Foundations of Reversible Computation

Bogdan Aman, Gabriel Ciobanu, Robert Glück, Robin Kaarsgaard, Jarkko Kari, Martin Kutrib, Ivan Lanese, Claudio Antares Mezzina, Łukasz Mikulski, Rajagopal Nagarajan, et al.

Foundations of Reversible Computation

Bogdan Aman[1,2], Gabriel Ciobanu[1,2], Robert Glück[3], Robin Kaarsgaard[3], Jarkko Kari[4], Martin Kutrib[5], Ivan Lanese[6], Claudio Antares Mezzina[7], Łukasz Mikulski[8], Rajagopal Nagarajan[9], Iain Phillips[10(✉)], G. Michele Pinna[11], Luca Prigioniero[5,12], Irek Ulidowski[13], and Germán Vidal[14]

[1] Romanian Academy, Institute of Computer Science, Iaşi, Romania
baman@iit.tuiasi.ro
[2] A.I. Cuza University, Iaşi, Romania
gabriel@info.uaic.ro
[3] University of Copenhagen, Copenhagen, Denmark
{glueck,robin}@di.ku.dk
[4] University of Turku, Turku, Finland
jkari@utu.fi
[5] University of Giessen, Giessen, Germany
kutrib@informatik.uni-giessen.de
[6] Focus Team, University of Bologna/Inria, Bologna, Italy
ivan.lanese@gmail.com
[7] Università di Urbino, Urbino, Italy
claudio.mezzina@uniurb.it
[8] Folco Team, Nicolaus Copernicus University, Toruń, Poland
mikulskilukasz@gmail.com
[9] Middlesex University, London, England
R.Nagarajan@mdx.ac.uk
[10] Imperial College London, London, England
i.phillips@imperial.ac.uk
[11] Università di Cagliari, Cagliari, Italy
gmpinna@unica.it
[12] Università degli Studi di Milano, Milan, Italy
prigioniero@di.unimi.it
[13] University of Leicester, Leicester, England
iu3@leicester.ac.uk
[14] MiST, VRAIN, Universitat Politècnica de València, Valencia, Spain
gvidal@dsic.upv.es

Abstract. Reversible computation allows computation to proceed not only in the standard, forward direction, but also backward, recovering past states. While reversible computation has attracted interest for its multiple applications, covering areas as different as low-power computing, simulation, robotics and debugging, such applications need to be supported by a clear understanding of the foundations of reversible computation. We report below on many threads of research in the area of foundations of reversible computing, giving particular emphasis to the results obtained in the framework of the European COST Action IC1405, entitled "Reversible Computation - Extending Horizons of Computing", which took place in the years 2015–2019.

I. Ulidowski et al. (Eds.): RC 2020, LNCS 12070, pp. 1–40, 2020.
https://doi.org/10.1007/978-3-030-47361-7_1

1 Introduction

Reversible computation allows computation to proceed not only in the standard, forward direction, but also backward, recovering past states, and computing inputs from outputs. Reversible computation has attracted interest for multiple applications, covering areas as different as low-power computing [113], simulation [37], robotics [122] and debugging [129]. However, such applications need to be supported by a clear understanding of the foundations of reversible computation. Over the years, a number of theoretical aspects of reversible computing have been studied, dealing with categorical foundations of reversibility, foundations of programming languages and term rewriting, considering various models of sequential (automata, Turing machines) and concurrent (cellular automata, process calculi, Petri nets and membrane computing) computations, and tackling also the challenges posed by quantum computation, which is in a large part naturally reversible. We report below on those threads of research, giving particular emphasis to the results obtained in the framework of the European COST Action IC1405 [78], titled "Reversible Computation - Extending Horizons of Computing", which took place in the years 2015–2019 and involved researchers from 34 different countries.

The contents of this chapter are as follows. Section 2 covers category theory, Sect. 3 reversible programming languages, Sect. 4 term rewriting, and Sect. 5 membrane computing. We then discuss process calculi (Sect. 6), Petri nets (Sect. 7), automata (Sect. 8), and quantum verification and machine learning (Sect. 9). The chapter ends with a brief conclusion (Sect. 10).

2 Category Theory

Category theory is a framework for the description and development of mathematical structures. In category theory mathematical objects and their relationships within mathematical theories are abstracted into primal notions of *object* and *morphism*. Despite being a staple of the related field of quantum computer science for years (see, *e.g.*, [3,79,174]), category theory has seen comparatively little use in modelling reversible computation, where operational methods remain the standard. While the present section aims to give an overview of the use of categorical models in providing categorical semantics for reversible programming languages, categorical models have also been studied for other reversible computing phenomena, notably reversible event structures [65].

2.1 Dagger Categories

One approach to categorical models of reversible computation is given by dagger categories, *i.e.*, categories with an abstract notion of inverse given by assigning to each morphism $X \xrightarrow{f} Y$ an *adjoint* morphism $Y \xrightarrow{f^\dagger} X$, such that $(g \circ f)^\dagger = f^\dagger \circ g^\dagger$ and $\mathrm{id}_X^\dagger = \mathrm{id}_X$ (that is, composition is respected) and $f^{\dagger\dagger} = f$ for all compatible morphisms f and g. Note that this definition says nothing about how f and $f^\dagger$

ought to interact. As such, $f^\dagger$ is not required to "undo" the behaviour of f in any way, but can be *any* morphism with the appropriate signature, so long as the above constraints are met.

A useful specialisation of dagger categories, in connection with reversible computation, is dagger traced symmetric bimonoidal (or *rig*) categories, *i.e.*, dagger categories equipped with two symmetric monoidal tensors (usually denoted $- \oplus -$ and $- \otimes -$), interacting through a distributor and an annihilator, yielding the structure of a *rig* (*i.e.*, a ring without additive inverses). Iteration is modelled by means of a trace operator Tr (see [1,85,175]) such that $(\mathrm{Tr} f)^\dagger = \mathrm{Tr}(f^\dagger)$. These categories are strongly related to the dagger compact closed categories [3,174] that serve as the model of choice for the Oxford school of quantum computing.

