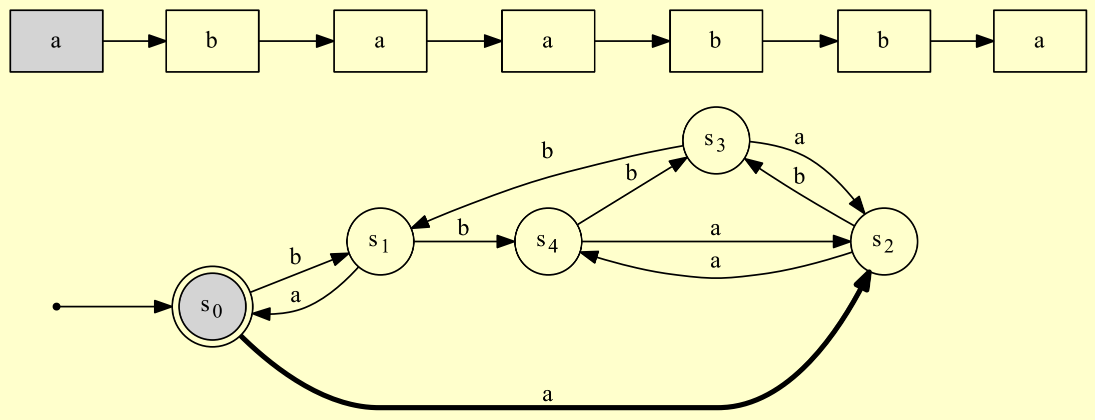
XWizard: **The Online Informatics Toolbox**

– Handbook for Students –

Lukas König, Friederike Pfeiffer-Bohnen



Script ID-10700

XWizard: **The Online Informatics Toolbox**

– Handbook for Students –

Contents

1 What is XWizard?

XWizard is a free (web) tool for the automatic **visualization, manipulation and PDF generation** of many types of objects from theoretical computer science (such as Turing machines, push-down and finite automata, Chomsky grammars etc.). A broad range of algorithms can be applied to the objects, producing intuitive and customizable views. XWizard is well-suited for students' self-studies, and it is powerfull in aiding teachers at the creation of course material such as exercises (the X in XWizard stands for "eXercise" – and also for "anything").

XWizard can be used to create a variety of object types. To give an overview of its range of functionality, the following section lists the most important object types including their main features. If you want to learn how to work with these objects, manipulate them and create new ones, skip to section 3 right away.

1.1 Native Object Types

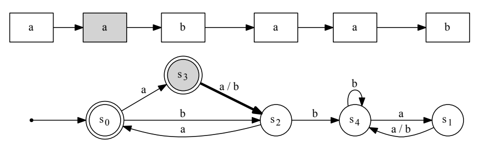
The main XWizard object types are categorized into one of the two groups "Theoretical Computer Science" or "Practical Computer Science" (being aware that not all do perfectly fit into only one group; in future these groups may change). The list below follows this categorization as well. Below the name of each object type, its main features are listed.

Below the figures, there is a link to the corresponding object on the XWizard website which can be used to play around with XWizard and informally get the idea of how it works.

Theoretical Computer Science

- **Finite state machines (FSMs):**
  - pre-implemented examples
  - arbitrary self-defined or random FSMs
  - step-wise simulation (det. or non-det.) of arbitrary input words
  - minimization
  - creation of equivalent deterministic FSM
  - conversion into equivalent
    - push-down automaton
    - Turing machine
    - Chomsky grammar
    - regular expression
  - ...

FSM:



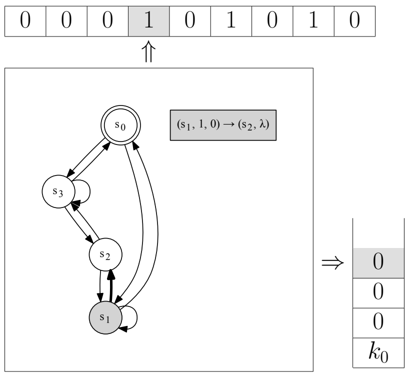
Minimization table:

s_1	$\times_0$			
s_2	$\times_0$	$\times_1$		
s_3	$\times_1$	$\times_0$	$\times_0$	
s_4	$\times_0$	–	$\times_1$	$\times_0$
	s_0	s_1	s_2	s_3

Script ID-13414

- **Push-down automata (PDAs):**
  - pre-implemented examples
  - arbitrary self-defined PDAs
  - step-wise simulation (det. or non-det.) of arbitrary input words
  - multiple calculations in parallel
  - conversion into equivalent
    - Chomsky grammar
    - Turing machine

    under construction
  - ...



Script ID-13417

- **Turing machines (TMs):**
  - pre-implemented examples
  - arbitrary self-defined or random TMs
  - simulation (det. or non-det.) of arbitrary input words
  - multiple calculations in parallel
  - conversion into equivalent Chomsky grammar

    under construction
  - step-wise simulation

    under construction
  - ...

