Direction-Reversible Self-Timed Cellular Automata for Delay-Insensitive Circuits

DANIEL MORRISON, IREK ULIDOWSKI

Department of Computer Science, University of Leicester, England

Received XX December XX; In final form X April XXXX

We introduce a new Self-Timed Cellular Automaton capable of simulating reversible delay-insensitive circuits. In addition to a number of reversibility and determinism properties, our STCA exhibits direction-reversibility, where reversing the direction of a signal and running a circuit forwards is equivalent to running the circuit in reverse. We define also several extensions of the STCA which allow us to realise three larger classes of delay-insensitive circuits, including parallel circuits. We then show which of the reversibility, determinism and direction-reversibility properties hold for these classes of circuits.

Key words: delay-insensitive circuit, reversibility, direction-reversibility, asynchronous cellular automata, computational universality

1 INTRODUCTION

Delay-insensitive (DI) circuits are a category of asynchronous circuits which make no assumption about delays within modules and lines (wires) connecting the modules, and have no global clock. It is argued in [11] that typical logical gates such as NAND and XOR are not Turing-complete when operated in a DI environment. Therefore implementations of DI modules and circuits in alternative technologies, such as cellular automata ([6]) and RSFQ circuits ([20]), have been researched actively in recent years. DI circuits were introduced by Keller ([3]) who characterised the conditions required for correct DI operation and gave various universal sets of modules for a large class of circuits. Subsequent work by Patra and Fussell ([19]) went into finding more efficient universal sets of modules for this class of circuits.

Reversible modules were originally studied by Fredkin and Toffoli ([2]) who proposed a number of synchronous universal logic gates. More recently, Morita, Lee, Peper and Adachi carried out research into finding efficient universal sets of reversible *serial* DI modules (where only one signal travels around a circuit) with memory, such as *Rotary Element* ([12]), and *Reading Toggle* and *Inverse Reading Toggle* ([9]). The sets of all possible 2-state reversible serial modules with two, three and four pairs of input/output lines were enumerated in [14]. Implementations of reversible serial DI modules in *Self-Timed Cellular Automata* (STCAs) ([21]), a special type of asynchronous cellular automata, are shown in several papers including [10, 9, 8, 7]. How these various concepts relate to each other was discussed by Morita in [13]. Further investigations in [16, 17] examine the effects of combining reversibility with parallelism in the context of DI modules and the resulting limitations of such modules, while introducing a new notation for describing DI modules called *set notation*.

In this paper we introduce four novel STCAs for non-arbitrating parallel DI circuits, including two STCAs for the reversible versions of such circuits. Our two main STCAs have several very useful properties, they are *locally deterministic*, *locally reversible* and support what we call *direction-reversibility*. This allows us to operate a circuit in reverse by changing the direction of signals and utilising its output lines as input lines (and vice versa). This removes the need for separate constructions to realise the inverse of a circuit. New notions of global determinism and global reversibility are introduced for parallel DI circuits, and these properties are proven for these two STCA. We also introduce two further extensions to the STCAs which simulate irreversible circuits. These two additional STCAs are shown to be locally deterministic and globally deterministic. Finally, we prove that the final STCA can be used to realise Keller's class of DI circuits.

Section 2 introduces DI modules and some universality results and Section 3 introduces STCAs. In Section 4 we define two new STCAs for the simulation of reversible serial and serial modules. We then show how to extend our STCAs to support two subclasses of parallel modules and give a universality result for Keller's class of DI circuits. Section 6 concludes the paper. All constructions in this paper were verified using an STCA simulator. The software, as well as the constructions in this paper, can be found at [15].

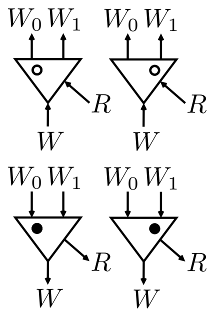
2 DELAY-INSENSITIVE MODULES

We define delay-insensitive modules using a high-level *set notation* developed in [17] which naturally describes external concurrent behaviour of modules. The notation represents a module as a mapping between the module's states together with sets of concurrent input signals and the resulting states with output signals. This is in contrast with the sequential machine style notation of Keller ([3]) where modules are simple state machines which process single inputs at a time. Details how set notation is derived from the sequential machine notation can be found in [17].

A module is a 4-tuple $\{Q, I, O, T\}$, where Q is a finite set of states ranged over with q, $q' \ldots$ or $S, S' \ldots$, I is the set of input lines and O is the set of output lines both ranged over with a, $b \ldots$. The map $T \subseteq Q \times (P(I) \setminus \emptyset) \rightarrow Q \times (P(O) \setminus \emptyset)$ is called the *transition map* and it assigns an input set in a given state to an output set and a new state. Informally, T denotes what we call the input/output behaviour of a module. This describes the effects of a concurrent set of input signals to a module. The environment is required to signal the set of lines corresponding to one defined input set. Upon receiving the full input set, the module produces signals on lines corresponding to the output set and changes the state, according to T. The environment may then begin signalling a new input set. If an input set is undefined in a given state, it is assumed that this input set is never signalled in the given state. We note that Keller's sequential machine notation uses a transition map which is a *partial function*, whereas our notation is more general by also allowing maps which are not functions; see also Remark 4.

The *inverse* of $\{Q, I, O, T\}$ is the module $\{Q, O, I, T^{-1}\}$ where T^{-1} is the inverse of T. Note that since T is a map, T^{-1} is defined. We call a module *serial* if all input and output sets are singletons, and no two input sets are equal in any given state. Modules which are not serial are called *parallel*. We call a serial module *reversible* if T is a bijection, otherwise it is *irreversible*.

