# Evolving Reactive Agents with SignalGP

Alexander Lalejini and Charles Ofria

Department of Computer Science and BEACON Center for the Study of Evolution in Action,
Michigan State University, East Lansing, MI, USA
lalejini@msu.edu

## Introduction

We introduce SignalGP, a technique for creating digital organisms that harnesses the event-driven programming paradigm. These organisms can evolve to automatically react to signals from the environment or from other agents in a biologically-inspired manner. In addition to introducing SignalGP, we summarize previous results demonstrating the value of the event-driven paradigm in environments dominated by agent-agent and agent-environment interaction. Our full introduction to SignalGP can be found in (Lalejini and Ofria, 2018).

Digital evolution has its roots in Genetic Programming (GP), wherein computer programs are evolved using natural principles. For example, Avida (Ofria et al., 2009) is a popular digital evolution system that uses self-replicating linear genetic programs as its organisms. The organisms generally follow an imperative programming paradigm where computation is driven procedurally. Program execution starts at the top of the program and proceeds in sequence, instruction-by-instruction, jumping or branching as dictated by executed instructions (McDermott and O'Reilly, 2015).

In contrast to imperative programming, program execution in event-driven computing is directed primarily by signals (events), simplifying the development of programs that dynamically react to events around them. This biologically-realistic mode of execution is often employed when developing software for agents that must frequently interact with each other or the environment (such as robotics or distributed systems). By capturing the event-driven paradigm, SignalGP aims to improve our capacity to evolve computer programs that operate in interaction-heavy environments, expanding our ability to generate complex agent-agent and agent-environment interactions.

## SignalGP

In SignalGP, signals direct computation by triggering the execution of program modules (functions). Here, we present SignalGP in the context of linear GP, wherein programs are represented as linear sequences of instructions; however, the ideas underpinning SignalGP generalize across a variety of digital evolution systems.

SignalGP agents (programs) are defined by a set of functions. Each function contains a linear sequence of instructions and is referred to using a tag. SignalGP *events* contain a tag and event-specific data. Agents react to events by running matched functions that specify how that event should be handled. SignalGP augments tag-based referencing techniques demonstrated by Spector *et al.* (Spector et al., 2011) to determine which function is triggered by an event. Here, tags are arbitrarily represented as fixed-length bit strings. An event triggers the function with the most similar tag where tag similarity is the proportion of matching bits between the two bit strings. Agents may generate internal events and are subject to events generated by the environment or by other agents. When an event triggers a function, the function is run with the event's associated data as input, allowing agents to react on-the-fly to signals; SignalGP agents can react to many events simultaneously by processing them in parallel. Each SignalGP program instruction has a tag argument (which it may or may not use), providing an evolvable mechanism for instructions to reference internal functions or to generate events. Mutations in SignalGP can alter tags or instruction content within functions, as well as duplicating or deleting whole functions. As function tags and instructions evolve, their relationship with events and each other can be refined over time. Figure 1 gives a high-level overview of SignalGP. For a full description of our implementation of SignalGP, see (Lalejini and Ofria, 2018).

## Experimental Results

In (Lalejini and Ofria, 2018), we demonstrated the value of incorporating the event-driven programming paradigm in GP using two distinct test problems: a changing environment problem and a distributed leader-election problem. In both problems, a program's capacity to react efficiently to external events is crucial. Here, we briefly report a subset of our results for the changing environment problem.

In the changing environment problem, the environment can be in one of $K$ states. To maximize fitness, agents must match their internal state to the current state of the environ-