	1	$\star$
A	$(B,1,L)$	$(B,1,R)$
B	$(C,\star,L)$	$(A,1,L)$
C	$(D,1,L)$	$(H,1,R)$
D	$(A,\star,R)$	$(D,1,R)$
H		

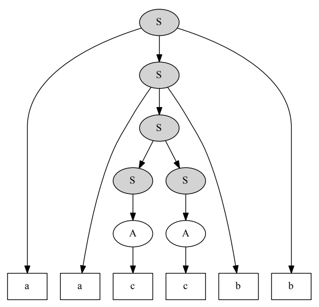
Tape	Transition
$\hat{\star}$	$(A,\star) \to (B,1,R)$
$1\hat{\star}$	$(B,\star) \to (A,1,L)$
$\hat{1}1$	$(A,1) \to (B,1,L)$
$\hat{\star}11$	$(B,\star) \to (A,1,L)$
$\hat{\star}111$	$(A,\star) \to (B,1,R)$
$1\hat{1}11$	$(B,1) \to (C,\star,L)$
$\hat{1}\star 11$	$(C,1) \to (D,1,L)$
$\hat{\star}1\star 11$	$(D,\star) \to (D,1,R)$
$1\hat{1}\star 11$	$(D,1) \to (A,\star,R)$
$1\star\hat{\star}11$	$(A,\star) \to (B,1,R)$
$1\star 1\hat{1}1$	$(B,1) \to (C,\star,L)$
$1\star\hat{1}\star 1$	$(C,1) \to (D,1,L)$
$1\hat{\star}1\star 1$	$[(D,\star) \to (D,1,R)]$

Script ID-16304

- **Chomsky grammars:**
  - pre-implemented examples
  - arbitrary self-defined or random grammars
  - parse tree visualization
  - single word derivation
  - tree of all derivable words, limited by derivation or word length
  - conversion into equivalent
    - epsilon-free grammar
    - Chomsky normal form
    - Greibach normal form
    - Kuroda normal form
    - PDA
  - ...

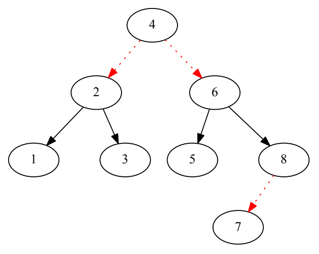
(Most algorithms applicable to type-3 / type-2 (or type-1: Kuroda NF) grammars only.)

$G = (\{A, S\}, \{a, b, c\}, P, S)$
$P = \{S \to A \mid SS \mid aSb,$
$A \to c \mid AA\}$



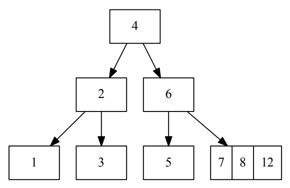
Script ID-13423

- **Red-black trees:**
  - pre-implemented examples
  - arbitrary red-black trees given by
    - insertion order or
    - explicit tree definition
  - conversion into equivalent
    - 2-3-4 tree
  - based on strings or numbers



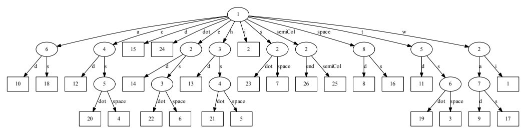
Script ID-13425

- **2-3-4 trees:**
  - pre-implemented examples
  - arbitrary 2-3-4 trees given by
    - insertion order or
    - explicit tree definition
  - conversion into equivalent
    - 2-3-4 tree
  - based on strings or numbers



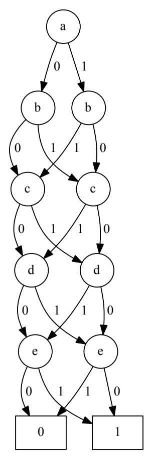
Script ID-13426

- **Pat trees:**
  - pre-implemented examples
  - arbitrary Pat Trees given by text base
  - so far visualization only


Script ID-13428

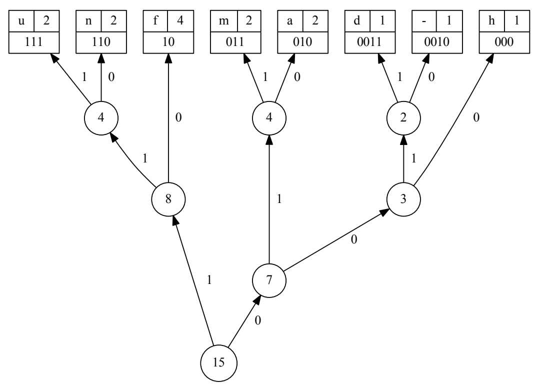
Practical Computer Science

- **Binary decision diagrams (BDDs):**
  - pre-implemented examples
  - calculation of BDDs from arbitrary Boolean functions
  - Visualization of BDDs
  - Visualization of step-wise creation of BDDs



Script ID-62

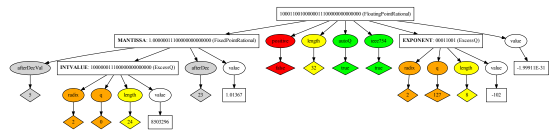
- **Huffman codes:**
  - pre-implemented examples
  - calculation of Huffman trees for arbitrary symbol distributions
  - Two different visualization modes (bottom-up and top-down)



Script ID-72

- **Number representations:**
  - ExcessQ
  - Fixed-point rational
  - Floating-point rational (including IEEE754)
  - One's complement
  - Two's complement
  - Conversion between these representations and decimal
  - Arbitrary bases from binary to 36.

$1\ 00011001\ 00000011100000000000000_{ieee754} = -1.99911E-31$

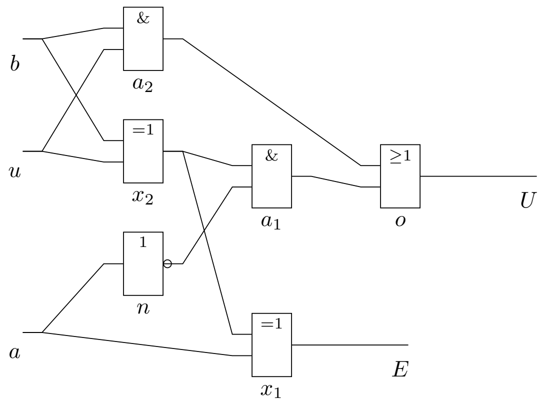


Script ID-15538

Furthermore, the following object types are currently available, but under construction, i. e., they only have little functionality so far, and the depiction might be confusing (particularly this is the case for large logic circuits):

- **Logic circuits:**

  under construction (So far, logic circuits can only be displayed.)