A *network* or *circuit* of modules is a collection of instances of modules, such that every output of a module is connected to at most one input of another module and every input of a module is connected to at most one output of another module. We say that a network of modules is *delay-insensitive* (or DI for short) if the network operates correctly regardless of delays in any of the lines or modules. We call a network a *realisation* of a module if the input/output behaviour of a network simulates correctly the input/output behaviour of the module. A network is serial if all modules within the network



$$0 = (W, W_0).1$$
$$1 = (W, W_1).0 + (R, W_1).1$$

$$0 = (W_1, W).1$$
$$1 = (W_1, R).1 + (W_0, W).0$$

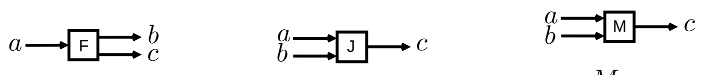
FIGURE 1
Reading Toggle (top) and *Inverse Reading Toggle* (bottom) modules. A signal on R (W_0) is not permitted when RT (IRT) is in state 0. This definition of *RT* and *IRT* is from [8].

are serial, otherwise it is parallel. We say that a set of modules is *universal* for a class of modules, if any module within the class can be realised using only modules from the set.

As in [17], we use a CCS-like ([23, 25]) notation to present more easily the definitions of modules. A typical module (Q, I, O, T) is given by a set of equations of the form $q_i = (A_{i1}, B_{i1}).q_{i1} + \cdots + (A_{ij}, B_{ij}).q_{ij}$ where $i, j \geq 0$. The expression $(A_{kl}, B_{kl}).q_k$ is called an *action* of q_k for all appropriate k, l, $A_{kl} \subseteq I$ and $B_{kl} \subseteq O$. There is one such equation for each q_k in Q, and, for all appropriate k and l, $(A_{kl}, B_{kl}).q_{kl}$ is an action of q_k if and only if $(q_k, A_{kl}, q_{kl}, B_{kl}) \in T$. Informally, each action corresponds to an element of T and vice versa. For example, $S_0 = (\{a, b, c\}, \{d, e\}).S_1$ says that in state S_0, the environment may signal the lines a, b and c exactly once each, in any order or concurrently. This will result in a signal on d, a signal on e (in any order or concurrently), and a new state S_1. Set brackets are omitted for singleton sets.

Figure 1 shows the modules *Reading Toggle* (*RT*) and *Inverse Reading Toggle* (*IRT*) ([9, 8]), which are both reversible serial, and are each other's inverses. We use $0, 1$ as states and R, W, W_0, W_1 as inputs and outputs to be consistent with the notation in [8]. The set $\{$*RT, IRT*$\}$ is universal for reversible computation ([9], [8]), and a network of *RT* and *IRT* modules can realise any reversible serial module. Hence $\{$*RT, IRT*$\}$ is universal for the class of reversible serial modules.

Figure 2 shows several other common DI modules. For consistency with



$F = (a, \{b, c\}).F$ $J = (\{a, b\}, c).J$ $M = (a, c).M + (b, c).M$

FIGURE 2
Fork, *Join* and *Merge* modules.

the original definitions, F, J and M denote states. *Fork* and *Join* are parallel modules. *Merge* is a serial module and is irreversible, as its transition map is not a bijection.

Definition 1. A module is *arbitrating* (*arb* for short) if there is a state with different actions $(A, B).q'$ and $(A', B').q''$ such that either $A \subseteq A'$ or $A' \subseteq A$. A module is *non-arbitrating* (*non-arb*) if it is not arbitrating.

Informally, arbitration corresponds to a form of non-determinism. As each possible input set in a given state corresponds to a set of signals arriving, for a module to be deterministic in a delay-insensitive environment, no input set can be a subset of another input set in the same state, and no input set can lead to two different output sets or different states. All modules defined so far are non-arb. An example of a simple arb module is $D = (\{a, b, c\}, x).D + (\{a, b\}, y).D$. Here, the input set $\{a, b\}$ is a subset of the input set $\{a, b, c\}$ in the state D. It is clear that all serial modules are non-arb due to the requirement that no two input sets in a state are equal.

Example 2. Keller's *Arbitrating Test and Set* module (*ATS* for short) [3] is an example of an arb module. The set notation definition of *ATS* (as given in [17]) is as follows:

$$\begin{aligned} S_1 &= (T, T_1).S_1 + (\{R, T\}, T_1).S_0 + (\{R, T\}, T_0).S_1 \\ S_0 &= (T, T_0).S_1 \end{aligned}$$

ATS is a two-state module, which can hold one of two memory values (referred to as 0 and 1), and allows the arrival of signals on T and R concurrently. If T is processed, the module outputs a signal on T_1 or T_0, corresponding to the held value, and resets the held value of the module to 1. If R is processed, the module sets the held value to 0 but produces no output, so this is not visible to the environment. Hence the held value does not directly correspond to the state of the module as defined by the set notation. Therefore, signalling R

and T concurrently in S_1 may result in different behaviours, which is reflected in the (set notation) definition of S_1.

Definition 3. A module is *backwards-arbitrating* (*b-arb* for short) if there are two states q_1 and q_2 in the definition of the module such that $(A, B).q_1'$ and $(A', B').q_2'$ are actions of q_1 and q_2, respectively, and $q_1' = q_2'$ and either $B \subseteq B'$ or $B' \subseteq B$. A module is *non-backwards-arbitrating* (*non-b-arb*) if it is not b-arb.

Many b-arb modules are not logically reversible because the inverses of their transitions maps are not bijections. There are, however, logically reversible b-arb modules, whose inverses are not forwards-deterministic in a DI environment due to the presence of inclusion between input sets in the same state. Consider the module defined by the equation $S_0 = (\{a\}, \{x, y\}).S_0 + (\{b\}, \{x\}).S_0$. The map T given by $(S_0, a, S_0, \{x, y\})$ and (S_0, b, S_0, x) is clearly a bijection. However, the inverse of this module, given by the equation $S_0^{-1} = (\{x, y\}, \{a\}).S_0^{-1} + (\{x\}, \{b\}).S_0^{-1}$ is clearly non-deterministic when delay-insensitivity is assumed, as signalling the set $\{x, y\}$ concurrently could produce a or it could result in b and a signal pending on y.