The use of dagger traced symmetric bimonoidal categories to model reversible computations goes back at least as far as to the works by Abramsky, Haghverdi and Scott (see, *e.g.*, [2,4]) on (reversible) combinatory algebras, though its applications in reversible programming were perhaps best highlighted by the development of the Π and Π^0 calculi [34,83]. In addition, the reversible functional programming language Theseus [82] exhibits a correspondence with the Π^0 calculus. However, dagger traced symmetric bimonoidal categories are not strictly enough to model Π^0, as such categories fail to account for the recursive data types formed using $- \oplus -$, $- \otimes -$, and their units. In his recent thesis, Karvonen [94] describes precisely the categorical features necessary for such a correspondence, which he calls traced ω-continuous dagger rig categories.

Another notable application of this line of research is found in [167], where a reversible Π^0-like language is extended to describe quantum computations without measurement, but with support for (necessarily terminating) primitively recursive functions.

2.2 Inverse Categories

Another approach to model reversible computation is inverse categories [95] (see [40] for a more modern presentation), a specialisation of dagger categories in which morphisms are required to be partial isomorphisms. More precisely, each morphism $X \xrightarrow{f} Y$ may be *uniquely* assigned a *partial inverse* $Y \xrightarrow{f^\dagger} X$ satisfying $f \circ f^\dagger \circ f = f$.

The development of inverse categories as models of reversible computation was pioneered in the thesis of B.G. Giles [58], though a concrete correspondence was never provided. This work, combined with the comprehensive account of inverse categories with joins given in the thesis of Guo [67], was exploited in [86] to give an account of reversible recursion in inverse categories with joins.

Much of this theory was then put to use in [87], where the authors managed to show soundness, adequacy, and (under certain conditions) full abstraction for reversible flowchart languages [185] in a class of inverse categories with joins.

2.3 Monads and Arrows for Reversible Effects

The first account of monads pertaining to reversible computing was given in [71] as *dagger Frobenius monads*. Though these arise naturally in quantum computation in the context of measurement, it turns out that they are exceedingly rare in the case of classical reversible computing. A better concept for modelling and programming with reversible effects turns out to be that of *dagger* and *inverse arrows* [70], with examples such as reversible computation with mutable memory, errors and error handling, and more.

3 Foundations of Reversible Programming Languages

Reversible programming languages bridge the gap between the hardware and the specific application, and therefore play a central role in the development of reversible computing. Reversible languages must be expressive and usable in a variety of application domains. Their semantics must be precise and their programs accessible to program inversion, analysis and verification. Additionally, they must have efficient realisations on reversible devices and on standard ones. Recent programming language studies have advanced the foundations and theory of reversible languages in several interrelated directions.

3.1 Language Cores

Reversible languages have been reduced to their computational cores:

R-Core [63] is a structured reversible language consisting of a single command for *reversible store updates*, a single control-flow operator for *reversible iteration*, and data structures built from a single binary constructor and a single symbol. Despite its extreme simplicity, the language is *reversibly universal*, which means it is as computationally powerful as any reversible language can be. Its four-line program inverter is as concise as the one for Bennett's reversible Turing machines. The core language and a recent extension with *reversible recursion* were equipped with a denotational semantics [61,63,64].

R-While [62] adds reversible *rewrite rules* and *pattern matching* as syntactic sugar to R-Core, which makes the family of structured reversible languages more accessible to foundational studies and educational purposes than do reversible Turing machines and other reversible devices. The *procedural extension* [64] draws a distinction between tail-recursion by iteration and general recursion by reversible procedures, a notoriously difficult transformation problem in program inversion [96,151]. The *linear-time self-interpretability* makes the language also suitable for foundational studies of computability and complexity from a programming language perspective [84].

CoreFun [80] is a typed reversible functional language that seeks to reduce reversible functional programming [184] to its essentials so that it can serve as a foundation for modern functional language concepts. The language has a formal semantics and a type system to statically check for reversibility of programs.

3.2 Formal Semantics

Precise semantics is the foundation of every programming language, and formality is from where programming languages derive their usefulness and power.

A program is regarded as reversible if each of its meaningful subprograms is partially invertible. Thus, *reversible programs have reversible semantics* [61]. A foundation of the semantics has been established for structured reversible languages built on inverse categories [59,60]. This class of languages includes Janus, a reversible language that was originally formalised by conventional (irreversible) operational semantics, and the R-Core and R-While languages. For example, predicates and assertions occurring in reversible alternatives and reversible iterations are modelled by decision maps, in contrast to conventional semantics. A benefit of the reversible semantic approach is that program inverters and equivalences of reversible programs can be derived directly from the semantics.

The assumption of countable joins in inverse categories is suitable in a categorical account of reversible recursion [86], which enables modelling of procedures in reversible structured and functional languages. Reversibility of Janus was proved with a proof assistant [153].

3.3 Compilation Principles

High-level languages are more productive in most application domains, but high levels of computational abstractions do not come for free. A clean and effective translation to lower abstraction levels is required and sophisticated optimisations may be necessary to generate high quality implementations.

Dynamic memory management is a central runtime mechanism to support dynamic data structures in reversible machines. Its purpose is to support reversible object-oriented languages as well as the core languages described above. Garbage collectors that use multiple references [142] to overcome linearity requirements and heap manager algorithms have been developed and experimentally evaluated. To ease the analysis and optimisation when translating from a high-level reversible language to the underlying reversible machine, the *reversible single static assignment* (RSSA) form can be a suitable intermediate representation in optimising compilers [141]. Its aim is to allow for advanced optimisations such as register allocation on reversible Von Neumann machines.