Script ID-8861

- **Regular expressions:**

  under construction (More precisely, regular expressions do work well, but all they can do so far is listing their words, see example.)

$(a+b)^\star c + \emptyset^\star$

First 16 words: $\lambda, c, ac, bc,$
$aac, abc, bac, bbc, aaac, aabc, abac, abbc, baac, babc,$
$bbac, bbbc, \ldots$

Script ID-14238

Note that the very specific object type MARB (for example SCRIPT ID-1847) as well as the simple calculator (SCRIPT ID-235) and the meta-type MetaProperties (SCRIPT ID-15847) are not listed here. A quick help for each of them is provided on the XWizard website; the MetaProperties type can be used to create the overview of object types on page 11 of this document. Moreover, XWizard is constantly under construction, meaning that both new object types and improvements of the available ones can occur at any time.

1.2 Plain PDF Generator Code Types

Besides the prefabricated object types, XWizard can process arbitrary LaTeX and Graphviz code. This feature has originally been implemented for technical reasons[1], but once available, it brings along the possibility to create complex objects which go far beyond the native XWizard types. A detailed description of the features which come with these plain types is not within the scope of this "average user"-oriented documentation. For more details on customizing the XWizard output, creating complex objects which may contain data from several of the native types, and on generating exercises to be solved online by students (cf. Sec. 4), see the handbook "XWizard for Teachers".

As a pleasant side-effect, students unfamiliar with LaTeX or Graphviz are free to use XWizard as an easy-to-use learning tool which works with arbitrary documents and does not require the extensive installation of LaTeX or Graphviz on a home computer.

2 Access to XWizard

XWizard can be simply accessed via:

`www.xwizard.de`

or by clicking (or scanning) any of the script links in this document, such as:

SCRIPT ID-10700

In addition to the web version, there exists a download version called **Very Fast PDF Generator (VFP)**. It can be retrieved (including installation instructions) from:

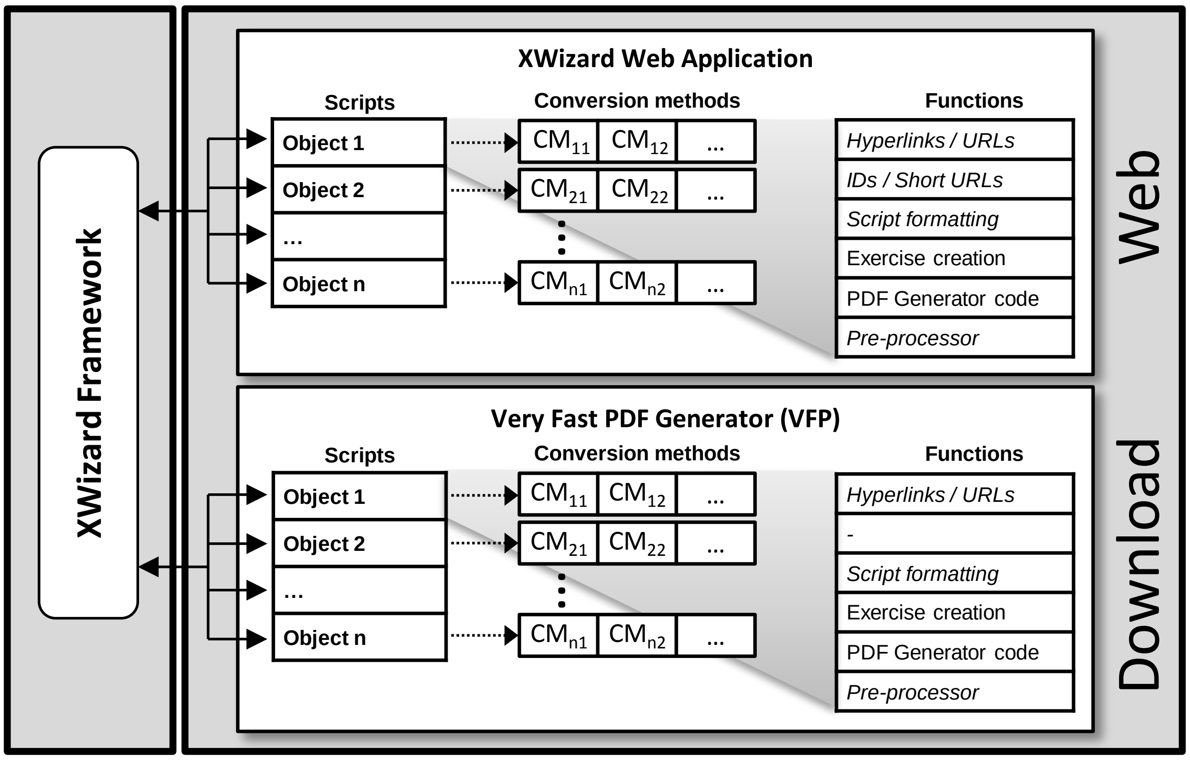
`https://sourceforge.net/projects/easyagentsimulation`

Both versions are mostly innerchangeable, therefore, the term XWizard will refer to both in the following, except where a difference between the two is explicitly addressed. Such as in the following:

XWizard can be switched between English and German language; so far, VFP is only available in English.

The following figure shows the structure of the XWizard components. The XWizard work flow based on scripts, conversion methods etc. is described in the next section.

[1]Since all XWizard objects are created by either PDF LaTeX or Graphviz DOT (or GNUPlot when using the download version).



XWizard and all implementations related to it are *free software*[2]. They are allowed to be run for any purpose (except commercial), and the sourcecode can be studied, changed, and further distributed in accordance with this legal notice: `http://www.xwizard.de:8080/Wizz?impressum&lang=eng`.

