Language Convergence Infrastructure

Vadim Zaytsev, `zaytsev@uni-koblenz.de`
Software Languages Team, Universität Koblenz-Landau, Germany

Overall setup. The methodology for grammar convergence has been presented in [1] and elaborated in a large case study [2]. In short, it is a method of establishing relationships between language grammars by extracting them from available grammar artifacts and transforming until they become equivalent. The relationship is represented by the transformation chain properties: its length, the type of steps it consists of, the correspondence with the properties expected a priori from documentation, etc.

Grammar convergence is a complicated process that can only be automated partially and therefore requires expert knowledge to use successfully. In order to make things as simple as possible for the grammar engineer, we need a solid transformation operators suite and a powerful tool support. The former, called XBGF, where BGF stands for BNF-like Grammar Format—the XML variant that we use to store grammar knowledge, was sketched in [1] and is far too complex and voluminous to be explained in this abstract. Thus, the latter becomes the main topic of our interest.

Overall tool support for grammar convergence has been developed and released as a part of SLPS, Software Language Processing Suite, `slps.sf.net`. It comprises several stand-alone scripts targeting comparison, transformation, benchmarking, validation, extraction—most of those scripts written in Python, Prolog and Shell. The centre that ties it all together is called LCI, or Language Convergence Infrastructure. It operates on a DSL we call LCF (F for Format) in which the input configuration must be expressed.

Core convergence tools. There are three top-level *tools* that are used universally on all grammars: comparison, transformation and validation. *Comparison* tool references an external program that takes two BGF grammars and returns the verdict on their equivalence. Since the premise of grammar convergence method was to document grammar relationships, the comparator is not expected to do any sophisticated matching. *Transformation* tool takes a BGF grammar and an XBGF script and applies the latter on the former, resulting in a transformed BGF grammar (or an error return code). *Validation* is an optional tool that is asked to check the XML validity of every grammar produced in the convergence process.

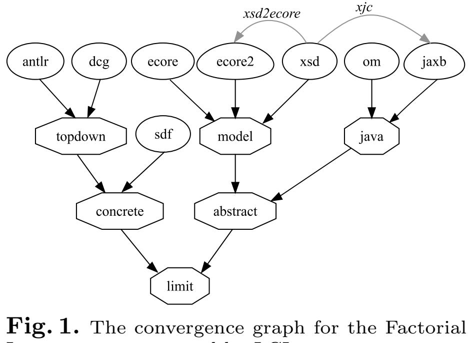

Fig. 1. The convergence graph for the Factorial Language as generated by LCI.

Convergence sources. In LCF one can specify one or more *sources* — the places where new languages are fed into LCI. Each source has a *name*, *grammar* properties, *instance* properties and *testing* properties. There are three kinds of properties for grammars and instances: *extraction, parsing* and *evaluation.* All of them refer to external tools (or tool wrappers) that use a unified interface which allows LCI to execute them, detect abnormal termination and accept the output. The only property that is needed for a source to be valid is grammar extraction, the rest is optional. It happens only once per source even if the source is used more than once. When it succeeds, LCI stores the extracted grammar in order to fall back to the old snapshot if it ever goes wrong in one of the future runs. If there are no instance properties present, no coupled transformations (see below) take place. Testing properties consist of a list of test sets that can be used to find bugs in this source's grammar. Optionally, a source could also include a priori known relationships that are invisible for LCI otherwise (e.g., ECore model generated from XSD can be marked as derived from the XSD source).

Convergence targets. By *target* we mean a node in convergence tree where two or more grammars become equivalent (i.e., they converge). In LCF a target has a *name* and any number of *branches*. A branch has an *input* grammar (be it another target or a source) and a list of transformations, grouped by *phases*. A convergence phase relates to the strategy advised by [2], the notion is used to separate preliminary nominal matching scripts from language-preserving refactorings doing structural matching and from unsafe steps like relaxation, correction or extension. Whenever a script fails, that branch is terminated prematurely, implying that all consecutive transformations will fail. For all branches that reach the target, their results are compared pairwise to all others. If all branches fail or the comparator reports a mismatch, the target fails.

Grammar transformation. Any step is either bound to an XBGF file or relates to a generator. This is not necessarily a one-to-one relation, in Java case study some scripts were designed so universally that they were re-used several times for different sources. Transformation *generators* are external tools that take a BGF grammar as an input and produce an XBGF script applicable to that grammar and containing transformations of a certain nature. For example, one of the generators used for FL case study in [1] could make a script that stripped all terminal symbols from any grammar—this was needed to converge concrete syntax of a language with abstract syntax of the same language. Generators are defined on top-level just as transformation or comparison tools, so that they can be applied in different places. LCI is prepared for a generator to fail or to produce inapplicable scripts.

Coupled transformations. It is possible to implement all transformation operators to be applicable not only to grammars, but also to instances (parse trees). If this is done and the corresponding instance extractors and parsers are provided in LCF, then LCI is not limited to converging grammars only. One can specify one or more *test sets* to be extracted from somewhere or perhaps just copied. For every source that has a test set attached, for every test case in that set, LCI performs coupled extraction, transformation and comparison. Additionally, *evaluators* can be provided that can execute test cases and compare return values with expected ones. Test sets must be present in a unified format for LCI to figure out applicable actions. Test cases will also be validated if the validation tool is specified.

Language documentation. Grammars can be extracted from language documents, test sets can also be formed from the samples present in the standards. We also have designed a transformation suite for language documentation, so it is possible to converge languages as such triples (text, grammar, samples).

Conclusion. Language Convergence Format allows to express the domain concepts of grammar convergence for further automation, allowing a language engineer to take on convergence scenarios of considerable size. Test cases can complement the method by using coupled transformations. Language documents can be transformed, too, to support language evolution or surface relationships between language specifications.

References.

1. R. Lämmel, V. Zaytsev. An Introduction to Grammar Convergence. In *iFM'09*, *LNCS* 5423, pages 246–260. Springer, 2009.
2. R. Lämmel, V. Zaytsev. Recovering Grammar Relationships for the Java Language Specification. In *SCAM'09*, 2009. To appear.