '[' | ']' | '"' | '/' | '|'
 nonSpecialChars ::= c / c != <specialChars>
 filepath        ::= (<letterStr> | '/' | '\' | '-' | '_' | '.' | ' ')+

 #| Opciones
 optionList      ::= '[' optionListElems ']'
 optionListElems ::= <optionPair> (',' <optionPair>)*
 optionPair      ::= <letterStr> ':' <optionValue>
 optionValue     ::= <boolValue>
                 | <numberValue>
                 | <literalValue>
                 | <identifierValue>

 boolValue       ::= 'true' | 'false'
 numberValue     ::= <integer> | <float>
 literalValue    ::= '"' <nonSpecialChars> '"'
 identifierValue ::= <letterStr>

 #| Texto
 text        ::= ('\bold' | '\italic' | '\emph') '{' <textChars> '}'
 textChars   ::= (<nonSpecialChars>)+ <textChars> | '\' <specialChars> <textChars> |
<empty>

 #| Configuracion del documento
 config      ::= '\config{' <configName> '}' <optionList>
 configName  ::= 'size' | 'pagenumbering' | ... | 'verbatimnumbering'

 #| Metadata del documento
 metadata    ::= <title> <autho> <date>
 title       ::= '\title{' <text> '}' | <empty>
 author      ::= '\author{' <text> '}' | <empty>
```

```
 date        ::= '\date{' <text> '}' | <empty>

 #| Comandos
 command ::= <text>
         | <paragraph>
         | <section>
         | <subsection>
         | <figure>
         | <table>
         | <list>
         | <verbatim>
         | <newpage>
         | <hline>

 paragraph         ::= '\begin{paragraph}' <optionList> <text> '\end{paragraph}'
 section           ::= '\section{'<text> '}' <optionList>
 subsection        ::= '\subsection{'<text> '}' <optionList>
 figure            ::= '\figure{' <filepath> '}' <optionList>
 table             ::= '\begin{table}' <optionList> <tableContents> '\end{table}'
 tableContents     ::= <text> ('|' <text>)* '\break'
 list              ::= '\begin{list}' <optionList> <listContents> '\end{list}'
 listContents      ::= '\item' <text>
 verbatim          ::= '\begin{verbatim}' <optionList> <verbatimContents>
'\end{verbatim}'
 verbatimContents  ::= (<anyChar>)+ (<notFollowedBy> '\end')>
 newpage           ::= '\newpage'
 hline             ::= '\hline' <optionList>

 #| Documento
 document    ::= (<config>)* <metadata> (<command>)+
```

Funcionamiento

Para mantener el sistema lo mas modular posible **SPF** esta dividido en componentes, cada una independiente de las demas. Dado un archivo en formato **SPF** este atraviesa una serie de modulos para producir un PDF, en orden, son:

1. Parser: Se encuentra en **Parser.hs**. Se encarga de *parsear* el archivo, es decir convertir el texto del mismo en un AST. Cabe notar que en este punto no se validan los datos, el parser solo se ocupa de la correctitud *sintactica* del archivo, no la *semantica*. Puede fallar.

2. Validador: Esta compuesto por mutiples archivos, se encuentran en el directorio **Validation**. Valida los contenidos del AST generado por el parser. El trabajo principal del validador es verificar que las opciones en el archivo, tanto de configuracion como de los comandos sean correctas; Esto conlleva verificar que cada par **clave:valor** sea del tipo y forma correcta para el comando o la configuracion a el que corresponden. Devuelve otro AST y puede fallar.

3. Recursos: Se encuentra en **Resources.hs**. Carga todos los recursos necesarios para typesetear el documento. Por un lado estan las fuentes, siempre deben cargarse. Por el otro lado estan los recursos especificados por el usuario, es posible incluir recursos externos en el archivo(Mediante el comando **figure**), estos deben ser cargados en su totalidad antes de poder generar el documento. La carga de recursos puede fallar.

Un poco mas sobre ASTs

¿Como fallar?

Como manejar fallos en un programa de este tipo no es una pregunta facil de resolver, principalmente porque es posible tener *muchos* fallos.