The recent languages Joule [173] and ROOPL [68] demonstrated that well-known *object-oriented concepts* can be captured reversibly by extending a Janus-like imperative language. *Reversible data types* [43], that is data structures with all of its associated operations implemented reversibly, are enabled by dynamic allocation of constructor terms on the heap [11]. A reversible dynamic memory management based on the Buddy Memory system [99] has been developed and tested in a compiler targeting the assembly language of a reversible computer [43].

3.4 Reversibilisation Techniques

A separate approach to reversibility is *reversibilisation*, which turns irreversible computations into reversible computations. This can be achieved by extending the semantics of an irreversible language or by instrumenting an irreversible program to continually produce information that ensures reversibility.

Some reversibilisation techniques work without user interaction, while others require annotation of programs. Techniques have been developed in recent years that add tracing to term rewriting systems [150] and instrument C++ programs with incremental state saving [171]. Other investigations have focused on techniques for debugging concurrent programs [121,149] and on extending the operational semantics of an irreversible language with tracing [72], thereby defining the inverse semantics of the language. Hybrid approaches aim to combine reversibilisation and reversible sublanguages [172]. In general, the minimisation of the additional computational resources required for sealing information leaks by reversibilisation remains a central challenge.

4 Term Rewriting

Term rewriting [17,98,178] is a foundational theory of computing that underlies most rule-based programming languages. A *term rewriting system* (TRS) is specified as a set of rewrite rules of the form $l \to r$ such that l is a nonvariable term and r is a term whose variables appear in l. Positions are used to address the nodes of a term viewed as a tree. A *position* p in a term t is represented by a finite sequence of natural numbers, where $t|_p$ denotes the *subterm* of t at position p and $t[s]_p$ the result of *replacing the subterm* $t|_p$ by the term s. *Substitutions* are mappings from variables to terms.

Given a TRS $\mathcal{R}$, we define the associated rewrite relation $\to_{\mathcal{R}}$ as the smallest binary relation satisfying the following: given terms s, t, we have $s \to_{\mathcal{R}} t$ iff there exist a position p in s, a rewrite rule $l \to r \in \mathcal{R}$, and a substitution σ such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$. Given a binary relation $\to$, we denote by $\to^*$ its reflexive and transitive closure, i.e., $s \to^*_{\mathcal{R}} t$ means that s can be reduced to t in $\mathcal{R}$ in zero or more steps. The goal of term rewriting is reducing terms to so-called *normal forms*, where a term t is called *irreducible* or in *normal form* w.r.t. a TRS $\mathcal{R}$ if there is no term s with $t \to_{\mathcal{R}} s$. Computing normal forms can be seen as the counterpart of computing *values* in functional programming.

We also consider Conditional TRSs (CTRSs) of the form $l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_n \twoheadrightarrow t_n$, with $\twoheadrightarrow$ interpreted as *reachability* ($\to^*_{\mathcal{R}}$). Roughly speaking, $s \to_{\mathcal{R}} t$ iff there exist a position p in s, a rewrite rule $l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_n \twoheadrightarrow t_n \in \mathcal{R}$, and a substitution σ such that $s|_p = l\sigma$, $s_i\sigma \to^*_{\mathcal{R}} t_i\sigma$ for all $i = 1, \ldots, n$, and $t = s[r\sigma]_p$. Consider, e.g., the following CTRS $\mathcal{R}^{\mathsf{fn}}$:

$$\begin{aligned}
&\beta_1 : \mathsf{fn}([\,]) \to [\,] \\
&\beta_2 : \mathsf{fn}(\mathsf{person}(n, l) : xs) \to n : ys \Leftarrow \mathsf{fn}(xs) \twoheadrightarrow ys \\
&\beta_3 : \mathsf{fn}(\mathsf{city}(c) : xs) \to ys \Leftarrow \mathsf{fn}(xs) \twoheadrightarrow ys
\end{aligned}$$

where we use ":" and [ ] as list constructors. Here, β_1, β_2 and β_3 denote labels that uniquely identify each rewrite rule. Function fn takes a list of persons of the form person(*first_name*, *last_name*) and cities of the form city(*city_name*) and returns a list of first names. Note that it could be specified in a typical functional language (say, Haskell) as follows:

$$\begin{aligned}
&\mathsf{fn}\ [\,] = [\,]\\
&\mathsf{fn}\ ((\mathsf{Person}\ n\ l) : xs) = n : ys \ \ \text{where}\ ys = \mathsf{fn}\ xs\\
&\mathsf{fn}\ ((\mathsf{City}\ c) : xs) = ys \ \ \text{where}\ ys = \mathsf{fn}\ xs
\end{aligned}$$

4.1 Reversible Term Rewriting

In general, term rewriting is not reversible, even for injective functions; namely, given a rewrite step $t_1 \to t_2$, we do not always have a decidable method to get t_1 from t_2. One of the first approaches to reversibility in term rewriting is due to Abramsky [2], who considered reversibility in the context of *pattern matching automata*.[1] Abramsky's approach requires a condition called *biorthogonality* (which, in particular, implies injectivity), so that the considered automata are reversible. This work can be seen as a rather fundamental delineation of the boundary between reversible and irreversible computation in logical terms. However, biorthogonality is overly restrictive in the context of term rewriting, since almost no term rewrite system is biorthogonal. Another example of a term rewrite system with both forward and reverse rewrite relations is the *reaction systems for bonding* in [159]. It has been used to model a simple catalytic reaction, polymer construction, by a scaffolding protein and a long-running transaction with a compensation.

