УДК 004.832.32

System Description: Russell - A Logical Framework for Deductive Systems

Vlasov D.Yu. (Sobolev Institute of Mathematics, Novosibirsk State University)

Russell is a logical framework for the specification and implementation of deductive systems. It is a high-level language with respect to Metamath language [7], so inherently it uses a Metamath foundations, i.e. it doesn't rely on any particular formal calculus, but rather is a pure logical framework. The main difference with Metamath is in the proof language and approach to syntax: the proofs have a declarative form, i.e. consist of actual expressions, which are used in proofs, while syntactic grammar rules are separated from the meaningful rules of inference.

Russell is implemented in c++14 and is distributed under GPL v3 license. The repository contains translators from Metamath to Russell and back. Original Metamath theorem base (almost 30 000 theorems) can be translated to Russell, verified, translated back to Metamath and verified with the original Metamath verifier. Russell can be downloaded from the repository `https://github.com/dmitry-vlasov/russell`

Keywords: *logical framework, formal mathematics, deductive system, proof checker*

1. Introduction

Recently the ambitious QED project [2] has celebrated its 20 year anniversary, while the claimed goals of this project are still far from being reached. Several papers [10], [9], [4], [6], [3] addressing the history of QED clearly state that yet there is no computer language, which has all the expected features of a QED system. Summarizing these papers, it can be said that the major barriers of QED are:

- the 'Balcanization' of QED-like systems, i.e. when there are different languages with different foundations and there is no simple way to share formalized proofs between them [6]
- the lack of a powerful automation, which could seriously reduce end user efforts to prove a theorem [4]
- the difference between the standard practical mathematical language, which is used in papers and textbooks, and the formal language of QED-like systems [9]

Russell is another system, which is intended to reach QED goals. It was developed to address all these obstacles, and some of them are to some extent removed in the Russell design.

2. The Russell System

The Russell system is a general purpose framework for definition and usage of different formal deductive systems. The variety of formal systems, which can be represented in Russell, is quite wide, although limited. For example, non-monotonic deductive systems cannot be given in Russell. As a high-level language with respect to Metamath [7], Russell inherits its foundations, which are close to the notion of Post's canonical system [8]. The approach of Metamath to syntax of expressions is more general than that of Russell: syntactic rules in Metamath are indistinguishable from the meaningful rules of inference and can have meaningful (essential, in Metamath terms) premises. In Russell, the grammar of expressions must be context free by the language definition. In all other aspects Metamath and Russell share the same foundations.

2.1. Pure Logical Framework

What distinguishes Metamath and Russell from other logical frameworks is purity: their deduction engines don't use any built-in logic (in the form of axioms and inference rules). The deduction used in Metamath and Russell is concerned with making a proper substitution, applying it to a certain expression and checking the coincidence of the result with some other expression. From the very general point of view, such logic-neutrality is a good property, because we want to avoid the situation when some formal deductive system has an a-priory better fitness to the language than the other because of the propinquity with the underlying logic of a logical framework. Similar arguments are mentioned in the paper [6], where the statement of *foundational pluralism* is given. In fact, the property of logic-neutrality (or pureness) is crucial to the desired foundational pluralism, because it gives a uniform core language for a vast variety of deductive systems. Thus, 'Balkanization' of formal mathematics could be at least based on the same language (although the sharing of proofs between different foundations is still challenging).

What distinguishes Russell from Metamath:

- Expression grammar syntax rules are separated from the general logical inference rules, so the grammar is guaranteed to be context free.
- The Russell proof language is more human-friendly than the Metamath proof language.
- Russell uses a special syntactic construct for definitions, while in Metamath definitions

are simply axioms with labels, starting with 'df' prefix.

2.2. Brief Description and Comparison of Metamath and Russell

At first, let's get familiar with Metamath (the complete description of Metamath syntax and semantics may be found in [7]). Metamath is a very simple language, and the complete list of Metamath keywords is: `$c $v $d $f $e $a $p $= $. ${ $}`

The syntax of expressions in Metamath is not distinguished from the syntax of axioms and inference rules. The latter assertions are treated differently inside proofs, but on the syntax level there is no difference between these two kinds of assertions. This feature makes language simpler, as the same mechanism is used for syntactic inferences and for the logical inference.