3 Working with XWizard

There are two main mechanisms which can be utilized when working with XWizard: **scripts**, which define a specific object, and **conversion methods**, which provide a mechanism to apply algorithms to an object (e. g., minimizing an FSM). More precisely, scripts establish the universal framework to control XWizard, meaning that, in principle, all functionality can be exploited by just writing scripts. Conversion methods are implemented as a special case within this framework. They provide a simple graphical access point for users and do not involve writing scripts at all. Many typical applications of XWizard can be established solely with conversion methods.

3.1 Scripts

XWizard's basic workflow is simply to process a script, translate it into a PDF image and display this image. A script is usually built up of three parts (or else four, when encoding a script conversion at the bottom, see below), as illustrated in the following example of a PDA:

```
pda:                                                   ⇐ Script preamble
    (s0, 0, k) => (s1, 0k);                            ⎫
    (s0, 1, k) => (s3, 1k);                            ⎪
    (s1, 0, 0) => (s1, 00);                            ⎪
    (s1, 1, 0) => (s2, lambda);                        ⎪
    (s1, lambda, k) | (s3, lambda, k) => (s0, k);      ⎪
    (s2, lambda, 0) => (s1, lambda);                   ⎬ Main script part
    (s2, lambda, k) => (s3, bk);                       ⎪
    (s3, 0, 1) => (s3, b);                             ⎪
    (s3, 0, b) => (s3, lambda);                        ⎪
    (s3, 1, 1) => (s3, 11);                            ⎪
    (s3, 1, b) => (s3, b1);                            ⎭
```

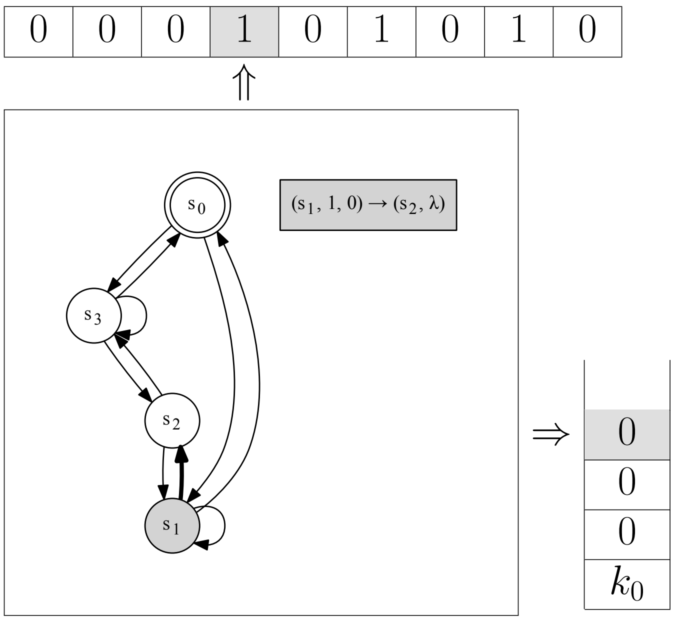
[2]`https://en.wikipedia.org/wiki/Free_software`

```
--declarations--
    e=#n#;
    s0=s0;
    F=s0;
    kSymb=k;
    inputs=000101010;
    simSteps=3;
--declarations-end--
```

$\Big\}$ Variable declarations

All scripts include the following three parts: a **preamble** determines the type of object defined by the script; a **main part** defines the actual structure of the object; variables in a **declarations part** can be used to assign many types of additional properties (both the complete declarations part and some of the variables can be omitted, in which case the according variables are set to standard values).

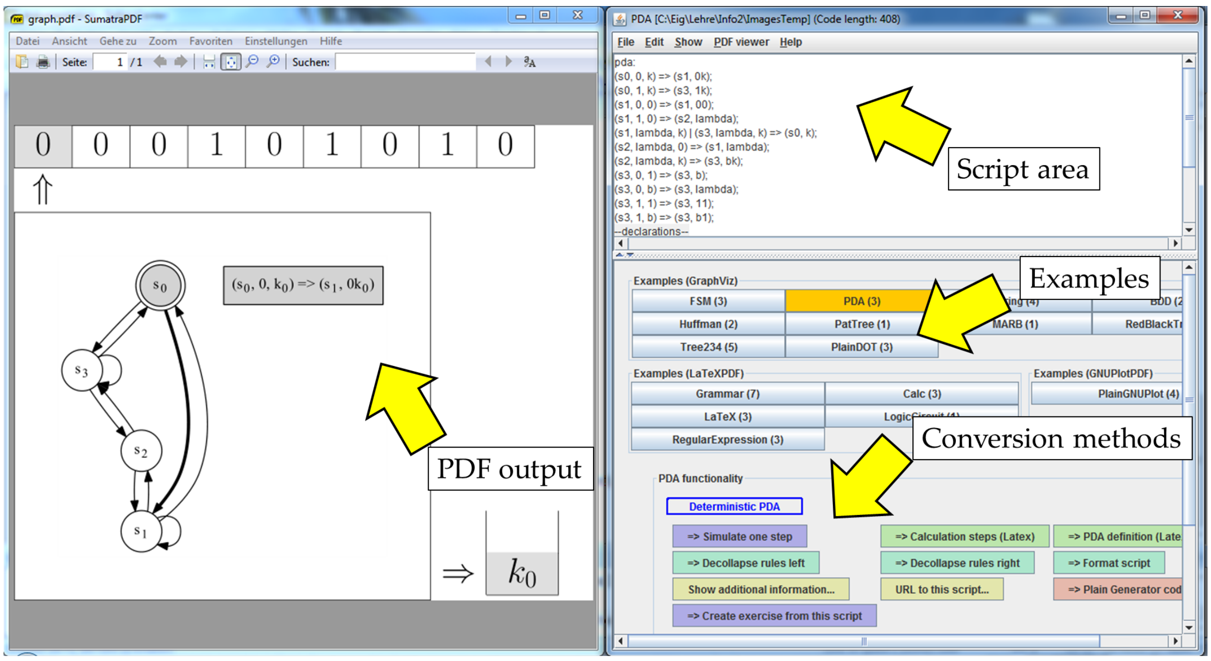
This example script will create the following image (using Graphviz and LaTeX in the background):