In the context of the COST action IC1405, Nishida *et al.* [148,150] introduced the first generic notion of *reversible rewriting*, a conservative extension of term rewriting based on a so-called *Landauer embedding*. In this approach, for every rewrite step $s \to_{\mathcal{R}} t$, one should store the applied rule β, the selected position p, and a substitution σ with the values of some variables (e.g., the variables that occur in the left-hand side of a rule but not in its right-hand side). Therefore, reversible rewrite steps have now the form $\langle s, \pi\rangle \rightharpoonup \langle t, \beta(p,\sigma) : \pi\rangle$, where $\rightharpoonup$ is a reversible (forward) rewrite relation and π is a *trace* that stores the sequence of terms of the form $\beta(p,\sigma)$. The dual, inverse relation $\leftharpoondown$ is also introduced, so that its union $\rightleftharpoons$ can be used to perform both forward and backward reductions.

Moreover, [148] also introduces a scheme to *compile* the reversible extension of rewriting into the system rules. Essentially, given a system $\mathcal{R}$, new systems $\mathcal{R}_f$ and $\mathcal{R}_b$ are produced, so that standard rewriting in $\mathcal{R}_f$, i.e., $\to_{\mathcal{R}_f}$, coincides with the forward reversible extension $\rightharpoonup_{\mathcal{R}}$ in the original system, and analogously $\to_{\mathcal{R}_b}$ is equivalent to $\leftharpoondown_{\mathcal{R}}$. Therefore, $\mathcal{R}_f$ can be seen as an *injectivisation* of $\mathcal{R}$, and $\mathcal{R}_b$ can be seen as the *inversion* of $\mathcal{R}_f$.

[1] Although he did not consider rewriting explicitly, pattern matching automata can also be represented in terms of standard notions of term rewriting.

For instance, the injectivisation $\mathcal{R}_f^{\mathsf{fn}}$ of the previous CTRS $\mathcal{R}^{\mathsf{fn}}$ is as follows:

$$\begin{aligned}
&\mathsf{fn}^{\mathsf{i}}([\,]) \to \langle [\,], \beta_1 \rangle \\
&\mathsf{fn}^{\mathsf{i}}(\mathsf{person}(n,l) : xs) \to \langle n : ys, \beta_2(l, ws) \rangle \Leftarrow \mathsf{fn}^{\mathsf{i}}(xs) \twoheadrightarrow \langle ys, ws \rangle \\
&\mathsf{fn}^{\mathsf{i}}(\mathsf{city}(c) : xs) \to \langle ys, \beta_3(c, ws) \rangle \Leftarrow \mathsf{fn}^{\mathsf{i}}(xs) \twoheadrightarrow \langle ys, ws \rangle
\end{aligned}$$

together with the corresponding inversion $\mathcal{R}_b^{\mathsf{fn}}$:

$$\begin{aligned}
&\mathsf{fn}^{-1}([\,], \beta_1) \to [\,] \\
&\mathsf{fn}^{-1}(n : ys, \beta_2(l, ws)) \to \mathsf{person}(n,l) : xs \Leftarrow \mathsf{fn}^{-1}(ys, ws) \twoheadrightarrow xs \\
&\mathsf{fn}^{-1}(ys, \beta_3(c, ws)) \to \mathsf{city}(c) : xs \Leftarrow \mathsf{fn}^{-1}(ys, ws) \twoheadrightarrow xs
\end{aligned}$$

For example, the following rewrite derivation in $\mathcal{R}^{\mathsf{fn}}$:

$$\mathsf{fn}([\mathsf{person}(\mathsf{john}, \mathsf{smith}), \mathsf{city}(\mathsf{london}), \mathsf{person}(\mathsf{ada}, \mathsf{lovelace})]) \to^* [\mathsf{john}, \mathsf{ada}]$$

is now as follows in $\mathcal{R}_f^{\mathsf{fn}}$:

$$\begin{aligned}
&\mathsf{fn}^{\mathsf{i}}([\mathsf{person}(\mathsf{john}, \mathsf{smith}), \mathsf{city}(\mathsf{london}), \mathsf{person}(\mathsf{ada}, \mathsf{lovelace})]) \\
&\qquad \to^* \langle [\mathsf{john}, \mathsf{ada}], \beta_2(\mathsf{smith}, \beta_3(\mathsf{london}, \beta_2(\mathsf{lovelace}, \beta_1))) \rangle
\end{aligned}$$

where $\beta_2(\mathsf{smith}, \beta_3(\mathsf{london}, \beta_2(\mathsf{lovelace}, \beta_1)))$ is the trace of the computation. Besides proving some fundamental properties of reversible rewriting, Nishida et al. [150] have developed a prototype implementation of the reversibilisation transformations (injectivisation and inversion), which is publicly available through a web interface from http://kaz.dsic.upv.es/rev-rewriting.html.

4.2 Application to Bidirectional Transformations

The framework of *bidirectional transformations* considers two representations of some data and the functions that convert one representation into the other and vice versa (see, e.g., [75] for an overview). Typically, we have a function called "get" that takes a *source* and returns a *view*. In turn, the function "put" takes a possibly updated view (together with the original source) and returns the corresponding, updated source. In this context, *bidirectionalisation* [128] aims at automatically producing one of the functions, typically producing a function put from the corresponding function get. For this purpose, a so-called *complement* function is often introduced so that get becomes injective (see, e.g., [55]).

In [152], Nishida and Vidal present a bidirectionalisation technique based on the injectivisation and inversion transformations of CTRSs from [150]. They also prove a number of relevant properties which ensure that changes in both the source and the view are correctly propagated and that no undesirable side-effects are introduced.

To be precise, given a get function f, the corresponding put can be automatically defined as follows:

$$\mathsf{put}_{\mathsf{f}}(v, s) \to s' \Leftarrow \mathsf{f}^{\mathsf{i}}(s) \twoheadrightarrow \langle v', \pi \rangle, \mathsf{f}^{-1}(v, \pi) \twoheadrightarrow s'$$

Note that the trace of a computation, π, plays the role of a *complement* (following the terminology in the literature of bidirectional transformations).

