This is the published version of a paper presented at *Nicograph International 2014, Visby, Sweden, May 2014*.

Citation for the original published paper:

Fridenfalk, M. (2014)
A Simplified Representation of Finite State Machines by Generic States.
In: *Nicograph International 2014* (pp. 163-164).

N.B. When citing this work, cite the original published paper.

Permanent link to this version:
http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-235501
A Simplified Representation of Finite State Machines by Generic States

Mikael Fridenfalk
Department of Game Design
Uppsala University
Visby, Sweden
mikael.fridenfalk@speldesign.uu.se

Abstract—Wildcards are of significant value for the simplification of state diagrams and truth tables for the representation of finite state machines. In this paper a new type of wildcard is introduced, called a generic state, which without any loss of information may further simplify such representations.

Keywords— finite state machine; minimization; sequential automata; simplification; state diagram; truth table; wildcard

I. INTRODUCTION

The Finite State Machine (FSM) is a universal tool that finds applications in many fields from analog and digital electronics [3,5], and artificial intelligence (AI) [8], to biotechnology [4]. In the development of computer games, FSMs are often used for the implementation of non-player characters [2,9], and in network programming. An FSM, a special case of a sequential machine [1], or automata [7], can be fully specified by a state diagram or a truth table [6]. An FSM may in general be described as a system with a limited number of states, where transitions between states are defined by logical conditions or rules that are evaluated at each state update. These logical conditions are based on the input values of the FSM as a function of the current state.

Commonly, in discrete-time systems, the current state may be denoted as \( z \) and next state as \( z + 1 \). The output values of an FSM may be defined as a function of the current state, or more generically as a combined function of the current state and the input values. The representation of an FSM is however in general implementation-neutral in the sense that no matter in which order the conditions are evaluated, the result is expected to be the same. In the general case, on an implementation level in computer programming, the transitional conditions for FSMs are often evaluated sequentially, why an FSM can be implemented in many different ways (without altered behavior).

There are many applications where an FSM can be designed graphically. An example is Stateflow [11], which in addition, by the assistance of the Simulink Coder [10], is able to convert the Stateflow diagram directly to code in C or C++, for compilation and execution on e.g., an embedded system.

II. WILDCARDS

Wildcards are presently used for the representation of FSMs to decrease the size of state diagrams and truth tables [7]. As an example, Table I shows a truth table for an FSM with originally 40 lines (3 Boolean inputs yields \( 2^3 \) lines for each state, multiplied by 5 states), here reduced to 18 lines. In digital electronics, an uppercase X is often used to denote a wildcard. A wildcard is in this specific context a symbol that may be substituted for 0 or 1 without any consequences. Since a wildcard is often denoted by an asterisk (*) in computer programming (such as in the Unix operating system), the same notation has also been used in this paper.

Wildcards are however, as a rule, typically only used to simplify the transitional conditions in the representation of an
FSM by a truth table. The use of wildcards in state diagrams is nominally implicit, as reflected by the example in Fig. 1, corresponding to Table II, where the loop transitions from each state to itself (i.e. no change), included in Table I, have been omitted. Such simplifications can be performed without information loss, but only if it is clear from the context that any transitional conditions that have not been defined, refer to a loop from a state to itself.

This paper presents a proposal to extend the use of wildcards from input values solely, to states. The idea is that a generic state could similarly be denoted with an asterisk to represent any arbitrary state of the FSM.

As shown in Fig. 2, using a generic state, while the number of states was increased by one, the number of transitions was in this example reduced by over 50%. Such transitions are common in the design of FSMs for AI-systems in computer games (for instance to reset a non-player character to its original state or in general any state that is a destination for a multiple number of states).

IV. CONCLUSION

The new proposal using a generic (or wildcard) state seems to work well for the simplification of state diagrams and truth tables. Since the type of transitions that a generic state is able to simplify is common in practical applications, this method could potentially contribute to streamline the representation of FSMs, particularly for systems of high complexity.

REFERENCES