For example, the definition of well-formed-formula with $\rightarrow$ and $\neg$ logical connectives in Metamath looks like:

```
$c ( ) -> -. $.
$v ph ps $.
${
    wph $f wff ph $.
    wn $a wff -. ph $.
$}
${
    wph $f wff ph $.
    wps $f wff ps $.
    wi $a wff ( ph -> ps ) $.
$}
```

Here we define constants with `$c <c_1> .. <c_n> $.` construction. Here `->` stands for $\rightarrow$ (implication) and `-.` stands for $\neg$ (negation), and this is how implication and negation are represented in the original Metamath theorem base. All constants are used as a terminal grammar symbols. Note, that brackets are also treated as definable symbols, not pre-defined ones. Variables, symbols which may be substituted with, are also defined with `$v <v_1> .. <v_n> $.` construction. Two constructions with labels `wi` and `wn` are essentially grammar rules with a single non-terminal `wff`, in BNF notation:

```
wff ::= -. wff | ( wff -> wff )
```

The definition of axioms of the Hilbert-style propositional calculus looks like:

```
${
    wph $f wff ph $.
    wps $f wff ps $.
    ax-1 $a |- ( ph -> ( ps -> ph ) ) $.
$}
${
    wph $f wff ph $.
    wps $f wff ps $.
    ax-mp.1 $e |- ph $.
    ax-mp.1 $e |- ( ph -> ps ) $.
    ax-mp $a |- ps $.
$}
${
    a1i.1 $e |- ph $.
    a1i $p |- ( ps -> ph ) $=
        wph wps wph wi a1i.1 wph wps ax-1 ax-mp $.
$}
```

The first two are classical axiom $(\varphi \to (\psi \to \varphi))$ and a rule of inference (modus ponens):

$$\frac{\varphi \quad (\varphi \to \psi)}{\psi}$$

Now let's describe the Metamath syntactic construction in details. Technically the construction `<id> $f <type> <var> $.` is called a 'floating' hypothesis, and means that `<var>` has a type `<type>`. Since Metamath semantics is based on a stack machine, when `id` label is executed, the expression (pair of symbols) `<type> <var>` is pushed on a stack. The `<id> $e <s_1> ... <s_n> $.` construction is called an 'essential' hypothesis, and has a classical meaning of a premise of a proposition. When met in a proof (which essentially is an reverse polish notation (RPN) program for the stack machine), its expression (i.e. sequence of symbols `<s_1>...<s_n>`) is pushed on a stack.

The propositions may be axiomatic or provable, and axiomatic ones are designated as `<id> $a <s_1>...<s_n> $.` Provable propositions are written as a syntactic construction `<id> $p <s_1>...<s_n> $= <l_1>...<l_n> $.` and the sequence of labels `<l_1>...<l_n>` here is a proof of the proposition, i.e. the program for a stack machine, written in RPN form.

The braces `${` and `$}` are used to define a scope of an assertion as a whole, with hypotheses (both, floating and essential) and proposition. When met in a proof, the label of an assertion acts as an operation on the stack: it fetches the appropriate number of expressions from stack, accordingly to its arity (here arity is a sum of the number of floating and essential hypotheses), then it matches floating hypothesis with the corresponding expressions from stack, checks their type and forms a substitution. Then this substitution is applied to the essential hypotheses of the assertion, and proof checker verifies, that the corresponding expressions on the stack are symbol-wise the same. After all these checks, the substitution is applied to the statement of an assertion and the result is pushed on a stack.

What is left to finish the verification of a Metamath proof, when a proof of some provable assertion (i.e. theorem) is executed on a stack, is to ensure, that exactly one expressoin is left on the stack and it coincides with the proposition of the theorem symbol-wise.

