Using `ruleR` as a framework for rule-based item generation

Maria Rafalak
m.rafalak@practest.com.pl

Philipp Doebler
philipp.doebler@googlemail.com

December 4, 2013

1 Introduction

The success of a psychological test is largely determined by the quality of its items. In classic linear testing, the number of items needed to measure a single latent trait with acceptable reliability is often in the range of 20 to 60 items. While it is certainly possible to construct this number of items with the help of heuristics it is often possible to identify rules governing the item construction process. For example for the classic Advanced Progressive Matrices Test ([Rav62]) five (broad) rules used in its construction have been identified ([CJS90]). These rules and their combinations are used on the rows and/or columns of a matrix resulting in many different stimuli. Also number sequences used in intelligence tests are often derived from basic rules like addition and subtraction (of constants or two previous numbers), multiplication and the digitsum (sum of the digits of a number).

Recently *automated item generation* (AIG; [IK02]) has been explored in various applied contexts ([Are05, ASGH06, HBZ09, HBKK10, Zeu10]). The main idea of AIG is to identify the underlying template or rule(s) that constitute an item and to generate new items (potentially infintely many) using the template or rule(s). Two main approaches can be identified: *item cloning* (IC) and *rule-based item generation* (RIG). The first approach uses an existing item (a parent), typically with known psychometric qualities, and produces a clone (a child) of that item by changing its *surface features* (or incidentals). For example in a statistics exam for university students, the cover story of the example is changed, but the student is nevertheless to make the same calculations (e.g. [HBZ09]). RIG on the other hand focuses on the rules (sometimes called radicals) that govern the item construction process. Once the rules and their relations (and their relation to surface features of items) are known, a new item can be generated from a (combination of) rule(s). Often it is possible to predict the difficulty of an item by using a linear logistic test model or a relative (LLTM; [Fis73]; [GGvdL11]).

There are several situations in which automated item generation is favourable:

1. Linear tests, especially in high stakes situations like college admission, are often used only in one year since the test security can not be guaranteed once the test has been exposed to a large population ([AS12]). Here automated item generation leads to tests for which the answers can not be learnt by heart.

2. Computer adaptive testing (CAT; [vdLG10]; [VDLG00]; [WDF$^+$00]) relies on large pools to cover a wide range of potential person abilities. It is often expensive to produce items, so automating the process is certainly desirable here. Also if the psychometric properties of parent items or rules are known, the CAT algorithm can generate items on the fly.

The identification of rules is not the same as an implementation of a rule-based item generation algorithm, but it is a necessary step. Besides providing a basis for RIG, an analysis of the cognitive task at hand is a result of this identification. Another side product is that a suitable psychometric model can sometimes be found after such an analysis.

In the following we identify some rules for number sequence items and explain details of their implementation in the `ruleR` package. We aim to provide a framework to generate number sequence

items. The package uses S4 classes to represent rules and is written with the understanding that the user will eventually want to extend the existing possibilities; this vignette demonstrates an extension.

While R itself is not a front end for computer based testing, its applicability has been successfully demonstrated, for example in the form of the concerto testing platform ([Kos12]). Several R packages are worth mentioning in this context: catR ([MM11]), which provides functionality for computer adaptive testing, ltm ([Riz06]), which can be used to perform a range of psychometric analyses and RMySQL ([JD12]) which handles the interaction of R and MySQL databases.

2 Number sequences

Before we showcase a sample R session in which ruleR is used to generate number sequences, we explain some of the ideas behind the package.

2.1 Number sequence items and their cognitive analysis

The analysis of number sequence items often focuses on their psychometric properties, be it those derived from classical test theory ([LNB68]) or from item response theory (IRT; [HRS95, vdLH97, ER00, BK04]). CTT and IRT have developed precise notions of difficulty (and discriminatory power) of items. Interesting questions though remain: What determines the difficulty of an item? Can the difficulty be predicted from the way the item is constructed?