There is a mismatch between the notion of logical reversibility when considering a module's definition, and the notion of reversibility in a DI environment. Therefore, in the context of parallel modules, we shall use the term *reversible* to mean strictly non-arb non-b-arb modules, and not simply modules where T is bijective. *Fork* and *Join* are examples of reversible modules which are each others' logical inverses, as inverting the definition of one yields the other. *Merge* is an example of a b-arb module: two actions $(a, c).M$ and $(b, c).M$ share the same output set $\{c\}$, as well as the same target state M.

Remark 4. We note that set notation is more expressive than the sequential machine style of Keller, and as a result it is possible to define modules using this notation which cannot be specified as traditional sequential machines. An example is the module *Choice* which implements a binary choice:

$$C = (c, a).C + (c, b).C$$

Note that *Choice* is the inverse of *Merge*. *Choice* cannot be specified as Keller's sequential machine because its transition map is *not* a function. This distinction is not relevant when studying non-arb modules.

In [16], we defined a *Distributed (one-bit) Memory* (*DM*) module which is reversible and serial, and showed that $\{$*DM, Merge*$\}$ is universal for the class of serial modules. As $\{$*RT, IRT*$\}$ is universal for any reversible serial module, it can realise *DM*. It is shown in [17], by giving a general construction, that the set $\{$*DM, Fork, Join*$\}$ is universal for the class of non-arb non-b-arb modules and $\{$*DM, Fork, Join, Merge*$\}$ is universal for the class of non-arb modules. It is shown in [3] that the set $\{$*Merge, Fork, Select, ATS*$\}$ is universal for Keller's class of DI modules, where *Select* is a simple serial module with a one-bit memory [3].

Proposition 5. 1. $\{$*RT, IRT, Merge*$\}$ is universal for serial modules.
2. $\{$*RT, IRT, Fork, Join*$\}$ is universal for non-arb non-b-arb modules.
3. $\{$*RT, IRT, Fork, Join, Merge*$\}$ is universal for non-arb modules.
4. $\{$*Merge, Fork, RT, IRT, ATS*$\}$ is universal for Keller's class of DI modules.

Proof. Universality of the two non-arb sets of modules is achieved through the illustration of a general construction, one for each of the two classes of modules, given in [17]. It is shown in [9] that $\{$*RT, IRT*$\}$ is universal for any reversible serial module, so it is capable of simulating *DM*. As *Select* is serial, it can be realised using the set $\{$*RT, IRT, Merge*$\}$. $\square$

3 SELF-TIMED CELLULAR AUTOMATA

A Self-Timed Cellular Automaton (STCA for short), introduced in [21], is a special type of asynchronous cellular automaton. It is given by a set of *update rules* together with a two-dimensional infinite array of *cells*. In Figure 3, adopted from [8], a cell is depicted as a square (for example, the square containing triangles with a, b, c, d). Each cell is divided into four subcells which are depicted in Figure 3 as small triangles, each of which can be in one of two states, 0 or 1. We depict the state 0 with a clear triangle, and the state 1 with a black triangle. The default state of a subcell is 0 and is known as the *quiescent* state. The state of all cells and their subcells in the two-dimensional array is known as a *configuration*, and the state of cells and subcells in the initial array is called the *initial configuration*. Configurations are ranged over by $C, C' \dots$ and $D, D' \dots$. In this paper we identify an STCA with the set of its update rules R.

A set of subcells in a configuration may be involved in an *update*, where the states of subcells are modified according to one of the *update rules*. An update involves a full cell (comprised of its four subcells) together with the cell's four adjacent neighbouring subcells on the two-dimensional plane. Figure 3 shows how a general update rule is depicted (image adopted from [8]),

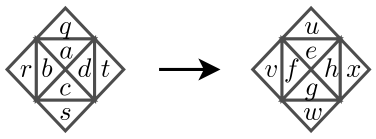


FIGURE 3
Depiction of an update rule.

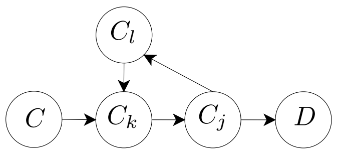
where $a, b, c, d, e, f, g, h, q, r, s, t, u, v, w, x \in \{0, 1\}$. This means that subcells a, b, c, d, q, r, s, t of a configuration are updated to e, f, g, h, u, v, w, x, respectively, giving a new configuration. Following [21], an update of a set of subcells may only occur if an update rule is defined for the current state of the given subcells.

Subcells are assumed to update instantaneously and randomly at any time if a corresponding update rule is defined. However, as two adjacent cells share subcells in their update codomain, we assume, following [22], that no two adjacent cells may update simultaneously.

Definition 6. A set of update rules is *locally reversible*, if no two update rules have identical right-hand sides. A set of update rules is *locally deterministic* if no two update rules have identical left-hand sides. An update causes the current configuration to change to a new configuration. An *execution* of a configuration C is a sequence of configurations $C \to C' \to C'' \ldots$, where $\to$ represents that one or more updates have occurred simultaneously. The reflexive and transitive closure of $\to$ is denoted by $\to^*$. Configuration C' is *reachable* from C if $C \to^* C'$, and we call C' a *derivative* of C.

In this paper we assume that configurations give rise only to those executions that satisfy weak fairness ([4]):

Definition 7. An execution $C_1 \to C_2 \to \ldots$, with $C = C_1$, is *weakly fair* whenever if
• there are different C_k, C_l in the execution with $k < l$ such that $C_{l+1} = C_k$ (a "loop" containing C_k and C_l is reachable from C) and
• there is $D \neq C_i$, for $k \leq i \leq l$, such that $C_j \to D$, for some $k \leq j \leq l$, (D is not one of the configurations of this loop and the execution can leave the loop by updating to D),
then $C_m = D$ for some $m > l$ (the execution leaves the loop eventually).



$$\begin{tikzcd} & C_l \arrow[d] & & \\ C \arrow[r] & C_k \arrow[r] & C_j \arrow[ul] \arrow[r] & D \end{tikzcd}$$

FIGURE 4
A weak-fair execution. Each node represents a unique configuration.