As a note on Metamath language, we can imagine, that hypothetically one could add substantial (i.e. essential, in the Metamath terminology) hypotheses to the syntactic grammar rules, but of course, it would make little sense, since the common practice is to use context-free grammars for a language.

Now let's look, how the same rules and axioms look like in Russell syntax:

```
constant { symbol ( ;; }
constant { symbol ) ;; }
constant { symbol -> ;; ascii -> ;; latex \rightarrow ;; }
constant { symbol -. ;; ascii -. ;; latex \lnot ;; }
type wff ;;
rule wn (ph : wff) {
    term : wff = # -. ph ;;
}
rule wi (ph : wff, ps : wff) {
    term : wff = # ( ph -> ps ) ;;
}
axiom ax-1 (ph : wff, ps : wff)  {
    prop 1 : wff = |- ( ph -> ( ps -> ph ) ) ;;
}
axiom ax-mp (ph : wff, ps : wff)  {
    hyp 1 : wff = |- ph;;
    hyp 2 : wff = |- ( ph -> ps );;
    -----------------------
    prop 1 : wff = |- ps;;
}
theorem a1i (x : wff, y : wff)  {
    hyp 1 : wff = |- x ;;
    -----------------------
    prop 1 : wff = |- ( y -> x );;
}
proof of a1i {
    step 1 : wff = ax-1 () |- ( x -> ( y -> x ) );;
    step 2 : wff = ax-mp (hyp 1, step 1) |- ( y -> x );;
    qed prop 1 = step 2 ;;
}
```

At first, the constant symbols are defined, just as in Metamath. Note, that in Russell a symbol may have a unicode representation (in UTF-8 encoding), ascii representation and latex representation. The ascii representation is used for correct translation of Russell sources to Metamath. Latex representation may be used for the latex sources generation, and unicode representation is used in Russell sources in order to represent mathematical symbols as close to the real mathematical practice as possible.

Then we define a type `wff` - a non-terminal symbol of grammar of expressions of propositional logic (in Metamath `wff` is just a constant and is not treated specifically). The next two syntactic constructions are the rules (productions) of a corresponding context-free grammar and are read as: if `ph` and `ps` are of type `wff` (i.e. are well formed formulas), then the expressions `-. ph` and `( ph -> ps )` are also of type `wff`, i.e. are also well formed formulas. Context-freeness here is obligatory by design, because it is impossible to use any kind of hypothesis other then floating (typing) ones. The `term` keyword in the definition of the rule body means, that the following sequence of symbols is just a raw expression, any may not be directly inferable. For example, the expression `term : set = # { a , b }` is a set, and therefore has no truth-value semantics. The keyword `;;` is a symbol sequence terminator, like `$.` in Metamath.

Then there are two axioms: `ax-1` and `ax-mp` (rules of inference are considered a non-zero-ary axiomatic assertions, so are called also 'axioms'). The typing hypotheses here are represented in a common for many programming languages manner: a comma-separated list of variable-type pairs, separated by colon, like `ph : wff, ps : wff`. The proposition of assertion is marked up with `prop` keyword, while hypotheses are marked up by `hyp` keyword and are separated from the propositions (there may be several propositions) by a `--------` keyword (not less then 5 minus symbols), which mimics the separation line in the classical representation of rules of inference as $\frac{H_1,\ldots,H_n}{P}$. The expressions in assertions have a sequence starter symbol $\vdash$ (a turnstile), which means, that these expressions are *logical*, i.e. directly used in inferences.

Then the theorem `a1i` and its proof follows. The syntactic form of a theorem is just the same as the form of an axiomatic assertion, except for the heading keyword `theorem` instead of `axiom`. A theorem must have at least one proof. The proof has a classical linear structure, with explicit links, showing, from which previously proven step or hypothesis the current one follows and by which assertion. Also each step is provided with the appropriate expression, which is essentially what is proved in this step. The `qed` statements shows, which step (usually the last one) is symbol-wise the same as the proposition of a theorem.