A statistical approach towards an answer to these questions is provided by the linear logistic test model (LLTM, [Fis73]), or more generally by explanatory IRT models ([DBW04]). These models have been developed to explain item properties from intrinsic item features (and other variables). An analysis with such models requires to study the underlying cognitive process to isolate relevant item features.

For the cognitive analysis of number sequence items, we will focus on the radicals (basic rules) and their relationships[1]. Table 1 lists some possible radicals. We will assume that the examinee is familiar with the radicals and knows that they can be combined. Solving a number sequence item then involves combining the known radicals and checking which combination generates the sequence at hand.

As an example we study some combinations of the Fibonacci rule and the digit sum rule. The next element in a Fibonacci sequence is formed by adding up the two previous element. Each such sequence needs two *seeds* to get started. Taking 1 and another 1 as seeds results in the following sequence:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, ...$$

The digit sum is formed by adding up all the digits of a number. For example, the digit sum of 23 is $2 + 3 = 5$. Figure 1a shows one way to combine the Fibonacci and digit sum rule: after the sum of the two previous numbers is calculated, the digit sum is formed. Using 23 and 4 as seeds, we obtain the following sequence:

$$23, 4, 9, 4, 4, 8, 3, 2, 5, 7, 3, 1, 4, 5, 9, 5, ...$$

But there are other ways to combine the Fibonacci and digit sum rule, we could for example first calculate the digit sum of both numbers and then apply the Fibonacci rule, as shown in Figure 1c. Again using 23 and 4 as seeds we get the following sequence:

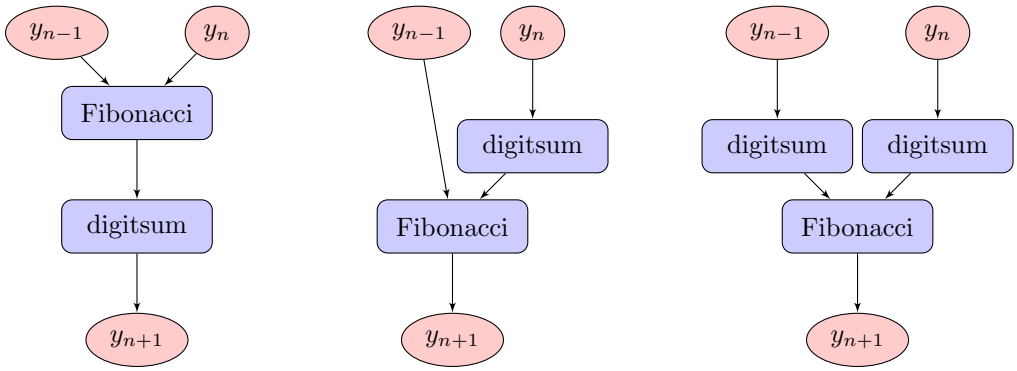
$$23, 4, 9, 13, 13, 8, 12, 11, 5, 7, 12, ...$$

The two combinations result in sequences which differ after the third element and both are more difficult than pure Fibonacci items. Yet another way to combine rules is shown in Figure 1b. Probably the items one could make out of these sequences will have different psychometric properties. To handle the fact that not only the presence of the radicals influences the difficulty but

[1] For the sake of brevity we will not discuss the influence of the magnitude of the numbers here.

Table 1: Some radicals and som possible combinations

radical	arity	term
negation (neg)	1	$-a$
addition (add)	2	$a + b$
digit sum (digitsum)	1	sum of the digits of a (base 10)
combination	**arity**	$y_{i+1} =$
add a constant	1	$\mathrm{add}(y_i, c)$
subtract a constant	1	$\mathrm{add}(y_i, \mathrm{neg}(c))$
add the digitsum of y_i and y_{i-1}	2	$\mathrm{add}(y_i, \mathrm{digitsum}(y_i))$
add a constant to the digitsum	1	$\mathrm{add}(c, \mathrm{digitsum}(y_i))$
digitsum after adding a constant	1	$\mathrm{digitsum}(\mathrm{add}(y_i, c))$
digitsum after Fibonacci	2	$\mathrm{digitsum}(\mathrm{add}(y_{i-1}, y_i))$



(a) Rule A ~ (b) Rule B ~ (c) Rule C

Figure 1: Three different ways to combine the Fibonacci and digitsum rules

also the (partial) order in which they are applied, `ruleR` proposes a tree structure[2] to describe the combinations of radicals.