Informally, once an execution reaches a loop and if it is possible to break from the loop by updating to a configuration outside the loop, then the configuration will be reached eventually. An example of this can be seen in the execution graph in Figure 4, where each node represents a unique configuration. Weak fairness allows us to achieve the effect of Keller's ([3]) *finite-blocking* property which requires that signals are assimilated eventually by modules. It also allows us to guarantee that a module eventually produces an output in response to an input.

Next, we define two new important properties of STCAs.

Definition 8. Let C be a configuration of an STCA.
1. C is *globally deterministic* if there exist a configuration D such that, for all C', if $C \to^* C'$, then $C' \to^* D$.
2. C is *globally reversible* if there exist a configuration D' such that, for all C', if $C' \to^* C$, then $D' \to^* C'$.

Informally, if C is globally deterministic then all executions from C must eventually reach the configuration D required by Definition 8. Correspondingly, if C is globally reversible then all executions to C originate from some configuration D' as required in Definition 8. This is a modification of global reversibility defined in [8], and is made here to accommodate for parallel signals travelling through an STCA and looping execution sequences.

Remark 9. There is some relationship between the global determinism and global reversibility of configurations in Definition 8 and the *Forward Diamond* (FD) and *Reverse Diamond* (RD) properties of labelled transition systems (LTS for short) in [23, 24]. If we defined the notions of Definition 8 in the setting of LTSs, then FD would imply general determinism and RD would imply global reversibility, but not vice versa.

For the purpose of this paper, we assume that the configurations are finite

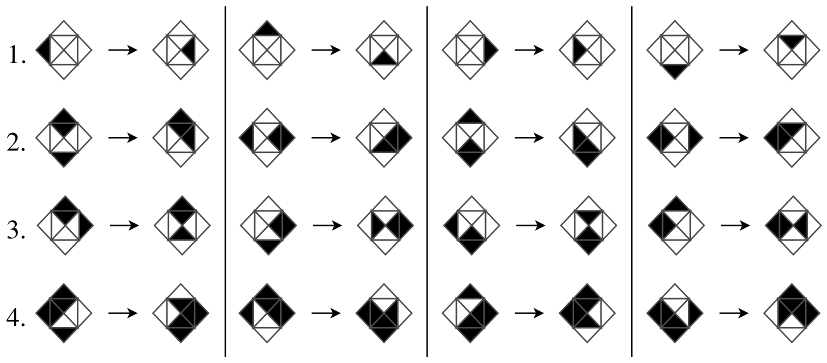


FIGURE 5
The set of rules *RS* for reversible serial modules. Each class 1-4 consists of a single rule (on the left) together with three of its rotations (on the right). The classes represent the following behaviours: 1) Signal movement; 2) Signal right turn; 3) Signal left turn (direction-reversal of class 2); 4) Toggle of a *memory structure* by a signal.

two-dimensional arrays such that the four "edges" of the grid are rows of cells which are not involved in updates.

Examples of STCAs developed for the simulation of reversible serial DI circuits can be found in [10, 9, 8, 7]. The STCAs in [10, 8, 7] contain locally reversible and locally deterministic rules.

4 DIRECTION-REVERSIBLE STCA FOR SERIAL DI MODULES

In this section we define a new STCA intended for the simulation of serial DI modules, including reversible modules.

In Figure 5, we give four different rules along with three rotations by the multiples of 90 degrees. Hence each line in Figure 5 represents an equivalence class of rules. In the context of DI circuits, a signal is represented by a single subcell in state 1, with the subcell adjacent to its longest side in the quiescent state. Signals are considered to "point" in the direction perpendicular to the subcell's longest side. We refer to this set of rules, (and the STCA defined by this set) as RS (for reversible serial).

Definition 10. Given an update rule r, $\delta(r)$ is the update rule obtained from r by (a) *inverting* r, namely swapping the left and right-hand sides, and then (b)

inverting the direction of the signal in the resulting rule, namely swapping the states of subcells inside the squares given by the following pairs (q, a), (c, s), (r, b), (d, t), (u, e), (g, w), (v, f), (h, x) in Figure 3. The rule $\delta(r)$ is called the *direction-reversal* (rule) of r and, given the set of rules R, $\delta(R)$ is the set of direction-reversal rules of the rules in R. STCA with the set of rules R is *direction-reversible* if $R = \delta(R)$.

For example, the direction-reversal of rule 2 in Figure 5 is rule 3, namely $\delta(2) = 3$, and vice-versa. Also, $\delta(1)$ is the third rule in class 1 and $\delta(4)$ is the third rule in class 4.

Proposition 11. Let $r \in RS$. Then $\delta(r) \in RS$ and, hence $\delta(RS) = RS$.

Proof. By Definition 10. □

Proposition 11 implies that for each construction in *RS* which performs an operation on a single signal, inverting the direction of the signal in the rules has the same effect as using the inverses of rules. Moreover, we have:

Proposition 12. Rules in *RS* are locally reversible and locally deterministic.

Proof. By Definition 6. □

Figure 6 shows a single construction which acts as either *RT* or *IRT*, depending which lines are used as inputs. This is a consequence of the direction-reversibility of *RS*. We say that such constructions, as well as configurations realising such constructions, are *direction-reversible*. This implies that *RS* can be used to perform universal computation, and to simulate any reversible serial module (see Section 1). Furthermore, as both *RT* and *IRT* are simulated by a single construction separate constructions are not required to realise the inverse of a reversible serial circuit. Inverting a circuit in *RS* requires simply changing the direction of the signal. This bidirectional nature implies a potential advantage when considering physical implementation.

Proposition 13. Any reversible serial module can be realised by a configuration in *RS*. Such configurations, and their derivatives, are globally deterministic, globally reversible, and direction-reversible.