For instance, given the previous function fn, the corresponding put function is defined as follows:

$$\mathsf{put}_{\mathsf{fn}}(v, s) \rightarrow s' \Leftarrow \mathsf{fn}^{\mathsf{i}}(s) \twoheadrightarrow \langle v', \pi \rangle, \mathsf{fn}^{-1}(v, \pi) \twoheadrightarrow s'$$

so that, e.g., $\mathsf{put}_{\mathsf{fn}}([\mathsf{peter}, \mathsf{ada}], \beta_2(\mathsf{smith}, \beta_3(\mathsf{london}, \beta_2(\mathsf{lovelace}, \beta_1))))$ reduces to $[\mathsf{person}(\mathsf{peter}, \mathsf{smith}), \mathsf{city}(\mathsf{london}), \mathsf{person}(\mathsf{ada}, \mathsf{lovelace})]$. Note that the first element has been updated from $\mathsf{person}(\mathsf{john}, \mathsf{smith})$ to $\mathsf{person}(\mathsf{peter}, \mathsf{smith})$.

However, $\mathsf{put}_{\mathsf{f}}$ is only defined for "compatible" view updates. E.g., the function $\mathsf{put}_{\mathsf{fn}}([\mathsf{ada}], \beta_2(\mathsf{smith}, \beta_3(\mathsf{london}, \beta_2(\mathsf{lovelace}, \beta_1))))$ cannot be reduced to a value. In [152], the use of *narrowing* [76,176]—an extension of rewriting that replaces matching with unification—is introduced to precisely characterise *compatible* (also called *in-place*) view updates.

For example, given the trace $\beta_2(\mathsf{smith}, \beta_3(\mathsf{london}, \beta_2(\mathsf{lovelace}, \beta_1)))$, narrowing allows us to compute the *view skeleton* $[x_1, x_2]$. This means that any view update that can be obtained as an instance of $[x_1, x_2]$ is compatible with the trace (and, thus, the put function is well defined).

Finally, [152] also discusses some directions for dealing with view updates that are not compatible.

5 Membrane Computing

Natural computing is a complex field of research dealing with models and computational techniques inspired by nature that helps us in understanding the biochemical world in terms of information processing. Membrane computing [154] and reaction systems [53] are two important theories of natural computing inspired by the functioning of living cells.

Membrane computing deals with multisets of symbols processed in the compartments of a membrane structure according to some multiset rewriting rules; some of the symbols (presented with their multiplicity within the regions delimited by membranes) evolve in parallel according to the rules associated with their membranes, while the others remain unchanged and can be used in the subsequent steps. It is also possible to send multisets of symbols in the neighbouring membranes, the systems being organised in a tree-like fashion. The evolution takes place in a maximal parallel manner: all the instances of the applicable rules have to be applied in order to reach the next state.

The situation is different in reaction systems. These systems represent a qualitative model: they deal with sets rather than multisets. Two major assumptions distinguish the reaction systems from the membrane systems: (i) threshold assumption: reaction systems have actually an infinite multiplicity for their resources; (ii) no permanency assumption: only entities produced at one step will be present in the system at the next step.

The issue of reversibility in various computational paradigms has gained interest in recent years. In one of the earliest papers on reversibility in membrane systems [5], the authors (under the influence of category theory) presented

reversibility as a form of duality. A full description of this kind of reversibility in membrane systems is given by Agrigoroaiei and Ciobanu in [6].

In [7], Aman and Ciobanu investigated the reversibility of biochemical reactions in parallel rewriting systems; these systems can easily represent some classes of membrane systems and Petri nets. Formally, a parallel rewriting system is a tuple $(O, \mathcal{R}, w_0)$, where O is a finite alphabet of objects, $\mathcal{R}$ is a set of rewriting rules and w_0 is a multiset of objects over O. For each rule $r \in \mathcal{R}$ there exist the non-empty multisets $lhs(r), rhs(r) \in O^+$ standing for the left-hand side and right-hand side of the rule, respectively, such that $r : lhs(r) \to rhs(r)$. Given a multiset of rules F, then the left-hand side and right-hand side of it can be defined as: $lhs(F) = \sum_{r \in \mathcal{R}} F(r) \cdot lhs(r)$ and $rhs(F) = \sum_{r \in \mathcal{R}} F(r) \cdot rhs(r)$.

A parallel rewriting system $(O, \mathcal{R}, w_0)$ evolves in a maximal parallel manner. This means that a non-empty multiset R of rules is applicable to a multiset w of objects if $lhs(R) \leq w$ and there does not exist $r \in \mathcal{R}$ such that $lhs(r) \leq w - lhs(R)$. By applying a multiset R of rules, a multiset w of objects is transformed into another multiset $w' = w - lhs(R) + rhs(R)$ of objects. If no multiset of rules is applicable, then the computation stops.

The new features of this approach are given by adding an external control specified by using a special symbol $\rho \notin O$ that informs the system that a rollback will be executed, and by constructing two new sets of rules $\overrightarrow{\mathcal{R}} = \{u \to v|_{\neg\rho} \mid u \to v \in \mathcal{R}\}$ and $\overleftarrow{\mathcal{R}}_\rho = \{v \to u|_\rho \mid u \to v \in \mathcal{R}\} \cup \rho \to \lambda$ to mark the rules that will be applied in forward and backward steps, respectively.

Several theoretical results are obtained, including the so-called loop results and the connections between the evolutions of these systems and their reversible extensions. If there exist multisets of rules not competing for the same resources, then the following results hold.