2.2 Using `ruleR` to generate number sequence items

We show how to use `ruleR` in an interactive R session, assuming that readers know some basic R. The `ruleR` package does intentionally not hide all traces of S4 classes, since it is expected users will want to write their own extensions. Good starting points to learn about S4 classes are [Cha06] or the vignette of the `Brobdingnag` package [Han11], but we will try to be fairly detailed.

Since you are reading this document, you probably have already obtained `ruleR`. The most recent (development) version is available from R-forge at `http://r-forge.r-project.org/projects/ruler/` and can be installed in an interactive R session by typing

```
> install.packages("ruleR", repos="http://R-Forge.R-project.org")
```

More stable versions are available via CRAN. As usual we load the package by

```
> library(ruleR)
```

If you get stuck, you can try

```
> help(ruleR)
```

[2]We thank G. Gediga for this idea.

Some rules for number sequences only need the current element to calculate the next, while others like the Fibonacci rule need the previous two elements. For this reason `ruleR` knows *single rules* and *double rules*. Let's have a look at the `list` of single rules

```
> singleRules

[[1]]
[1] "IdenSingleRule"

[[2]]
[1] "AddConstSingleRule"

[[3]]
[1] "MultConstSingleRule"

[[4]]
[1] "DigSumSingleRule"
```

and double rules

```
> doubleRules

[[1]]
[1] "AddDoubleRule"

[[2]]
[1] "MultDoubleRule"
```

Not that many, but remember we can assemble new rules from these radicals. Before doing that, we define some basic building blocks by

```
> idrule <- new("IdenSingleRule")
> add5 <- new("AddConstSingleRule", constantVal = 5, previousRule = idrule)
> mult2 <- new("MultConstSingleRule", constantVal = 2, previousRule = idrule)
> neg <- new("MultConstSingleRule", constantVal = -1, previousRule = idrule)
> DS <- new("DigSumSingleRule")
```

The `new` function is used to create S4 objects. The digit sum and the identity rules do not need any other information beyond the name of the object, while the multiplication and addition of a constant rule do need a constant value. One must also specify (the trivial) identity rule as the `previousRule` and we will see below how changing this argument results in combinations of single rules. We can calculate the next element with these simple rules:

```
> calculate(add5, 11)

[1] 16

> calculate(mult2, 23)

[1] 46
```

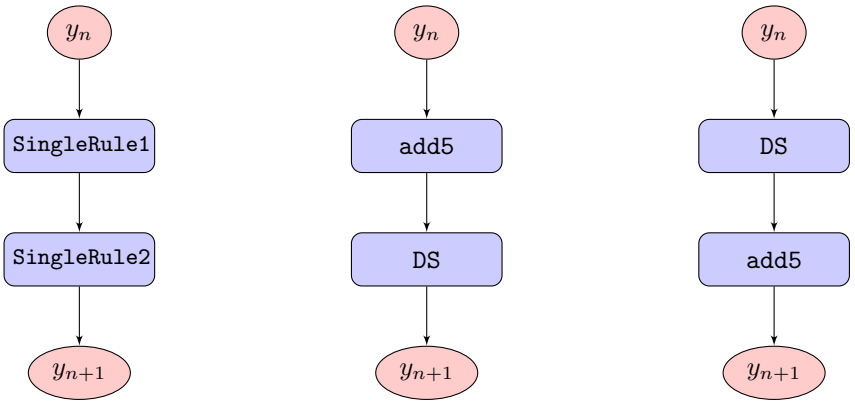
Since only the next element is given by `calculate`, it should not really be used. The `sequenceR` function is much better suited. It outputs a list, whose first component is a number sequence (again as a list) and the second is the rule used. Let's only display the sequence using `unlist`:

```
> unlist(sequenceR(11, add5, 6)[[1]])

[1] 11 16 21 26 31 36

> unlist(sequenceR(2, neg, 6)[[1]])

[1]  2 -2  2 -2  2 -2
```



