Quantitative Generalizations of Languages*

Thomas A. Henzinger

EPFL, Switzerland, and University of California, Berkeley

In the traditional view, a language is a set of words, i.e., a function from words to boolean values. We call this view "qualitative," because each word either belongs to or does not belong to a language. Let Σ be an alphabet, and let us consider infinite words over Σ. Formally, a *qualitative language* over Σ is a function A: $\Sigma^\omega \to \mathbb{B}$. There are many applications of qualitative languages. For example, qualitative languages are used to specify the legal behaviors of systems, and zero-sum objectives of games played on graphs. In the former case, each behavior of a system is either legal or illegal; in the latter case, each outcome of a game is either winning or losing. For defining languages, it is convenient to use finite acceptors (or generators). In particular, qualitative languages are often defined using finite-state machines (so-called ω-automata) whose transitions are labeled by letters from Σ. For example, the states of an ω-automaton may represent states of a system, and the transition labels may represent atomic observables of a behavior. There is a rich and well-studied theory of finite-state acceptors of qualitative languages, namely, the theory of the ω-regular languages.

There are two common, orthogonal quantitative generalizations of languages. In the first quantitative view, a language is a set of probability spaces on words. We call this view "probabilistic." A *probabilistic word* over the alphabet Σ is a probability space on Σ^ω. We write $\mathcal{D}(\Sigma^\omega)$ for the set of probabilistic words. A *probabilistic language* over Σ is a function B: $\mathcal{D}(\Sigma^\omega) \to \mathbb{B}$. Probabilistic languages can be defined by Markov decision processes (MDPs) whose transitions are labeled by letters from Σ. MDPs generalize ω-automata by distinguishing between controllable states, where an outgoing transition is chosen according to a policy[1] (or strategy), and probabilistic states, where an outgoing transition is chosen according to a given probability distribution. Given an MDP, and a policy for resolving all controllable decisions, the outcome is a probabilistic word. By collecting the outcomes of all policies in a set, we obtain a probabilistic language. Many basic questions about such finite-state generators of probabilistic languages are unsolved. For example, the language-inclusion problem for MDPs is central to the algorithmic verification of probabilistic systems: it asks, given two finite-state MDPs M_1 and M_2, if the probabilistic language defined by M_1 is a subset of the probabilistic language defined by M_2; in other words, if for every policy p_1 of M_1, there is a policy p_2 of M_2 such that the outcome of applying p_1

* This research was supported in part by the Swiss National Science Foundation and by the NSF grant CCR-0225610.

[1] Policies may in general be probabilistic. A *policy* is a function mapping each finite state sequence (representing the history of a behavior) to a probability distribution on the possible next states.

in M_1 is equal to the outcome of applying p_2 in M_2. To our knowledge, it is an open problem if this question can be decided.

In the second quantitative view, a language is a function from words to real values. These values may represent rewards or costs. We call this view "numerical." Formally, a *numerical language* over the alphabet Σ is a function C: $\Sigma^\omega \to \mathbb{R}$. We refer to $C(w)$ as the *value* of a word w in the language C. There are several ways of generating numerical languages. One mechanism for obtaining finite numerical values for infinite words is discounting, which gives geometrically less weight to letters that occur later in a word. Let M be a state machine whose transitions are labeled by letters from Σ. Given a real-valued discount factor $\lambda \in (0, 1)$, the value $M(w)$ of each word $w \in \Sigma^\omega$ can be defined as $1 - \lambda^n$, where n is the number of letters in the longest prefix of w that is accepted by M (if all prefixes of w are accepted by M, then $M(w) = 1$). Numerical languages are also generated by weighted state machines, whose transitions are labeled both with letters from Σ and with real values. The numerical label (or weight) of a transition may represent a reward obtained or a cost incurred by traversing the transition. Let M be a weighted state machine, and let r be a run of M over a word $w \in \Sigma^\omega$. The run r can be defined to assign to w either the supremal transition value occurring in r, or the limsup of all transition values in r, or their limit average, or their discounted sum (for some discount factor λ). The weighted state machine M, then, assigns to each word w as value the supremum of all values assigned to w by accepting runs of M over w. In game theory, objectives that try to maximize a numerical value are common and well-studied; in system modeling, the numerical value of a run may represent a resource requirement of a behavior, such as power consumption.

The probabilistic and numerical views can be combined, resulting in quantitative languages of the type $[\mathcal{D}(\Sigma^\omega) \to \mathbb{R}]$. In this talk, we survey some theoretical results about such quantitative generalizations of languages, and review some of their applications in system design and verification [1–6].

References

1. A. Chakrabarti, K. Chatterjee, T.A. Henzinger, O. Kupferman, and R. Majumdar. Verifying quantitative properties using bound functions. In *CHARME: Correct Hardware Design and Verification Methods*, Lecture Notes in Computer Science 3725, pages 50–64. Springer, 2005.
2. A. Chakrabarti, L. de Alfaro, T.A. Henzinger, and M. Stoelinga. Resource interfaces. In *EMSOFT: Embedded Software*, Lecture Notes in Computer Science 2855, pages 117–133. Springer, 2003.
3. K. Chatterjee, L. de Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, and M. Stoelinga. Compositional quantitative reasoning. In *Proceedings of the Third Annual Conference on Quantitative Evaluation of Systems*. IEEE Computer Society Press, 2006.
4. K. Chatterjee and T.A. Henzinger. Value iteration. In *25 Years of Model Checking*, Lecture Notes in Computer Science. Springer, 2007.

5. L. de Alfaro, T.A. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In *CONCUR: Concurrency Theory*, Lecture Notes in Computer Science 2154, pages 351–365. Springer, 2001.
6. L. de Alfaro, T.A. Henzinger, and R. Majumdar. Discounting the future in systems theory. In *ICALP: Automata, Languages, and Programming*, Lecture Notes in Computer Science 2719, pages 1022–1037. Springer, 2003.