A first result presents the **forward diamond** property:

If $w \xrightarrow{\overrightarrow{\mathcal{R}}} w'$ *and* $w \xrightarrow{\overrightarrow{\mathcal{R}'}} w''$, *where* $\overrightarrow{\mathcal{R}}$ *and* $\overrightarrow{\mathcal{R}'}$ *are two valid multisets of rules such that* $lhs(\overrightarrow{\mathcal{R}}) \cap lhs(\overrightarrow{\mathcal{R}'}) = \emptyset$, *then there exists a multiset* w_1 *such that* $w' \xrightarrow{\overrightarrow{\mathcal{R}'}} w_1$ *and* $w'' \xrightarrow{\overrightarrow{\mathcal{R}}} w_1$.

The second result presents the **reverse diamond** property:

If $w \xrightarrow{\overleftarrow{\mathcal{R}_\rho}} w'$ *and* $w \xrightarrow{\overleftarrow{\mathcal{R}'_\rho}} w''$, *where* $\overleftarrow{\mathcal{R}_\rho}$ *and* $\overleftarrow{\mathcal{R}'_\rho}$ *are two valid multisets of rules such that* $lhs(\overleftarrow{\mathcal{R}_\rho}) \cap lhs(\overleftarrow{\mathcal{R}'_\rho}) = \emptyset$, *then there exists a multiset* w_1 *such that* $w' \xrightarrow{\overleftarrow{\mathcal{R}'_\rho}} w_1$ *and* $w'' \xrightarrow{\overleftarrow{\mathcal{R}_\rho}} w_1$.

A forward step performed using the multiset $\overrightarrow{\mathcal{R}}$ of rules can be matched by a backward step performed using the multiset $\overleftarrow{\mathcal{R}_\rho}$ of rules, and vice-versa (**loop**):

$$w \xrightarrow{\overrightarrow{\mathcal{R}}} w' \quad \text{if and only if} \quad \rho w' \xrightarrow{\overleftarrow{\mathcal{R}_\rho}} w.$$

In [8], Aman and Ciobanu investigated reversibility in reaction systems. Reaction systems [53] deal with sets rather than multisets, assuming that each resource is present in the system in a sufficient amount to ensure that several reactions needing such a resource are not in conflict. Formally, a reaction system $\mathcal{A}$ is a tuple (S, A), where S is a finite alphabet and $A \subseteq rac(S)$. The set $rac(S) = \{(R, I, P) \mid R, I, P \subseteq S, R \cap I = \emptyset\}$ is the set of all reactions over S. Given a reaction $a = (R_a, I_a, P_a)$, the sets R_a, I_a and P_a contain the reactants, inhibitors and products of a, respectively. For a set $C \subseteq S$ and a set of reactions $A \subseteq rac(S)$, the result of applying A on C is defined by $res(A, C) = \bigcup_{a \in A} P_a$, and the evolution can be written as $C \xrightarrow{A} res(A, C)$. The set of all reactions from A that are enabled by C is $en(A, C) = \{a \in A \mid R_a \subseteq C, I_a \cap C = \emptyset\}$.

An interactive process is a pair $\pi = (\gamma, \delta)$ such that $\gamma = C_0, \ldots, C_{n-1}$, $\delta = D_1, \ldots, D_n$ with $n \geq 1$, where C_{j-1}, $D_j \subseteq S$ for $1 \leq j \leq n$ are the context and result sets, respectively. The sets D_j are computed using the equalities $D_1 = res(A, W_0)$ and $D_i = res(A, W_{i-1})$, where the sets $W_0 = C_0$ and $W_i = D_i \cup C_i$ for each $2 \leq i \leq n$ represent the states.

In order to have backward computations, we add to each state W_i a register T_i to remember objects no longer available after step i. The reverse of a set A of reactions is the set $\tilde{A} = \{(P_a, I_a, R_a) \mid (R_a, I_a, P_a) \in A\}$. If $\rho \notin W_i$ and $E_i \neq \emptyset$, then a forward computation $(W_i, T_i) \xrightarrow{E_i} (W_{i+1}, T_{i+1})$ takes place, where $T_{i+1} = inc(T_i) \cup \bigcup_{t \in W_i \setminus lhs(E_i)} (t, 0)$, $inc(T) = \bigcup_{(t,i) \in T} (t, i+1)$ and $W_{i+1} = res(E_i, W_i)$. However, if $\rho \in W_i$ and $\tilde{E}_i \neq \emptyset$, then a backward computation $(W_{i+1}, T_{i+1}) \overset{\tilde{E}_i}{\rightsquigarrow} (W_i, T_i)$ takes place, where $T_i = dec(T_{i+1})$, $dec(T_d) = \bigcup_{(t,i) \in T_d; i>0} (t, i-1)$ and $W_i = res(\tilde{E}_i, W_{i+1}) \cup zero(T_{i+1})$ with $zero(T) = \bigcup_{(t,0) \in T} t$.

If the states satisfy some preconditions, then backward reductions are the inverse of the forward ones, and vice-versa:

- If $W = res(\tilde{E}, W') \cup zero(T')$ and $\rho \in W'$, then

$$(W, T) \xrightarrow{E} (W', T') \text{ implies } (W', T') \overset{\tilde{E}}{\rightsquigarrow} (W, T).$$

- If $W' = res(E, W)$ and $\rho \notin W$, then

$$(W', T') \overset{\tilde{E}}{\rightsquigarrow} (W, T) \text{ implies } (W, T) \xrightarrow{E} (W', T').$$

An operational correspondence between reaction systems and rewriting theory is also proved. It allows a translation of the reversible reaction systems into some rewriting systems executable in the rewriting engine Maude [39].

In [163] Pinna pursues reversibility in membrane systems from a different perspective. The paper focuses on how to reverse steps in computations of membrane systems, without adding rules to represent the reverse application of the original rules. Just one assumption on rules is made, namely that rules are not allowed to rewrite a multiset of objects into an empty multiset: the application of a rule must have an effect, though this could be not observable. This requirement is driven by the necessity that, in order to reversely apply a rule, this one

must produce something. Furthermore, as in most rewriting systems, also in the considered membrane systems a computation step does not register the (multiset of) rules applied. Since this information may be crucial to reversely apply the same (multiset of) rules, one needs some strategies to solve the issue and obtain reversibility.