(a) Combining two (abstract) single rules

(b) Combining `add5` and `DS` this way ...

(c) ... and combining `DS` and `add5` the other way round.

Figure 2: Combining single rules

2.2.1 Combining single rules

Figure 2 shows how single rules are combined. Basically one plugs an existing single rule into another. Clearly the order matters, as the following example using the radicals `add5` and `DS` shows which is also depicted in Figures 2b and 2c:

```
> DSadd5 <- new("AddConstSingleRule", constantVal = 5, previousRule = DS)
> add5DS <- new("DigSumSingleRule", previousRule = add5)
> unlist(sequenceR(11, DSadd5, 6)[[1]])

[1] 11  7 12  8 13  9

> unlist(sequenceR(11, add5DS, 6)[[1]])

[1] 11  7  3  8  4  9
```

In addition to this, `ruleR` also allows the construction of intertwined rules, i.e. two single rules are used alternately to generate a number sequence. This is handled by the class `IntertwinedRule`.

2.2.2 Combinations involving double rules

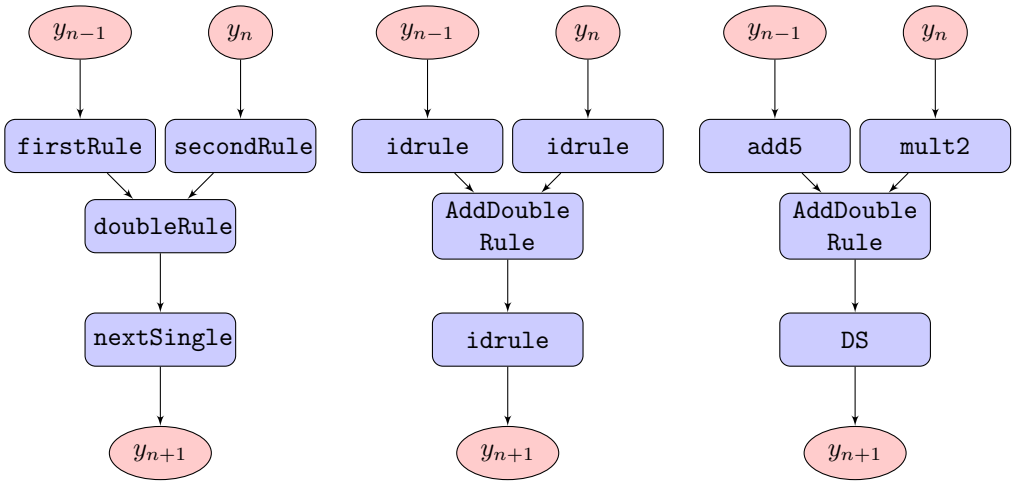
Let's create a Fibonacci rule:

```
> fib <- new("AddDoubleRule", firstRule = idrule, secondRule = idrule,
+            nextSingle = idrule)
> unlist(sequenceR(c(1,1), fib, 6)[[1]])

[1] 1 1 2 3 5 8
```

Note that `sequenceR` does not need an argument of length 2; if we had only specified a single integer it would have been repeated. From the arguments of the call to `new` above one can already guess that single rules are used in three places. Figure 3 displays the abstract pattern and two examples. Note that arbitrary complex single rules can be used in this construction process, so it is very easy to make very complex number sequence items this way. One example using radicals in all three places is:

```
> complex <- new("AddDoubleRule", firstRule = add5, secondRule = mult2,
+            nextSingle = DS)
> unlist(sequenceR(c(1,1), complex, 6)[[1]])
```