Script ID-13417

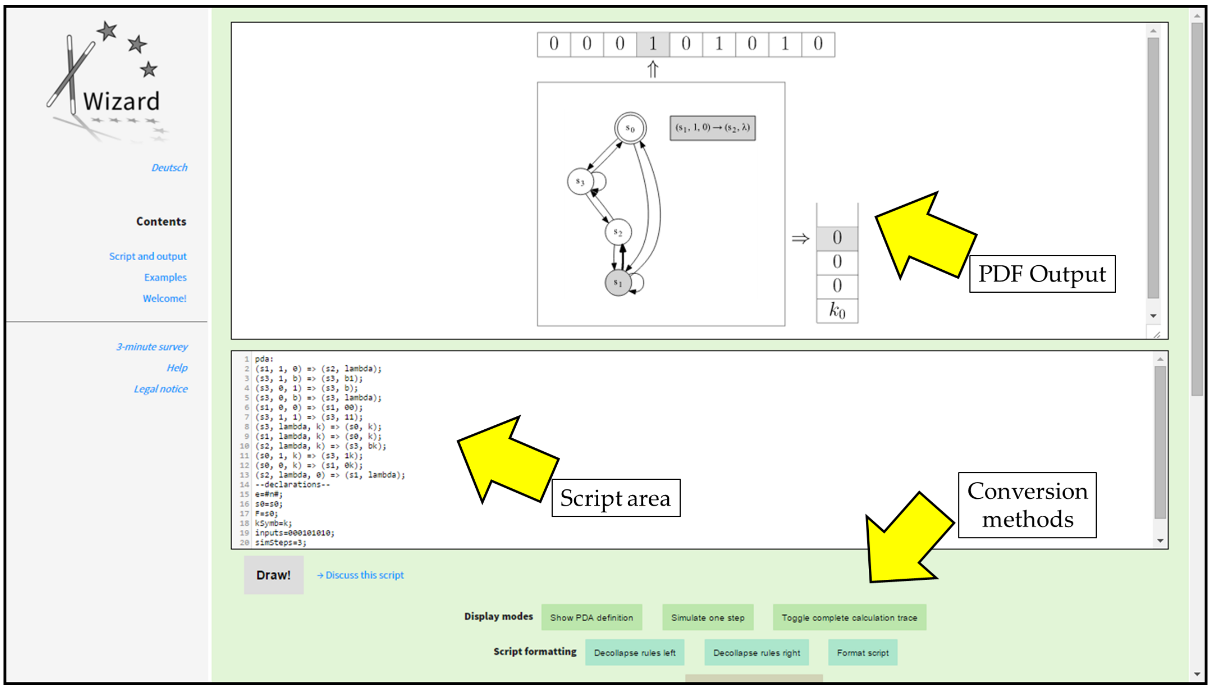
The script can be pasted into the script areas of both VFP and XWizard to produce the given output, see screenshots below (clicking or scanning the above QR code will open XWizard and automatically execute the script).

VFP:



```
pda:
(s0, 0, k) => (s1, 0k);
(s0, 1, k) => (s3, 1k);
(s1, 0, 0) => (s1, 00);
(s1, 1, 0) => (s2, lambda);
(s1, lambda, k) | (s3, lambda, k) => (s0, k);
(s2, lambda, 0) => (s1, lambda);
(s2, lambda, k) => (s3, bk);
(s3, 0, 1) => (s3, b);
(s3, 0, b) => (s3, lambda);
(s3, 1, 1) => (s3, 11);
(s3, 1, b) => (s3, b1);
--declarations--
```

XWizard:

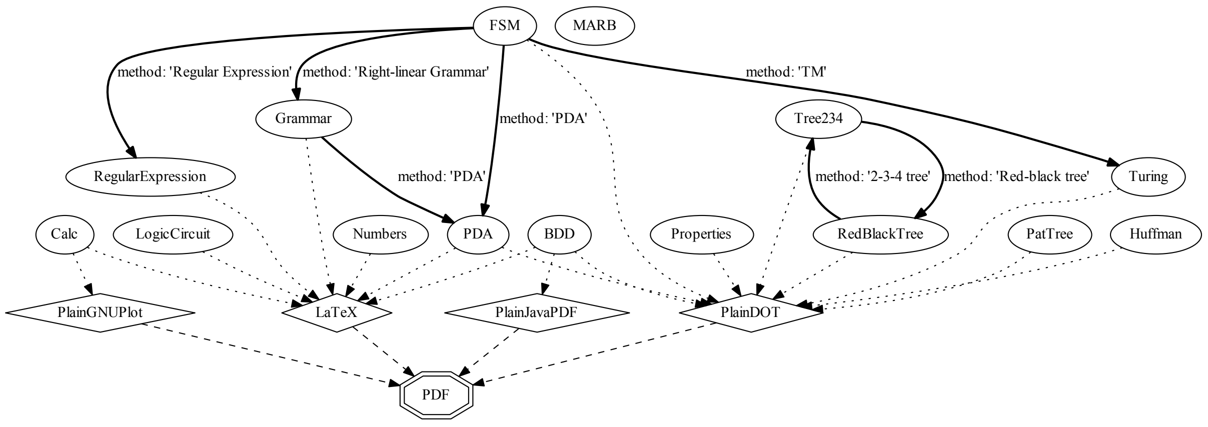


```
pda:
(s1, 1, 0) => (s2, lambda);
(s3, 1, b) => (s3, b1);
(s3, 0, 1) => (s3, b);
(s3, 0, b) => (s3, lambda);
(s1, 0, 0) => (s1, 00);
(s3, 1, 1) => (s3, 11);
(s3, lambda, k) => (s0, k);
(s1, lambda, k) => (s0, k);
(s2, lambda, k) => (s3, bk);
(s0, 1, k) => (s3, 1k);
(s0, 0, k) => (s1, 0k);
(s2, lambda, 0) => (s1, lambda);
--declarations--
e=#n#;
s0=s0;
F=s0;
kSymb=k;
inputs=000101010;
simSteps=3;
```

The calculation of the displayed PDA can be carried on by increasing the “simSteps” variable in the script or by repeatedly clicking the **conversion method** “Simulate one step” (cf. description of conversion methods below).

