The Syntax Sugar DFA

A DFA (Deterministic Finite Automaton) is an abstract machine that after receiving a particular input, is located in a specific state. It's very useful for parsing techniques, like the one needed for a code highlighter.

A DFA is formed by the following parts:

- an initial state
- a finite number of states
- a transition function

The automaton is called deterministic because, after the transition function has processed the input, the machine can return just one state.

Our DFA is a bit different. Other DFA also contains some ending states: for example, let's take a program that decides if a string is valid or not. If the transition function encounters valid symbols each time, it ends in an ACCEPT state. Otherwise, if a symbol isn't allowed, the machine ends in a REJECT state, and the string is recognized as erroneous.
Syntax sugar's DFA doesn't need that. It doesn't recognize an erroneous syntax like a compiler does, but it just highlights what it finds.

States.
The states and the transition function are defined in AbstractDFA.class.php, but the real implementation is in AbstractLanguageParser.class.php

```
define("NORMAL_S", 0);
define("GENERIC_COMMENT_STARTING_S", 1);
define("MULTILINE_COMMENT_BODY_S", 2);
define("MULTILINE_COMMENT_ENDING_S", 3);
define("SIMPLE_COMMENT_BODY_S", 4);
define("SIMPLE_COMMENT_ENDING_S", 5);
define("STRING_BODY_S", 6);
define("KEYWORD_S", 7);
```

As you can see, the states are nothing but values assumed by the variable $state.
These are the generic states, common to almost every language. More specific ones (like PREPROCESSOR_S for preprocessor macros) are defined later and used only when really needed.
StateTransition is the transition function I was calling about previously. It receives every character of the code as input, and then moves between states, doing the appropriate actions. The initial state is NORMAL_S, which indicates the body of the code; here the program checks for starting delimitators (comment or string delimitators) and saves ordinary characters in $partial to be confronted with the accepted keywords.
This is all very generic: strings, comments and keywords are characteristics typical of every language (except maybe assembly, which is a bit more particular). For a more particular behaviour, you have to look at the single classes.

```
case STRING_BODY_S:
        $this->data .= $c;
        if (($c == $delimiter) && (!$escaping))
        {
                $delimiter = null;
                $this->data .= "</span>";
                $this->SetState(NORMAL_S);
        }

        /*
         * Pays attention to the escape character \
         */
        $escaping = ($c == "\\") ? ($escaping ^ true) : false;
        break;
```

This state is entered if a string delimiter is encountered. The machine will be in this state until the same delimiter (not escaped) is found again. You can note that no keyword will be ever recognized inside a string (and that's good!).
The comment states are a bit more complicated because most of them involves two characters and not just one, but it's basically the same process.
As said, the DFA doesn't use an error condition; this happens because our purpose is not to look for syntax errors in the code. If a string appears outside the right context, the DFA doesn't care and highlights it anyway, while a syntax parser would generate an error state.

The StateTransition function can be extended by classes of languages which desires to add new functionalities. For example, the C class needs to add a new state for handling preprocessor lines:

```
array_push($this->states, PREPROCESSOR_S); /* adds a new state */
```

It handles this state, and for every other input calls the original StateTransition procedure.
As you can see, a DFA is a powerful machine which enable the programmer to create a good code, simple to understand and simple to mantain.

shainer
<syn.shainer@gmail.com>
http://giudoku.sourceforge.net