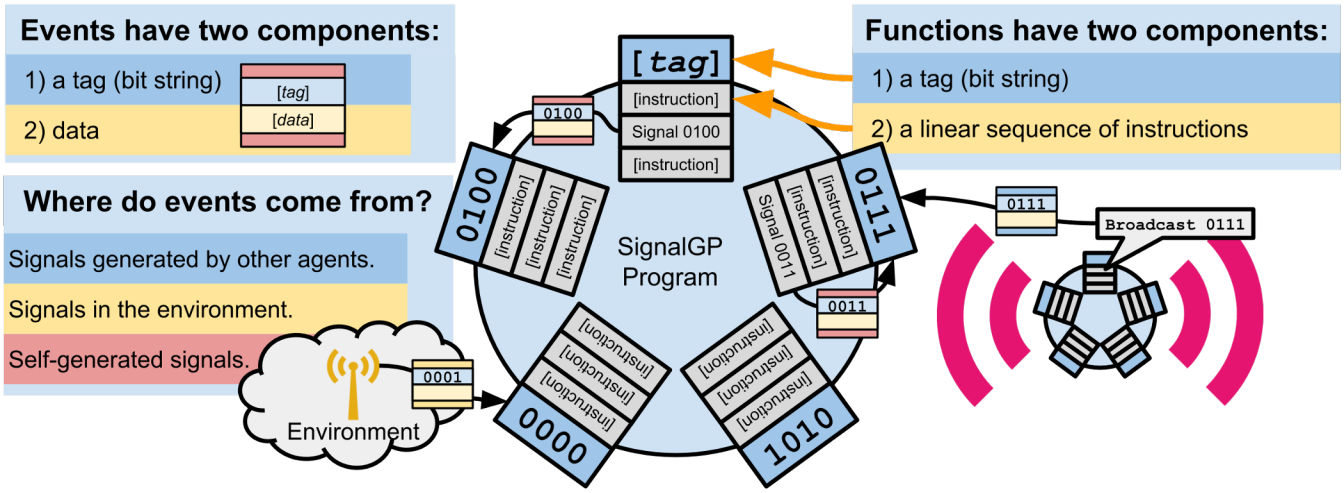


Figure 1: **A high-level overview of SignalGP.** SignalGP programs are defined by a set of functions. Events trigger functions with the *closest matching* tag. SignalGP agents can respond to many events simultaneously by processing them in parallel.

ment. We evolved SignalGP agents under three treatments, each with different mechanisms to sense the environment: (1) an event-driven treatment where environmental change events produced signals that can trigger functions; (2) an imperative control treatment where programs had to actively poll the environment to determine its state; and (3) a combined treatment where agents had either option available. In the imperative and combined treatments, we included new instructions that allowed programs to test the current environmental state. Here, we show the results for environment sizes two and eight ($K = 2$ and $8$) in Figure 2. We compared treatments using a Kruskal-Wallis test, and if significant ($p < 0.05$), we performed a post-hoc Dunn's test, applying a Bonferroni correction for multiple comparisons.

Agents evolved with fully event-driven SignalGP significantly outperformed those evolved in the imperative treatment across both the two-state (combined: $p = 1.21\text{e-}47$; event-driven: $p = 1.21\text{e-}47$) and eight-state environments (combined: $p = 1.29\text{e-}46$; event-driven: $p = 2.18\text{e-}45$). In the combined treatment, we further confirmed that evolution favored the event-driven strategy (Lalejini and Ofria, 2018).

## Conclusion

While our recent work demonstrates SignalGP in the context of linear GP, we have plans to extend SignalGP across a variety of evolutionary computation systems. Here, functions are exclusively represented as linear sequences of instructions; however, we can easily use any representation capable of processing inputs (*e.g.* other forms of GP, neural networks, *etc.*). We could even employ multiple representations within a single agent, providing evolution with a diverse toolbox and allowing digital organisms to comprise mosaics of representations.

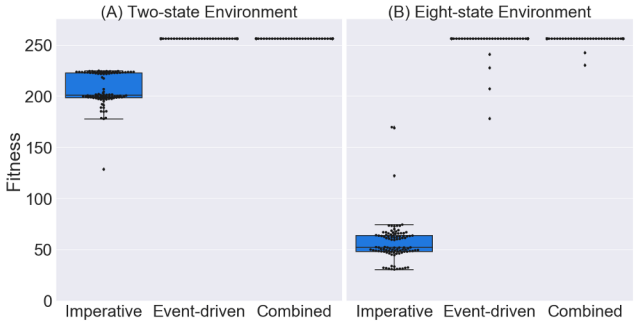


Figure 2: **Changing environment problem results** across the (A) two-state environment and (B) eight-state environment. The box plots indicate the fitnesses (each an average over 100 trials) of best performing agents from each replicate.

## Acknowledgements

We extend our thanks to Wolfgang Banzhaf and the members of the Digital Evolution Laboratory at Michigan State University for thoughtful discussions and feedback. This research has been supported by the National Science Foundation under Grants DGE-1424871 and DEB-1655715.

## References

Lalejini, A. and Ofria, C. (2018). Evolving Event-driven Programs with SignalGP. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM.

McDermott, J. and O'Reilly, U.-m. (2015). Genetic Programming. In Kacprzyk, J. and Pedrycz, W., editors, *Springer Handbook of Computational Intelligence*. Springer, Berlin, Heidelberg.

Ofria, C., Bryson, D. M., and Wilke, C. O. (2009). Avida: A software platform for research in computational evolutionary biology. In *Artificial Life Models in Software*. Springer London.

Spector, L., Martin, B., Harrington, K., and Helmuth, T. (2011). Tag-based modules in genetic programming. *Proceedings of the Genetic and Evolutionary Computation Conference*.