When using VFP, the PDF output is automatically updated each time the script area changes. When using XWizard, the image output is created after clicking the “Draw!” button on the web page. A PDF can be retrieved by clicking the “Download PDF” link; it appears when scrolling down below the PDF output.

An overview of the script types available at creation time of this document are depicted in the following figure (solid and dotted arrows denote script type transitions by conversion methods; dashed arrows denote the PDF creation process; diamond nodes represent plain PDF generator script types – see below for details on these terms).



A current version of the figure can be retrieved via this link (note that it is itself generated by a simple XWizard script):

Script ID-15812

For any of these object types, there are example scripts on the XWizard web page:

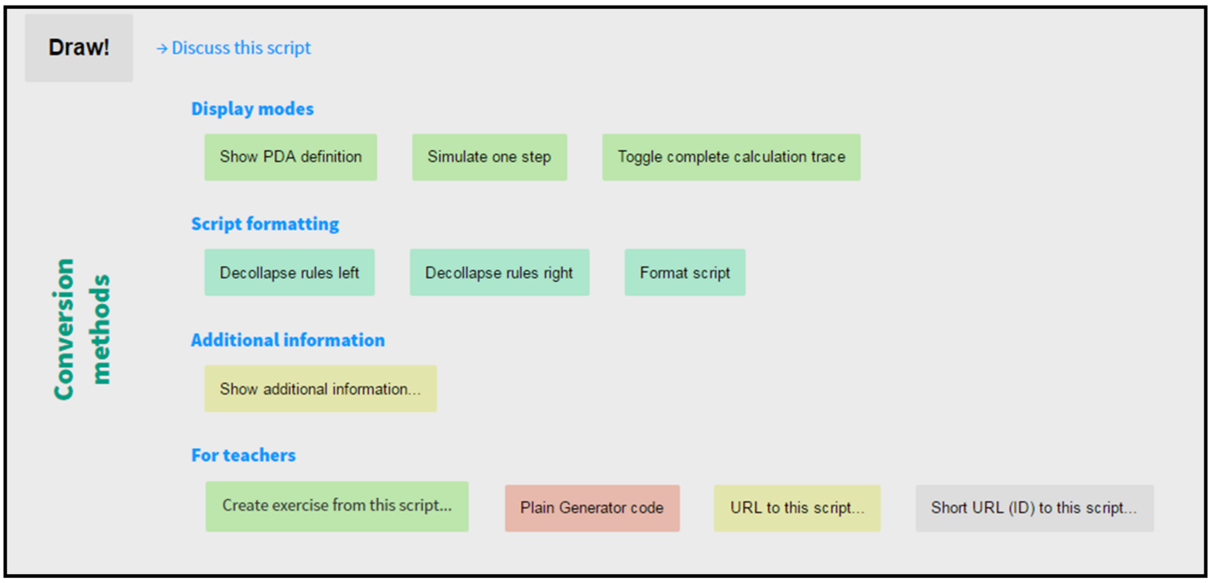
`http://www.xwizard.de:8080/Wizz?lang=eng&hide#Examples`

3.2 Conversion Methods

Besides creating and displaying objects, a main functionality of XWizard is to apply algorithms to scripts, e. g., to visualize the stepwise computation of a PDA or to minimize an FSM. Algorithms are applied to objects by using conversion methods, i. e., methods that transform one script into another.

When applying a conversion method, the script belonging to the current object is replaced with the new script created by the conversion method, and the object defined by the new script is created and displayed.

The easiest way to apply a conversion method to a script is to click the according button in the "Conversion methods" area of the graphical user interface. The following screenshot shows the conversion methods available for PDA scripts in XWizard:

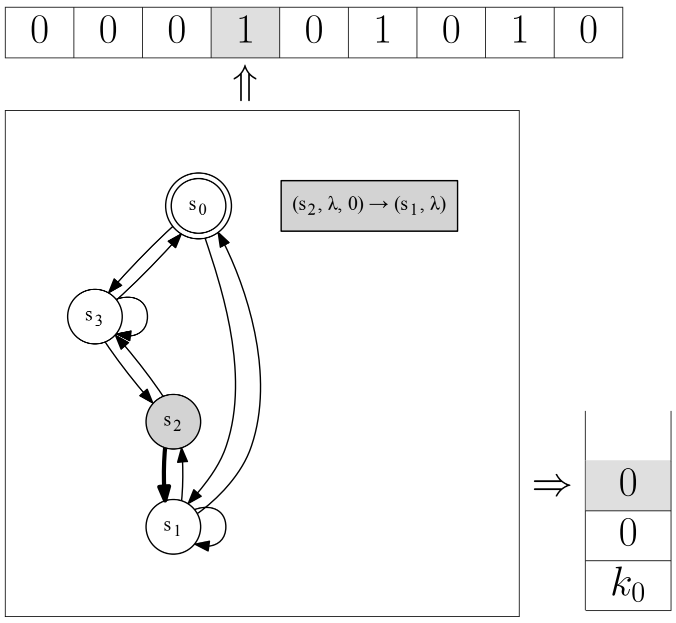


For example, when clicking the "Simulate one step" button, the PDA script will be translated into a very similar script where the display mode is switched to step-wise visualization; at repeated clicks, the `simSteps` variable will get incremented which, in turn, will result in a PDA simulated one step further than before. Overall, the conversion methods in the category "Display Modes" can be used to switch between step-wise and complete calculation view, a visualization of the PDA definition, and combinations of those (cf. feature overview in section 1.1).