A solution can be to enrich each object with the information on how the particular object has been produced, namely each object now may carry the name of the rule r used to produce it. Objects are then $O \times R \cup \{\bot\}$ where R is the set of rules $\bigcup_i R_i$, with i ranging over the membranes, and $\bot$ denotes that the object is present in the initial configuration. The unique assumption is that rule names are unique. The drawback of this solution is that once an object is used the information on how it has been produced is lost.

To overcome this problem, the proposed solution is to add to the notion of configuration, previously a vector of multiset of objects, with one element for each membrane, a memory organised as a labelled partial order. Each element of the partial order corresponds to an object and carries also the information on which rule produced it. According to this a memory m is a triple $(X, \preceq, l)$ where $\preceq$ is a partial order and $l : X \to O \times R \cup \{\bot\} \times \{1, \ldots, n\}$ is the labelling associating the object, the name of rule that produced it and the membrane where the object is allocated. A configuration of a membrane system with n membranes is then the pair $\mathcal{C} = (C, \mathsf{m})$, where $C = (w_1, \ldots, w_n)$ is the tuple of multisets over objects O and $\mathsf{m} = (X, \preceq, l)$ is a memory such that for each $i \in \{1, \ldots, n\}$ it holds that $w_i = \mathsf{obj}_i(max(\mathsf{m}))$, where max gives the multiset of maximal elements of the memory and obj_i forgets the information about the rule.

The effect of applying a vector of multisets of rules $\mathcal{R}$ does not consist only in updating suitably the multisets of objects forming a configuration in the classical sense, but also in adding the information on which rule produced a specific object in the memory. This will be denoted with $(C, \mathsf{m})\{\![\mathcal{R} > (C', \mathsf{m}')$ where $C \stackrel{\mathcal{R}}{\Longrightarrow} C'$ is the usual step in membrane systems computation and the new memory m' is obtained adding to m the objects produced by the rules in $\mathcal{R}$ and by updating the partial order so that the produced elements are greater than the ones consumed by these rules.

Then the reverse application of a vector of multisets of rules can be obtained by looking in this memory for the maximal elements, which correspond to the right-hand sides of the rules to be reversely applied. The proper configuration is then computed from the new memory obtained by removing the maximal elements. The reverse application of a vector of multisets of rules $\mathcal{R}$ is denoted with $(C, \mathsf{m}) < \mathcal{R}]\}(C', \mathsf{m}')$, where the maximal elements of m' corresponding to the right-hand sides of rules in $\mathcal{R}$ are removed obtaining a memory m and a configuration C where each element $w_i = \mathsf{obj}_i(max(\mathsf{m}))$.

The following result has been proved:
Let Π_m be a membrane system with memory, (C, m) a configuration, and $\mathcal{R}$ be a vector of multisets of rules such that $(C, \mathsf{m})\{\![\mathcal{R} > (C', \mathsf{m}')$. Then, for all multi-rule vectors $\mathcal{R}'$ such that $(C', \mathsf{m}') < \mathcal{R}]\}(C, \mathsf{m})$, it holds that $\mathcal{R}' = \mathcal{R}$.

This simple implementation has the advantage of properly realising the causal reversibility. Furthermore the memory allows also to capture the dependencies among objects in a membrane system computation.

6 Process Calculi

Process calculi are a class of algebraic models for concurrent and distributed systems. Process calculi allow one to express the behaviour of a concurrent system in a concise way, abstracting away from implementation details, and focusing on the interaction patterns among the components of the system. Thus, it is possible to express the behaviour of a system in a mathematically precise way and verification techniques can be easily developed on top of it.

Research on reversing process calculi can be perhaps tracked back to the Chemical Abstract Machine [30], a calculus inspired by chemical reactions whose operational semantics defines both forward and reverse reduction relations. The first attempts to reverse existing process calculi can be found in [44,46], where a reversible extension of CCS [140] was presented. A main contribution of [44] was the definition of the notion of *causal-consistent* reversibility: any action can be undone, provided that its consequences, if any, are undone first. This definition is tailored to concurrent systems, where actions may overlap in time, hence saying "undo the last action" is not meaningful. Notably, this definition relates reversibility to causality instead of time, thus it can be applied even in those settings, such as some distributed systems, where no unique notion of time exists. A survey on causal-consistent reversibility can be found in [120].

6.1 Reversing Process Calculi

Following [44], causal-consistent extensions of other and more expressive process calculi have been defined. They can be divided into two families, one dealing with calculi equipped with labelled transition system semantics (describing interactions between the process and the outside world), and one dealing with reduction semantics (describing the evolution of processes in isolation). The former is more general, while the latter is normally simpler and hence more easily applicable to expressive calculi. The first approach extended causal-consistent reversibility from CCS to any calculus defined using a specific SOS format (a subset of the *path* format [146]) [160,161], and to π-calculus [42]. In the second line of research we find extensions of a fragment of CCS with biological relevance [35,36], of the higher-order π-calculus [117,119], of the coordination language Klaim [56], of a π-calculus with sessions [179], and of a CCS with broadcast communications [133]. The instance of the framework in [160] on CCS is called CCSK. CCSK differs from the reversible CCS in [44] in the way history is kept. Indeed, the approach of [160] can be considered static, since the structure of processes does not change during computation, and the minimal history information needed to enable reversibility is kept in the processes themselves, while in [44] the process is consumed during execution (as standard in process calculi) and larger

memories are added to store history information. Nonetheless the two methods are equivalent as hinted at by [130] and fully proved by [115], where a mapping from an instance on CCS of [160] to the reversible CCS of [44] and vice versa is presented.