(a) `firstRule`, `secondRule` and `nextSingle` arguments (b) Fibonacci rule (object called `fib`) (c) A more complex example (object called `complex`)

Figure 3: Building double rules

```
[1] 1 1 8 4 3 6
```

Single and double rules can be created with the `createSR` and `createDR` functions, which will refer to the `singleRules` and `doubleRules` lists. Changing these lists has to be done with some care!

2.2.3 Advanced features

The package features functions to check sequences (for example `conCheck`, `check` and `duplicate`). Random generation of tests is done via `createTest` which uses a dictionary of rules in the process. The user is able to control the level of nesting (i.e. the number of rules in the dictionary to be combined) as well as the length of the number sequences, starting values, constants and the range of elements. Also the uniqueness of sequences can be analysed with the help of the package, i.e. help is provided to answer the question: Does any other combination of rules generate this sequence?

One last function we would like to demonstrate can come in handy when natural language is needed to describe sequences, for example when providing feedback to examinees. To describe a rule in natural language use `print`; the complex example above then gives

```
> print(complex)

RULES APPLIED TO THE FIRST ELEMENT:

Add 5

RULES APPLIED TO THE SECOND ELEMENT:

Multiply by  2

Add two previous elements

RULES APPLIED TO THE RESULT OF PREVIOUS OPERATIONS:

Take the sum of digits
```

2.3 Extending `ruleR` number sequences

As an example we extend `ruleR` by a modulo single rule. The modulo operation is the remainder of a division. For example 18 modulo 7 is 4 and 15 modulo 12 is 3. If we want to use this as a new single rule, we need a new class which contains `SingleRule`. We need a constant value representing the number by which we divide and we need a `calculateSpecific` method on which everything else is based:

```
> setClass("ModuloSingleRule",
+          contains="SingleRule",
+          representation(constantVal="numeric"),
+          S3methods=TRUE)
> setMethod("calculateSpecific",signature(x="ModuloSingleRule", y="numeric"),
+           function(x,y){
+             return(y%%x@constantVal)
+           })

[1] "calculateSpecific"
```

We can now define a new radical:

```
> mod7 <- new("ModuloSingleRule", constantVal = 7, previousRule = idrule)
```

and use this to construct a single rule:

```
> mult2mod7 <- new("ModuloSingleRule", constantVal = 7, previousRule = mult2)
> add5mod12 <- new("ModuloSingleRule", constantVal = 12, previousRule = add5)
> unlist(sequenceR(1,mult2mod7,6)[[1]])

[1] 1 2 4 1 2 4

> unlist(sequenceR(1,add5mod12,6)[[1]])

[1]  1  6 11  4  9  2
```

A closing remark

The tree structures provided by `ruleR` might seem overly complex at first glance if the sole aim is to generate number sequences. But such tree structures are well-suited for a later cognitive analysis of items and RIG. Also the simple case of number sequences revealed the benefits of using S4 classes already, since for example randomly combining radicals like in `createTest` is very difficult to implement without them. Many test designers are familiar with R these days, so in addition to providing number sequences, `ruleR` can also be seen as proof of concept of an item generator written in R.

References

[Are05] M.˜Arendasy. Automatic generation of Rasch-calibrated items: Figural matrices test GEOM and Endless-Loops Test EC. *International Journal of Testing*, 5:197–224, 2005.

[AS12] M.E. Arendasy and M.˜Sommer. Using automatic item generation to meet the increasing item demands of high-stakes educational and occupational assessment. *Learning and Individual Differences*, 22:112–117, 2012.

[ASGH06] M.˜Arendasy, M.˜Sommer, G.˜Gittler, and A.˜Hergovich. Automatic generation of quantitative reasoning items: A pilot study. *Journal of Individual Differences*, 27:2–14, 2006.