The according script created by the conversion method is the following:

```
pda:
    (s1, 1, 0) => (s2, lambda);
    (s3, 1, b) => (s3, b1);
    (s3, 0, 1) => (s3, b);
    (s3, 0, b) => (s3, lambda);
    (s1, 0, 0) => (s1, 00);
    (s3, 1, 1) => (s3, 11);
    (s3, lambda, k) => (s0, k);
    (s1, lambda, k) => (s0, k);
    (s2, lambda, k) => (s3, bk);
    (s0, 1, k) => (s3, 1k);
    (s0, 0, k) => (s1, 0k);
    (s2, lambda, 0) => (s1, lambda);
--declarations--
    e=#n#;
    s0=s0;
    F=s0;
    kSymb=k;
    inputs=000101010;
    simSteps=4;            /* Value increased by conversion method. */
    maxNondetCalcDepth=12
--declarations-end--
```

The resulting object looks like this:



Script ID-13100

Click the QR code and try it yourself!

3.3 Applying Conversion Methods via Scripts

The application of conversion methods via scripts is an advanced feature added for interested readers. It is not mandatory for working with XWizard in the usual way, so you may well skip this section.

As mentioned above, scripts control XWizard completely; this particularly means that even the application of conversion methods can be initiated via a special script type. Such a **conversion script** starts with the regular three parts, which determine the script to be converted. As a fourth part, a **conversion command** is written as the last line of the script. A conversion command looks like this:

```
**>CM-NAME<**
```

where `CM-NAME` is the English name of the conversion method to be applied or, if the conversion method requires parameters:

```
**>CM-NAME[p1, p2, ...]<**
```

where `p1, p2, ...` are the method parameters (they can be put in quotes if they are supposed to include special characters such as white spaces or commas: `["p, a, r, 1", "p, a, r, 2", ...]`). For example, the "Simulate one step" conversion method can be applied to a PDA script by adding the red-colored last line to it:

```
pda:
    (s1, 1, 0) => (s2, lambda);
    (s3, 1, b) => (s3, b1);
    (s3, 0, 1) => (s3, b);
    (s3, 0, b) => (s3, lambda);
    (s1, 0, 0) => (s1, 00);
    (s3, 1, 1) => (s3, 11);
    (s3, lambda, k) => (s0, k);
    (s1, lambda, k) => (s0, k);
    (s2, lambda, k) => (s3, bk);
    (s0, 1, k) => (s3, 1k);
    (s0, 0, k) => (s1, 0k);
    (s2, lambda, 0) => (s1, lambda);
--declarations--
    e=#n#;
    s0=s0;
    F=s0;
    kSymb=k;
    inputs=000101010;
    simSteps=3;
    maxNondetCalcDepth=12
--declarations-end--
**>Simulate one step<**   /* This is a conversion command. */
```

When entering this script, the result will be exactly the same as after clicking the according button on the script without the conversion command.

3.4 Syntax and Features of the Native Script Types

XWizard is constantly being further developed, therefore, it does not seem practical to describe all script types and features in this rather static handbook. On the other hand, the XWizard website includes a comprehensive help page in both English and German, with descriptions of all main script types, conversion methods and additional features. Therefore, please refer to this help page for an up-to-date guide to most XWizard features:

`http://www.xwizard.de:8080/Wizz?help&lang=eng` *(English help page)*
`http://www.xwizard.de:8080/Wizz?help&lang=ger` *(German help page)*

3.5 URLs and Short URLs to XWizard scripts

XWizard scripts can easily be exchanged with others by using URLs which, in XWizard, are either **regular URLs** or **short URLs**. For creating URLs, the according conversion buttons (1) "URL to this script" and (2) "Short URL (ID) to this script" can be used (note that, strictly speaking, (2) is not a conversion method and therefore not available in VFP). When creating a regular URL (1), the current script will be transformed into a URL, meaning that the complete script string will be encoded in the URL. As opposed to that, when creating a short URL (2), the script will be stored in a database and associated to a number, the script **ID**. The retrieved URL will only contain the ID, and therefore, it will usually be much shorter:

(1) Example of a regular URL to a BDD script:

`http://www.xwizard.de:8080/Wizz?template=bdd%3A+a%2Cb%2Cc%2Cd%2Ce%3A+01101001100101101001011001101001`