Proof. It is shown in [9] that any reversible serial module can be simulated using a network of *RT/IRT* modules. Composing multiple instances of the construction in Figure 6 allows a realisation of such a network. Local reversibility and local determinism of the rules, together with the presence of a single signal in the network, results in a unique execution sequence,

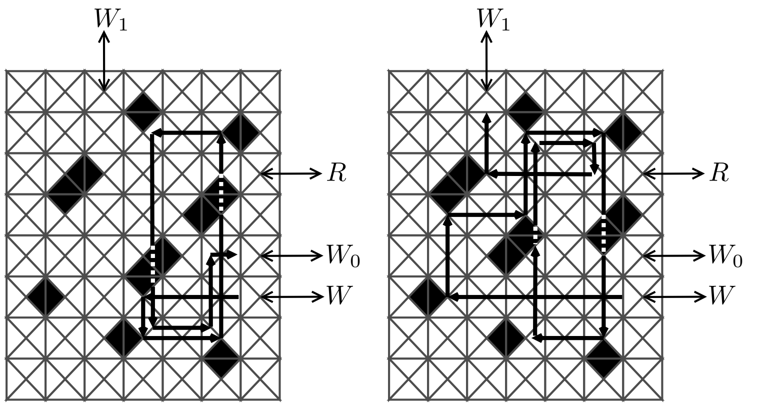


FIGURE 6
Left: *RT/IRT* in state 0. Right: *RT/IRT* in state 1. When R and W are used as inputs, the construction acts as *RT* with W_0 and W_1 as outputs, and vice-versa for *IRT*. The images show the path of the signal W when the construction is used as *RT*. A dotted white line through a memory structure indicates that the memory's state is toggled and the signal continues in the same direction.

thus guaranteeing global reversibility and global determinism. Direction-reversibility of *RT*/*IRT*, local reversibility and local determinism guarantee direction-reversibility. □

Our STCA *RS* uses only four classes of rotation-symmetric rules. The only STCA for simulating reversible serial circuits which utilises four classes of rotation-symmetric rules that we are aware of appears in [10, 7], but direction-reversibility is not supported. The STCA in [9] supports direction-reversibility, but the rules are not locally reversible or locally deterministic. We are not aware of any other STCA which allows direction-reversibility. Furthermore, the STCA in [9] requires five classes of rotation-symmetric and reflective-symmetric rules.

We now demonstrate a useful construction in *RS*. In [1] it is shown that any irreversible function (which we call I) of the type *Input* $\rightarrow$ *Output* can be converted to a reversible function which simulates I. In order to be left with a garbage-less result, this requires a complex series of operations A, B and C (see Figure 7), where: A is a reversible version of the original irreversible function I which performs the operation of I while recording the computation

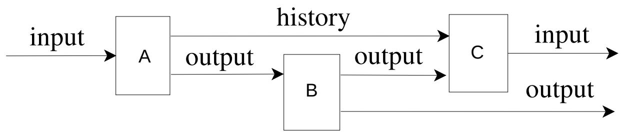


FIGURE 7
Reversible garbage-less implementation of an irreversible function I. A is the reversible version of I, B clones the output of A and C is the inverse of A.

history; B is a function which copies the output (excluding the computation history); and C is the inverse of A which removes the computation history and the first copy of the output, while reproducing the original input. This results in a new function of the type *Input* $\rightarrow$ (*Input* $+$ *Output*) which simulates I.

In order to facilitate implementations of these functions, in Figure 8 we show a construction called *Direction-Reverser* (*DR* for short) which takes an input signal on *IO*, toggles the internal state, and then produces an output on *IO*. Hence, this construction can be used to reverse the direction of a signal. If *DR*s are placed at the output lines of a circuit as in Figure 9, due to the direction-reversible nature of *RS*, an input to the circuit will eventually result in the output being "recorded" in one of these constructions, and the signal being returned to the original input line (but facing the opposite direction). The internal state of the circuit is also returned to its original configuration. Hence, when implementing reversible versions of irreversible functions in *RS*, it suffices to implement only the function A.

In Figure 10, we give the set of rules M for realising *Merge*. Informally, the new rules extend the left/right-turn structure so that a signal approaching from the previously unused side is (irreversibly) forwarded to the opposite side. Hence the left/right turn structure can now operate as *Merge*. The rules in M are locally deterministic but not locally reversible. This gives rise to our second STCA S $=$ *RS* $\cup M$. The set *S* (for serial) supports the reversible constructions demonstrated in this section. Interestingly, direction-reversibility is maintained for all of these constructions. However, attempting to perform a direction-reversal on constructions which utilise *Merge* (which are irreversible) may result in unexpected behaviours.

Theorem 14. Rules in *S* are locally deterministic. Any serial module can be realised by a configuration in *S*. Such configurations, and their derivatives, are

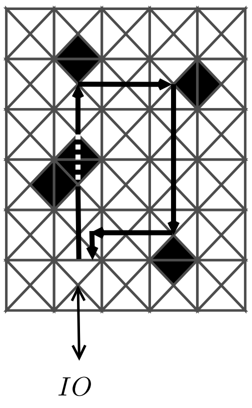


FIGURE 8
A *Direction-Reverser*. An input signal on *IO* causes a change in the state of the memory structure (with a white dotted line) and an output on *IO*.

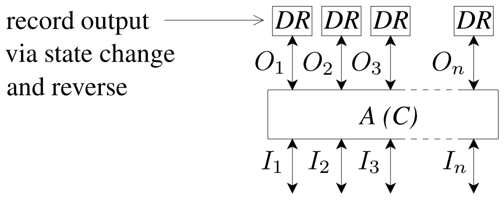


FIGURE 9
Direction-reversible STCA implementation of a reversible version of an irreversible function in Figure 7. The main part of the circuit realises both A and C.