[BK04] F.B. Baker and S.H. Kim. *Item response theory: parameter estimation techniques*. Dekker, 2nd edition, 2004.

[Cha06] John Chambers. How s4 methods work. available on CRAN, 2006.

[CJS90] P.A. Carpenter, M.A. Just, and P.˜Shell. What one intelligence test measures: A theoretical account of the processing in the Raven Progressive Matrices Test. *Psychological Review*, 97(3):404–431, 1990.

[DBW04] P.˜De˜Boeck and M.˜Wilson. *Explanatory item response models: a generalized linear and nonlinear approach*. Springer, 2004.

[ER00] S.E. Embretson and S.P. Reise. *Item response theory for psychologists*. Erlbaum, 2000.

[Fis73] G.H. Fischer. The linear logistic test model as an instrument in educational research. *Acta psychologica*, 37:359–374, 1973.

[GGvdL11] H.˜Geerlings, C.A.W. Glas, and W.J. van˜der Linden. Modeling rule-based item generation. *Psychometrika*, 76:337–35, 2011.

[Han11] Robin K.˜S. Hankin. *Brobdingnag: Very large numbers in R*, 2011. R package version 1.2-1.

[HBKK10] H.˜Holling, H.˜Blank, K.˜Kuchenbacker, and J.T. Kuhn. Rule-based item design of statistical word problems: A review and first implementation. *Psychology Science Quarterly*, 50:363–378, 2010.

[HBZ09] H.˜Holling, J.P. Bertling, and N.˜Zeuch. Automatic item generation of probability word problems. *Studies In Educational Evaluation*, 35:71–76, 2009.

[HRS95] R.K. Hambleton, H.J. Rogers, and H.˜Swaminathan. *Fundamentals of item response theory*. Sage, 1995.

[IK02] S.H. Irvine and P.C. Kyllonen. *Item generation for test development*. Lawrence Erlbaum, 2002.

[JD12] David˜A. James and Saikat DebRoy. *RMySQL: R interface to the MySQL database*, 2012. R package version 0.9-3.

[Kos12] Kosinski, M., Lis, P., Mahalingam, V., Sun, L., Rust, J. Concerto: Open-source Online R-based Adaptive Testing Platform. Software package, 2012. Available at: `http://www.psychometrics.cam.ac.uk`.

[LNB68] F.M. Lord, M.R. Novick, and A.˜Birnbaum. *Statistical theories of mental test scores*. Addison-Wesley, 1968.

[MM11] David Magis and David Magis. *catR: an R package to generate IRT adaptive tests*, 2011. R package version 2.2.

[Rav62] J.C. Raven. *Advanced Progressive Matrices*. London: H.K. Lewis, 1962.

[Riz06] Dimitris Rizopoulos. ltm: An r package for latent variable modelling and item response theory analyses. *Journal of Statistical Software*, 17(5):1–25, 2006.

[VDLG00] W.J. Van Der˜Linden and C.A.W. Glas. *Computerized adaptive testing: Theory and practice*. Springer, 2000.

[vdLG10] W.˜J. van˜der Linden and C.˜A.˜W. Glas. *Elements of Adaptive Testing*. Springer, New York, 2010.

[vdLH97] W.J. van˜der Linden and R.K. Hambleton. *Handbook of modern item response theory*. Springer Verlag, 1997.

[WDF+00] H.˜Wainer, N.J. Dorans, R.˜Flaugher, B.F. Bert˜Green, and R.J. Mislevy. *Computerized Adaptive Testing: A Primer*. Routledge, second edition, 2000.

[Zeu10] N.˜Zeuch. *Rule-based item construction: Analysis with and comparison of linear logistic test models and cognitive diagnostic models with two item types*. PhD thesis, WWU Münster, 2010. Retrieved from: `miami.uni-muenster.de`.