Pseudo-Random ALC Syntax Generation

Aaron Eberhart, Michelle Cheatham, and Pascal Hitzler

DaSe Lab, Wright State University, Dayton OH 45435, USA,
aaron.eberhart@gmail.com,{michelle.cheatham, pascal.hitzler}@wright.edu

Abstract. We discuss a tool capable of rapidly generating pseudo-random syntactically valid ALC expression trees [1]. The program is meant to allow a researcher to create large sets of independently valid expressions with a minimum of personal bias for experimentation.

Manually providing sufficient unique expressions for testing in development of a semantic reasoning application is both very time-consuming and potentially unscientific. For this reason, we developed an integrated generator for syntactically correct ALC expressions. The random nature of the process precludes the possibility of generating any meaningful semantic information. However, expressions obtained from our tool are more than sufficient to serve as test cases necessary to validate the basic functionality of a reasoning application. In this paper we will describe the problem that the application is designed to solve, briefly touching on some of the logics. Next we discuss some of the features unique to our implementation and the design trade-offs involved. Finally, we will contextualize our work within current and possible future developments, then close with relevant technical information.

Problem Usually the end goal of a reasoning application is to produce a system with semantic capabilities. The testing required to validate such a system is, however, mostly a response to purely syntactical questions. Does the application recognize and handle correct and incorrect forms of input? How do we define a valid input? Is there any significant potential for human-error affecting the results? These are problems that this project address. Devised as a solution to the first question, the expression generator is a way to bypass potential pitfalls in an intricate reasoning process by artificially creating only correct inputs. By restricting our domain to ALC we are able to explicitly define the sorts of expressions that will be allowed in our application. This also provides us with the basic logical framework for how the generator should run and the types of expressions it produces. If the application works properly, it should always only generate valid ALC expressions to feed into the reasoner. Similarly, it is important that, in addition to soundness, the generator be functionally complete over the set of all valid ALC expressions as well. In an empirical setting, this is often the more difficult test so we have endeavored to allow an output range as broad or specific as an experiment requires. Though there is always some degree of human-error in any study, it would be insufficiently rigorous in an experiment to enter in one's own formulas. The tool we present should rapidly generate many random correct expressions and help reduce the researcher's own bias in making test cases.

Logic The expression generator for this project is capable of building two distinct types of expression: one type for the TBox and one type for the ABox. Any TBox expression tree is created by first making an atom and then randomly choosing to expand the expression or end it. Expanding the expression tree involves either conjunction, disjunction, negation, or quantification. Ending a TBox expression creates an unused Concept to avoid circularity with itself.

TBox Expressions	
Operation	Result
New Expression	C, $\exists R.C$, or $\forall R.C$
Negation	$\neg$ {Original}
Conjunction	{Original} $\sqcap$ {New Sub-Expression}
Disjunction	{Original} $\sqcup$ {New Sub-Expression}
For All	$\forall R.${Original}
Exists	$\exists R.${Original}
Subset	$C_{(new)} \sqsubseteq$ {Original}
Equivalent	$C_{(new)} \equiv$ {Original}

For an ABox expression, either a ground atom is produced and the generator finishes, or a TBox style expression is created and then eventually ground so that it becomes an ABox expression. The ABox generator will not return an expression that is not ground.

ABox Expressions	
Operation	Result
New Expression	$C(a)$, $R(a, b)$, or {New TBox Expression}
Negation	$\neg$ {Original}
Conjunction	{Original} $\sqcap$ {New Sub-Expression}
Disjunction	{Original} $\sqcup$ {New Sub-Expression}
For All	$\forall R.${Original}
Exists	$\exists R.${Original}
For All Ground	$\forall R.${Original} : a
Exists Ground	$\exists R.${Original} : a

All predicate and constant variable names are randomly assigned except for the TBox finalizations; a, b, C, and R are only used here for convenience.

Solution A unique specialization of our program is that it separates the expression generation from any normalization concerns. This allows for extremely rapid expression generation as well as the creation of more natural statements. We have included methods that normalize ALC expression trees into NNF so that a normalized copy can be obtained by someone using the program. A new random number generator was also included due to the Java Random class' tendency to create very uniform expressions. Another feature of our program is that it contains constants that limit the types of expressions that can be created and can be tuned for a specific experiment. These control, for each expression created, the number of times the generator is allowed to make a sub-expression, the maximum quantification depth, and the maximum size. If their values are set very high or removed there is a possibility that the generator will get lost in subtree creation and take much longer than desired while building massive expressions. Some fine-tuning may be necessary to obtain the types of expressions needed for a given experiment.

Context Our generator is similar to other methods [2][3] that create expressions of greater expressivity. These strategies require much stronger requirements on the types of formulas produced through artificial structural limitations to produce CNF expressions. By restricting our generator to ALC we are able create a wider range of expressions that we feel is more appropriate for reasoner experimentation, while still operating at high efficiency. Our program has no specific known obstacles to expansion so that it can generate more expressive statements, if that became necessary in the future.

Technical Information This project was written in Java 1.8 with Eclipse Oxygen.1 4.7.1 build 20170914-1200. Testing was primarily performed on an Acer Aspire R5-471T computer with Windows 10 x64 and an Intel Core i5-6200U CPU running at 2.3GHz and 8GB of RAM. Current source code can be found at https://github.com/aaronEberhart/Reason.er.

The generator we describe can be useful in the development and testing of semantic reasoner experiments because it:

- Allows for relatively unbiased expression generation.
- Is optimized for the generation of ALC expressions.
- Enables quick creation of large numbers of formulas in an experiment.
- Creates only valid expressions, ensuring that tests run without problems.
- Produces semantically diverse statements that enable thorough testing.

References

1. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: Introduction to Description Logic. Cambridge University Press (2017)
2. Hladik, J.: A generator for description logic formulas. In: Horrocks, I., Sattler, U., Wolter, F. (eds.) Proceedings of DL 2005. CEUR-WS (2005), available from `ceur-ws.org`
3. Patel-Schneider, P.F., Sebastiani, R.: A new general method to generate random modal formulae for testing decision procedures. CoRR abs/1106.5261 (2011), `http://arxiv.org/abs/1106.5261`