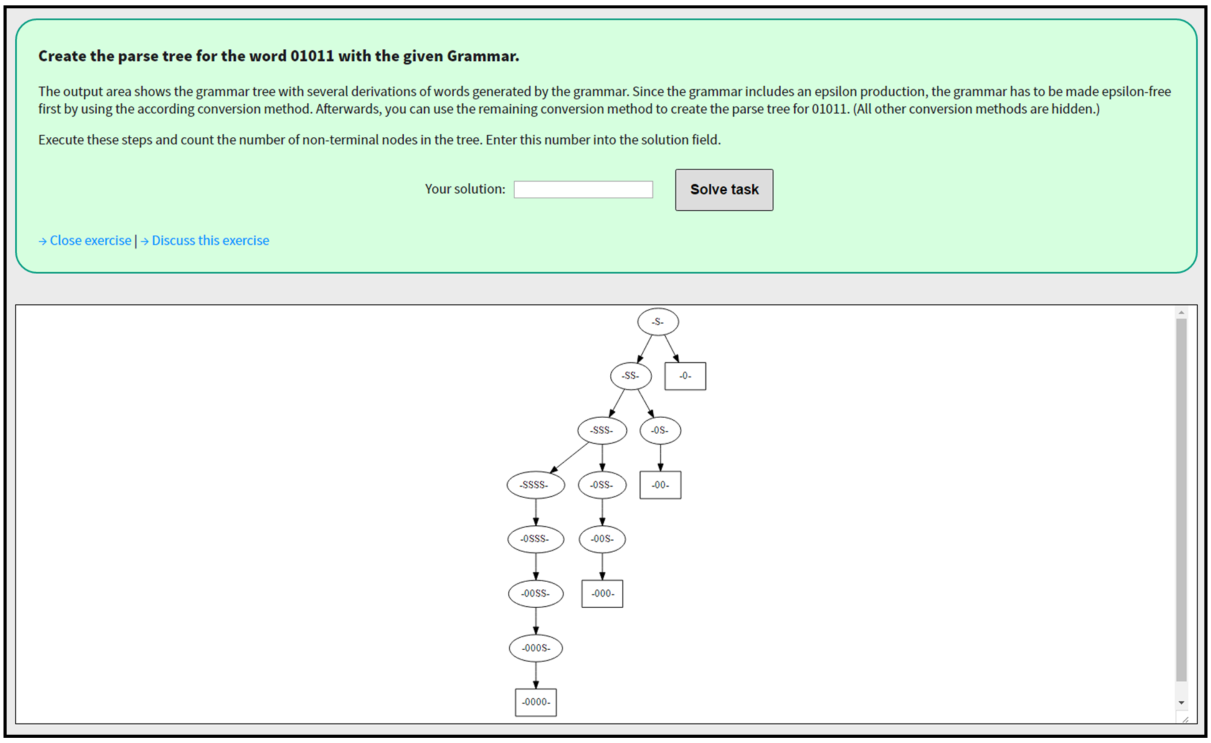
(2) Example of a short URL to the same script:

`http://www.xwizard.de:8080/Wizz?template=ID-16130`

Both URL types are in principle equivalent in their functioning. However, regular URLs can get so long that current browsers might not even accept them; also, anti-malware programs might produce alerts due to the unusual form of such URLs. Therefore, in many cases, short URLs should be preferred over regular ones. Nevertheless, it has to be noted that short URLs depend on the XWizard database while regular URLs are completely **stand-alone**. The database is of course properly maintained and archived, but a regular URL may, from this point of view, be a little more reliable than a short URL.

4 Solving Exercises with XWizard

Besides the regular mode described so far, XWizard can run in a so-called **exercise mode**. The exercise mode is triggered if a script includes the encoding of an "exercise" in its declarations part (see below for an example). An exercise is a question, e. g., posed by a teacher for a group of students, which the students are supposed to solve. The teacher can distribute a link to the students which leads them to the exercise. In exercise mode, the question to solve is displayed above the script area of XWizard. The user is allowed to use all **available** conversion methods in the quest of a solution, however, the exercise definition can state that some conversion methods are hidden. The screenshot below shows an example exercise including its script, displayed on the XWizard website. (Note that exercises cannot be displayed using VFP.)



Create the parse tree for the word 01011 with the given Grammar.

The output area shows the grammar tree with several derivations of words generated by the grammar. Since the grammar includes an epsilon production, the grammar has to be made epsilon-free first by using the according conversion method. Afterwards, you can use the remaining conversion method to create the parse tree for 01011. (All other conversion methods are hidden.)

Execute these steps and count the number of non-terminal nodes in the tree. Enter this number into the solution field.

Your solution:

Script ID-11020

The user can enter a presumed solution in the text field entitled "Your solution". If the solution is correct, XWizard will print out a congratulations message, as well as, if provided by the creator of the exercise, an additional explanation and a "secret word" (so far, the secret word has no further meaning, it is only displayed as a small reward for solving the task). After the solution has been entered correctly (and at any other time), XWizard can be switched back to regular mode by clicking the "Close exercise" link.

The script below shows how an exercise is encoded in the declarations part of a script (colored in red).

```
grammar:
    A => A, A | 0 | epsilon;
    E => A, 1, A;
    S => E, E, E | S, S | 0;
--declarations--
    e=#tit=~Create the parse tree for the word 01011 with the given Grammar.~,
       exp=~<P>The output area shows the grammar tree with several derivations
            of words generated by the grammar. Since the grammar includes an epsilon
            production, the grammar has to be made epsilon-free first by using
            the according conversion method. Afterwards, you can use the remaining
            conversion method to create the parse tree for 01011. (All other
            conversion methods are hidden.)</P><P>Execute these steps and count the
            number of non-terminal nodes in the tree. Enter this number into the
            solution field.</P>~,
       sol=~6~,
       cod=~parser-guru~,
       met=~.*Epsilon.*|.*Parse.*~,
       cur=~.*~,
       tar=~.*~,
       crypt=~false~,
       excrypt=~false~,#;
    N=S,E,A;
    T=0,1;
    S=S;
    displayMode=2;
    maxdepth=8;
    cutNonTerminalBranches=true;
    cutTerminalDoubleBranches=true;
    maxLengthWords=4;
    multiLetterSymbolsHaveIndex=true;
    parseTreeNum=0;
--declarations-end--
```

In this case, the exercise is encoded as plain, human-readable text – which makes cheating easy. To preserve the excitement of actually solving an exercise, two levels of encryption can be applied to the scripts by their creators, encrypting either the exercise definition only or the whole script. Details of the encoding of exercises and the encryption can be reviewed in the "handbook for Teachers".

Of course, the execution of conversion methods via script, as described in Sec. 3.3, is not available for encrypted scripts. Otherwise, hidden conversion methods could be executed.

5 Legal Note

Have fun with XWizard!

This document has been compiled on September 10, 2016