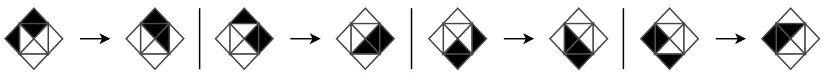


FIGURE 10
The set of rules M. The rotation-symmetric equivalences are included for ease of implementation, but are not required if turns (classes 2 and 3 in *RS*) are utilised.

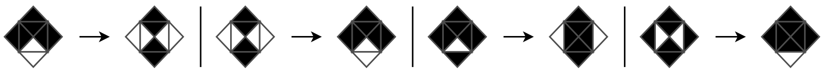


FIGURE 11
The set of rules P. The rules represent from left to right: (p1) *Fork* to *Join* evolution; (p2) *Join* to *Fork* evolution; (p3) *Fork* to *Join* evolution while an input produces two outputs; (p4) *Join* to *Fork* evolution while two inputs produce an output. In the third rule (p3), an input signal (the bottommost black subcell in the source of the rule) arrives at a *Fork*, produces two outputs (the leftmost and the rightmost black subcells in the target of the rule) and changes the configuration to a *Join*. In the last rule (p4), two input signals (the leftmost and the rightmost black subcells in the source of the rule) arrive at a *Join*, produce an output (the bottommost black subcell in the target of the rule) and change the configuration to a *Fork*.

globally deterministic. Configurations which realise reversible serial modules, and their derivatives, are direction-reversible.

Proof. By Proposition 5, $\{$*RT, IRT, Merge*$\}$ is universal for the class of serial modules. Composing multiple instances of the construction in Figure 6 and *Merge* structures allows us to realise any serial module. Local determinism of the rules, together with the presence of a single signal, results in a unique execution sequence, thus guaranteeing global determinism. The behaviour of the *RT*/*IRT* construction is unaffected by the additional rules in M. Hence, by Proposition 13, compositions are direction-reversible. □

5 EXTENDING TO PARALLEL DI MODULES

We now show how *RS* and *S* can be extended to cover parallel circuits. Recall that $\{$*RT, IRT, Fork, Join*$\}$ is universal for the class of non-arb non-b-arb modules (Proposition 5). Hence it suffices to add the rules for *Fork* and *Join*: these are given in Figure 11.

In order to maintain local reversibility and local determinism, rotation equivalent rules in P are not permitted. So, there is a design constraint on the layout of such circuits, and it implies that *Fork* and *Join* constructions must be oriented appropriately in order to function correctly. This is overcome by using left/right turn constructions when designing circuits. We note that the direction-reversal version of each rule in P is also in P, thus $\delta(P) = P$.

Figure 12 shows a *Fork* and *Join* construction in P. As *Fork* and *Join* are each other's inverses, they can be realised with a single direction-reversible

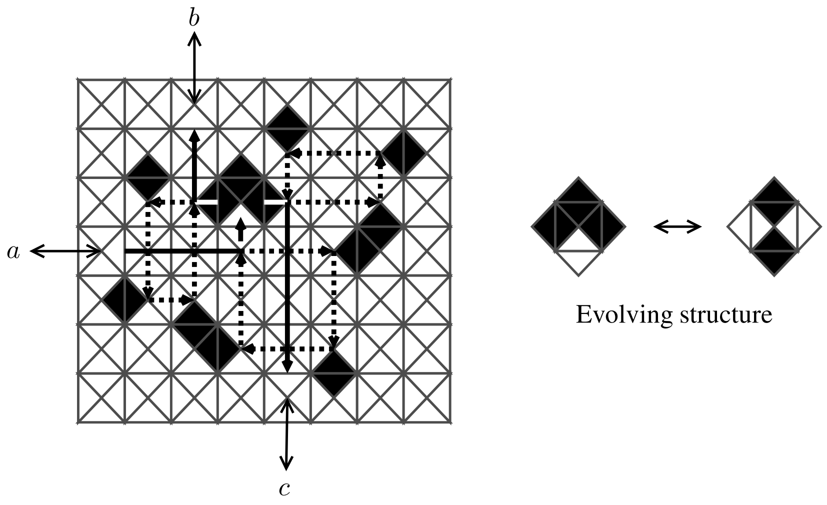


FIGURE 12
Fork and *Join*. The central structure evolves between *Fork* and *Join* patterns according to p1 and p2 in Figure 11. When a (correspondingly b, c) is used as an input, the construction acts as *Fork* (*Join*) with b and c (correspondingly a) as outputs. The image shows the path of signals when used as a *Fork*. Signals loop along the dotted lines until they encounter the evolving structure in the correct state. This will eventually happen due to the weak fairness assumption in Section 3.

construction. This is achieved by two simple structures that evolve constantly, one into the other. The central structure is an *evolving structure* (as opposed to previously seen *static structures*), as updates can occur continuously even when no signals are present. It can be verified that the construction is globally deterministic and globally reversible when signals are applied as intended.

Figure 13 contains rules C for crossing of signals. Note that $\delta(C) = C$.

We now define our third STCA *NANBP* $=$ *RS* $\cup P \cup C$, (where NANBP is for non-arb non-b-arb parallel). Since the construction for *Fork/Join* in Figure 12 is direction-reversible, globally deterministic and globally reversible when operated with appropriately placed signals, it is easy to see that when combining the construction with that of *RT/IRT* and connecting lines appropriately as in DI circuits, the resulting configuration is also direction-reversible, globally deterministic and globally reversible.

Theorem 15. Rules in *NANBP* are locally deterministic and locally reversible.

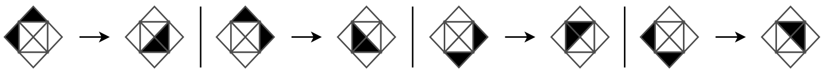


FIGURE 13
The set of rules C (for crossing) to implement the crossing of signals.

Any non-arb non-b-arb module can be realised by a configuration in *NANBP*. Such configurations, and their derivatives, are globally deterministic, globally reversible and direction-reversible.