So, if we compare the proof languages of Metamath and Russell, we can see, that Russell proofs are much closer to the human mathematical practice and may be understood without any external program tools. Metamath proof is an RPN programm, so the only way to understand the Metamath proof is to compute it as an RPN program, which demands a proof assistant (except for some trivial cases).

Note, that the exact substitution are not explicitly presented in Russell proof, while they are in Metamath. The substitutions for each proof step may be obtained by matching appropriate expressions from the proof with the expressions from an assertion of the step. In the example above, the substitutions for the first and second steps will be $\{x/ph, y/ps\}$ (in real mathematics substitutions may be much more complex then just a variable replacement). Technically, the proof in Russell as a sequence of steps is a stack trace obtained from executing a proof in an RPN form, which is stripped off all intermediate steps related to the syntax formation. When a Russell proof is verified or translated to Metamath, these intermediate steps are restored from the syntax tree of an expression and corresponding matching substitutions.

Thus, the Russell proof language is natural and simple, which imposes minimum restrictions and allows for user-defined grammars for expressions. Together with the possibility to use the conventional set theory it bridges the gap between the computer-based mathematics and the common practice mathematics from textbooks.

2.3. Definitions in Metamath and Russell

One of the most important features of any computer-based deductive system (framework) is safety and reliability [1]:

to what extent one can trust computer proofs ?

Reliability of a formal system is a complex subject, which involves several aspects. One of these aspects is the size of the axiomatic base used by a theory. If it is large then there can be some (unintentionally) hidden inconsistency inside of it, and if so, this will lead to the triviality of the whole theory (in case of an explosive logic). On the other hand, if the set of the true axioms is small and well-known (like some variant of ZF set theory) then its degree of reliability is very high.

If each definition is introduced as a new axiom, like it is done in Metamath, then the number of axioms increases fast as the theory grows, and at some point there is no guarantee that all of these axioms are consistent. To address this issue, definitions in Russell are introduced as a special syntactic construct and certain properties are checked for each definition to ensure that adding the underlying axiom will give a conservative extension of the theory. Conservativity here means that if something can be proved with the help of some definition, it can also be proved without it. This property is strictly proven, so it gives some more certainty about correctness of Russell theories.

Example of a definition in Russell:

```
definition df-or (ph : wff, ps : wff)  {
    defiendum : wff = # ( ph \/ ps ) ;;
    definiens : wff = # ( -. ph -> ps ) ;;
    -----------------------
    prop : wff = |- ( defiendum <-> definiens ) ;;
}
```

Here we define a logical 'or' connective on the basis of negation and implication. Keyword `defiendum` denotes what is defined (the disjunction of two formulas), and `definiens` denotes the extension of the notion 'or' - its definition as a formula. So far as both constructions are terms, and Russell semantics (as well as Metamath semantics) doesn't provide a mechanism for internal term rewriting (since it is a pure inference engine), we cannot just 'substitute' any occurrence of disjunction with the appropriate negation and implication. Instead, we make a new *axiom* out of the definition, which allows us to replace disjunction with its definition indirectly, by common logical rules. The proposition of this axiom is obtained from the `prop` expression of the definition by replacing `defiendum` and `definiens` meta-variables by the corresponding terms. For this particular definition this axiom's proposition will be:

```
( ( ph \/ ps ) <-> ( -. ph -> ps ) )
```

The fact that all Russell sources can be translated back to Metamath and checked with its original proof checker shows that Russell at least as reliable as Metamath. Moreover, the declarative format of proofs in Russell makes it possible for a human to do an independent verification of proofs. Of course, it would look strange to exploit human ability of checking a formal proof in a QED system, but still, from the philosophical point of view, human understanding is an ultimate judge, and is very important.

3. Implementation

Currently the Russell language is implemented as a translator from / to Metamath and is written in c++14. The Russel repository includes test scripts, which run a chain of translations:

```
Metamath -> Russell -> Metamath
```

The translation of the whole Metamath base (about 30 000 theorems) is rather fast in all directions. The most problematic from the performance point of view is expression parsing in Russell. Metamath uses an explicit construction of expressions in proofs, so it does not require any parsing or matching algorithm when checking its source. Unlike Metamath, Russell must parse expressions in order to get syntax trees and such parsing takes the most of time in comparison to all other steps, like matching or translation.