Proof. By Proposition 5, $\{$*RT, IRT, Fork, Join*$\}$ is universal for the class of non-arb non-b-arb modules. Composing multiple instances of the constructions in Figures 6 and 12 allows a realisation of such a network. Global determinism and global reversibility of both types of construction, together with the delay-insensitive property of the general construction in [17], guarantee that such a network is also globally deterministic and globally reversible. Direction-reversibility, globally determinism and globally reversibility of *RT*/*IRT* and *Fork*/*Join*, together with local reversibility and local determinism guarantee direction-reversibility. □

We note the STCA in [22] implements Keller's full class of parallel DI modules, but rules are not locally reversible or permit direction-reversibility, even when restricted to the subclass of reversible modules. We also note that the *Partitioned Cellular Automaton* (PCA) [5] is similar to a STCA but it has a smaller codomain for the update function, and three states per subcell. This PCA realises Keller's full class of parallel DI circuits, but as in [22] its rules are not local reversible or permit direction-reversibility when restricting to the reversible subclass. And, we note the ACA in [26] that simulates the universal NAND gate via the use of parallel arbitrating DI modules.

Finally, we define STCA *NAP* $= RS \cup P \cup C \cup M$ (for non-arbitrating parallel modules) and, correspondingly to Theorem 14, we have:

Theorem 16. Rules in *NAP* are locally deterministic. Any non-arb module can be realised by a configuration in *NAP*. Such configurations, and their derivatives, are globally deterministic. Configurations which realise non-arb non-b-arb modules, and their derivatives, are direction-reversible.

Proof. By Proposition 5, $\{$*RT, IRT, Fork, Join, Merge*$\}$ is universal for the class of non-arb modules. Composing multiple instances of the constructions in Figure 6, Figure 12 and *Merge* structures using the rules in Figure 10 allows a realisation of such a network. Global determinism of these constructions, together with the delay-insensitive property of the general construction in [17], guarantee that such a network is globally deterministic. The behaviour of the *RT*/*IRT* and *Fork*/*Join* constructions are unaffected by the additional rules in M. Hence, by Theorem 15, compositions are direction-reversible. □

We introduce one further universality result. Recall that the set of DI modules which can be defined using Keller's sequential notation is a strict subset of all DI modules definable using our set notation (Remark 4). Recall from Proposition 5 that $\{$*Merge, Fork, Select, ATS*$\}$ is universal for Keller's class of DI modules.

In Figure 14, we show how the STCA *NAP* can realise *ATS* (defined in Section 2) directly. The construction contains two consistent memory structures which can hold one of two values, 0 or 1 (represented by the grey and corresponding black subcells). A signal on R may only arrive when the memory structures hold a value of 1. In this case it can be verified that the signal sets the two memory structures to 0 and pends at the *Join* structure to the right of the construction. A signal on T will query the value in the upper memory structure. A value of 1 will cause the signal to leave on the $T1$ line. A value of 0 will cause the signal to enter the *Join* structure, synchronising with the pending R signal, before resetting the values to 1 and outputting on $T0$. A race condition may arise between signals on T attempting to query the upper memory, and R attempting to modify the value. A collision will not occur as it is assumed in Section 3 that no two adjacent cells may update simultaneously. It can be verified using [15] that this construction correctly implements the behaviour of *ATS*. This gives us the following result.

Theorem 17. Any module in Keller's class of DI modules can be realised by a configuration in *NAP*.

Proof. By Proposition 5, $\{$*Merge, Fork, RT, IRT, ATS*$\}$ is universal for Keller's class of DI modules. By Theorem 16, *NAP* is able to realise *Merge*, *Fork*, *RT* and *IRT*. Figure 14 shows how to realise *ATS*, also with *NAP*. □

6 CONCLUSION

In this paper we have introduced a new STCA for reversible serial DI circuits. It is locally reversible and locally deterministic. The STCA allows circuits to

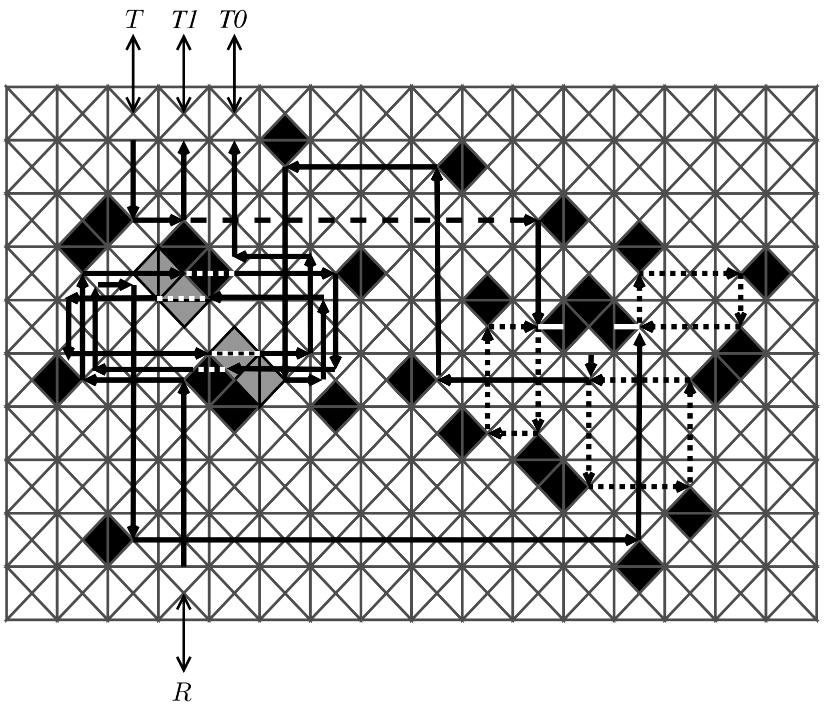


FIGURE 14
Implementation of Keller's *ATS* module. The two memory structures from the left of the figure hold a value of 1 (depicted by the black subcells), or a value of 0 (depicted by the grey subcells). The black and grey subcells both denote the subcell state of 1 but are used to depict the differing location of the memory structure depending on the held value. A white dotted line indicates that a signal toggles the memory structure and then continues in the same direction.

be inverted by reversing the direction of the signal: the new property we call direction-reversibility. We have discussed the potential advantages of this property, and shown how it can be used to realise efficiently garbage-less reversible functions. New notions of global determinism and global reversibility have also been introduced for asynchronous parallel circuits. Our STCA has been shown to be globally deterministic and globally reversible.