One of the features implemented in the toolchain of translators is that it can automatically divide the original Metamath source (the single file of almost 150 megabytes) into reasonably small parts following the internal layout inside the source file. After breaking it into pieces, one can browse the source file tree with the standard desktop navigation tools and watch source files in the standard desktop editors without the necessity to handle a single 150 Mb file.

Russell is not considered by the author as an experimental or model language. It is supposed to be a useful, universal, and convenient tool for all kinds of activity in the field of formal deduction. To achieve this, the language of implementation (c++), quality of source code, efficiency of algorithms, and usefulness to the end user are of a great importance. Russell implementation should be able to work with hundreds of thousands of assertions in a reasonable time, which is the subject of ongoing research and development.

4. Conclusion and Future Work

The Russell logic framework is a robust, fast, and reliable general purpose tool for representation of formal deductive systems. The Russell language is designed to be simple and easy-to-learn and provides proofs in a declarative form (which is a standard practice in informal mathematical texts).

Essentially, powerful automation is the only one blocking property left in the list of the desirable QED features. Therefore, the next challenge is to implement the automated proving feature so that the process of formal proof design would be easier. The first step to making an automated proof engine for Russell has been already taken and it showed the potential feasibility of such a goal. Since the proof search space suffers from an extreme combinatorial explosion, some extraordinary means are needed to cope with it. Standard techniques will not work here, since the nature of the underlying deductive system is apriori unknown to the prover (which is a consequence of the logic-neutrality property). For example, in general we can not assume, that the underlying logic is cut free (and actually it is not in the case of the Metamath theorem base).

To create a powerful prover for Russell we plan to use advanced machine earning techniques to make the prover use the experience of the already proven theorems. Ideally, it should be able to generate human-like proofs, formed as a combination of previously obtained proofs.

The other important goal is to support importing of other bases of formalized mathematics into Russell. Some successful attempts of importing HOL theorem base into Metamath have been already undertaken [5], so there is a hope that it would possible to join a large part of the already formalized mathematical knowledge under a common logical framework, but with different foundations. A more ambitious goal is to join these bases upon a common foundation.

Список литературы

1. *Adams M.* Proof Auditing Formalized Mathematics, Journal of Formalized Reasoning Vol. 9, No. 1, (2016), pages 3 – 32
2. *Anonymous.* The QED Manifesto, in: A. Bundy (ed.), Automated Deduction - CADE 12, LNAI 814, Springer-Verlag, pages 238 – 251. (1994)

3. *Barendregt H., Wiedijk F.*, The Challenge of Computer Mathematics, Transactions A of the Royal Society 363 no. 1835, pages 2351 – 2375, (2005)
4. *Blanchette J. C., Kaliszyk C., Paulson L. C., Urban J.* Hammering towards QED, Journal of Formalized Reasoning Vol. 9, No. 1, (2016), pages 101 – 148.
5. *Carneiro M.M.* Conversion of HOL Light proofs into Metamath, Journal of Formalized Reasoning Vol. 9, No. 1, (2016), pages 187 – 200.
6. *Kohlhase M., Rabe F.* QED Reloaded: Towards a Pluralistic Formal Library of Mathematical Knowledge, Journal of Formalized Reasoning Vol. 9, No. 1, (2016) pages 201 – 234.
7. *Megill N.* Metamath: A Computer Language for Pure Mathematics, Lulu Press, Morrisville, North Carolina, (2007)
8. *Post E.* Formal Reductions of the General Combinatorial Decision Problem, American Journal of Mathematics 65 (2), pages 197 – 215, (1943).
9. *Wiedijk F.* The QED Manifesto Revisited. In: From Insight to Proof, Festschrift in Honour of Andrzej Trybulec. (2007), pages 121 – 133.
10. *Wiedijk F.* The Seventeen Provers of the World, Volume 3600 of Lecture Notes in Computer Science, (2006) Springer-Verlag.