We have shown how to extend the STCA to all serial modules, non-arb non-b-arb modules, and finally to all non-arb modules while retaining local determinism and global determinism. We have shown that in the case of non-arb non-b-arb modules, global determinism, global reversibility and direction-reversibility are preserved. Finally, we have proven that the STCA for non-arb modules can be used to realise Keller's class of DI circuits.

7 ACKNOWLEDGEMENTS

The authors would like to thank the reviewers of this paper as well as ACRI 2014 reviewers for their useful comments and suggestions. They also acknowledge partial support of COST Action IC1405 Reversible Computation - extending horizons of computing.

REFERENCES

[1] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, 1973.

[2] E. F. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.

[3] R. M. Keller. Towards a theory of universal speed-independent modules. *IEEE Transactions on Computers*, 23(1):21–33, 1974.

[4] M. Z. Kwiatkowska. Defining process fairness for non-interleaving concurrency. In *Proc. of FSTTCS'90, LNCS 472*, pages 286–300. Springer, 1990.

[5] J. Lee, S. Adachi, and F. Peper. A partitioned cellular automaton approach for efficient implementation of asynchronous circuits. *Computer Journal*, 54(7):1211–1220, 2011.

[6] J. Lee, S. Adachi, F. Peper, and K. Morita. Embedding universal delay-insensitive circuits in asynchronous cellular spaces. *Fundamenta Informaticae*, 58(3-4):295–320, 2003.

[7] J. Lee, S. Adachi, Y. Xia and Q. Zhu. Emergence of universal global behavior from reversible local transitions in asynchronous systems. *Information Sciences*, 282:38-56, 2014.

[8] J. Lee, X. Huang, and Q. Zhu. Embedding simple reversed-twin elements into self-timed reversible cellular automata. *Journal of Convergence Information Technology*, 6(1):49–54, 2011.

[9] J. Lee, F. Peper, S. Adachi, and K. Morita. An asynchronous cellular automaton implementing 2-state 2-input 2-output reversed-twin reversible elements. In *Procs. of ACRI 2008, LNCS 5191*, pages 67–76. Springer, 2008.

[10] J. Lee, F. Peper, S. Adachi, K. Morita, and S. Mashiko. Reversible computation in asynchronous cellular automata. In *Procs. of UMC, LNCS 2509*, pages 220–229. Springer, 2002.

[11] A. J. Martin. The limitations to delay-insensitivity in asynchronous circuits. In *Procs. of AUSCRIPT'90*, pages 263–278. MIT Press, 1990.

[12] K. Morita. A simple universal logic element and cellular automata for reversible computing. In *Proc. of MCU 2001, LNCS 2055*, pages 102–113. Springer, 2001.

[13] K. Morita. Reversible computing systems, logic circuits, and cellular automata. In *Proc. of ICNC 2012*, pages 1–8. IEEE Computer Society, 2012.

[14] K. Morita, T. Ogiro, K. Tanaka, and H. Kato. Classification and universality of reversible logic elements with one-bit memory. In *Proc. of MCU 2004, LNCS 3354*, pages 245–256. Springer, 2004.

[15] D. Morrison. A STCA Simulator, http://www.cs.le.ac.uk/people/dm181, 2014.

[16] D. Morrison and I. Ulidowski. Reversible delay-insensitive distributed memory modules. In *Procs. of RC 2013, LNCS 7948*, pages 11–24. Springer, 2013.

[17] D. Morrison and I. Ulidowski. Arbitration and reversibility of parallel delay-insensitive modules. In *Procs. of RC 2014, LNCS 8507*, pages 67–81. Springer, 2014.

[18] D. Morrison and I. Ulidowski. Direction-Reversible Self-Timed Cellular Automata for Delay-Insensitive Circuits. In *Procs. of ACRI 2014, LNCS 8751*, pages 367–377. Springer, 2014.

[19] P. Patra and D. S. Fussell. Efficient building blocks for delay insensitive circuits. In *Procs. of Async'94*, pages 196–205. Society Press, 1994.

[20] P. Patra and D. S. Fussell. Efficient delay-insensitive RSFQ circuits. In *Procs. of ICCD'96*, pages 413–418. IEEE Computer Society, 1996.

[21] F. Peper, T. Isokawa, N. Kouda, and N. Matsui. Self-timed cellular automata and their computational ability. *Future Generation Computer Systems*, 18(7):893–904, 2002.

[22] F. Peper, J. Lee, S. Adachi, and S. Mashiko. Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers? *Nanotechnology*, 14(4):469, 2003.

[23] I.C.C. Phillips and I. Ulidowski. Reversing algebraic process calculi. *Journal of Logic and Algebraic Programming, 73*, pages 70–96. 2007.

[24] I.C.C. Phillips and I. Ulidowski. Reversibility and models for concurrency. In *Proc. of SOS 2007*, ENTCS 192, pages 93–108. 2007.

[25] I.C.C. Phillips, I. Ulidowski and S. Yuen. A Reversible Process Calculus and the Modelling of the ERK Signalling Pathway In *Procs. of RC 2013, LNCS 7581*, pages 218–232. Springer, 2013.

[26] O. Schneider and T. Worsch. A 3-state asynchronous CA for the simulation of delay-insensitive circuits. In *Procs. of ACRI 2012, LNCS 7495*, pages 565–574. Springer, 2012.