As discussed above, causally-consistent reversibility relates reversibility with causality. In CCS just one main notion of causality exists, and both the reversible variants of CCS above are based on it. In the π-calculus, many relevant notions of causality exist, which differ in the treatment of parallel extrusions of the same name. In [131] a uniform framework to define reversible π-calculi is presented. The framework is parametric w.r.t. a data structure that stores information about extrusions of a name. Different data structures yield different approaches to the parallel extrusion problem, leading to different ways of reversing a name extrusion, thus giving rise to different reversible variants of the π-calculus.

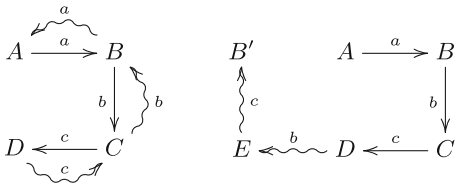

Fig. 1. Example of causal-consistent (left) and out-of-causal order reversibility (right)

6.2 Controlled Reversibility

The line of research described above focused on uncontrolled reversibility, defining how to reverse a process execution (in particular, which history and causal information is needed, and how to manage it), but not specifying when and whether to prefer backward execution over forward execution or vice versa. Uncontrolled reversibility allows one to understand how reversibility works, but not to exploit it into applications. Indeed, different application areas need different mechanisms to control reversibility. For instance, in biological systems the direction of the computation depends on physical conditions such as temperature and pressure, while in reliable systems reversibility is used to recover a consistent state when a bad event occurs. Triggered by these needs different mechanisms for controlling reversibility have been proposed (see the categorisation in [118]). For instance, [45,179] introduced irreversible actions to avoid going backward after a relevant result has been computed. Instead, [56,57,114,116,118,126] proposed an explicit rollback operator undoing a past action inside calculi where normal computation is forward, and a mechanism of alternatives allowing one to avoid trying the same path again and again. As shown in [57], the rollback operator satisfies a simple intuitive specification, namely that it is the smallest causal-consistent sequence of backward moves undoing the target action. Also, [18] let an energy potential drive the direction of computation while [158] introduced a forward monitor controlling the direction of execution of a reversible monitored

process. A process calculus with a prefixing operator to model locally-controlled reversibility is introduced in [102,103]. Actions can be undone spontaneously, as in other reversible process calculi, or as pairs of concerted actions, where performing a weak action forces the undoing of a past action. Concerted actions allow one to model out-of-causal order computation, where effects can be undone before their causes, which is forbidden in most other reversible calculi. This form of reversibility is common in biochemical reactions, e.g., in the hydration of formaldehyde in water into methanediol. Such a feature can be disabled by considering a reduced form of concerted actions.

Reversibility, both in causal order and out-of-causal order, can be modelled in reversible event structures [157].

Figure 1 shows the difference between causal-consistent (left) and out-of-causal order reversibility. In both cases, the system performs actions a, b and c to reach state D. On the left, in order to get back to the original state, one has to first undo (in Fig. 1 undoing is represented with squiggly arrows) c then b and finally a. On the right, since causes do not need to be respected, the system can undo b before c, reaching in this way a new state E which may not have been reachable from the initial configuration by just using forward steps. From there, a and c may or may not be undoable. In the example, only c can be undone, leading to B'. If undoing b and undoing c do commute, then $B = B'$.

6.3 Analysis Techniques

Despite the proliferation of calculi for reversibility, when the COST Action IC1405 started, analysis techniques for reversible calculi were very limited, consisting essentially in some limited analysis about behavioural equivalences (in particular, forward-reverse bisimilarity [161]) and a technique for causal compression in CCS with irreversible actions [101]. Thus, the work in the COST Action tackled analysis techniques in depth, considering behavioural equivalences, contracts [77] and session types [77].

Behavioural Equivalences. Understanding which notions of behavioural equivalences are suitable for reversible process calculi is a non-trivial, and still open, problem.

As shown in [119], notions of weak bisimilarity that do not distinguish forward actions from backward actions are very coarse, while notions of strong bisimilarity distinguishing them, such as forward-reverse bisimilarity [161], are very fine-grained, hence other notions are worth exploring.

In [135] Mezzina and Koutavas studied testing preorders, and in particular a safety one and a liveness one, in a reversible CCS where reductions are totally ordered and rollbacks lead systems to past states. Liveness and safety in this setting correspond to the should-testing [166] and inverse may-testing preorders [50] for the underlying forward calculus, respectively. In general, one would expect the models of these preorders to be based on both forward and backward transitions, thus offering complex proof techniques for verification. Instead, in [135] full abstraction of liveness and safety is based only on forward

transitions and limited rollback points, giving rise to considerably simpler proof techniques. Moreover, total reversibility allows one to make finer observations w.r.t. liveness, but not w.r.t. safety.

Contracts. (Binary) contracts are a behavioural model [77] to study the interactions between a client and a server. The first investigation of contracts in a reversible setting appeared in [21,22]. There, both the client and the server could rollback to a previous checkpoint at any moment. The main result was that the compliance relation, ensuring that the client and the server can successfully interact, and the sub-behaviour relation, are both decidable, and they remain so also when the possibility of skipping some messages is added.

In retractable contracts [23,24] the client and the server can both get back to previous decision points and take alternative paths only when the interaction is stuck. The main results in [23,24] are that retractable contracts are a conservative extension of contracts, both compliance and the subcontract relation are decidable in polynomial time, and the dual of a contract always exists and has a simple syntactic characterisation. Furthermore, retractable contracts are equivalent to a novel model of contracts featuring a speculative choice: all the options of the choice are explored concurrently, and the computation succeeds if at least one of the options is successful. In [20], a three-party game-theoretic interpretation of retractable session contracts [23] has been proposed. In such an interpretation a client is compliant with a server if and only if there exists a winning strategy for a particular player in a game-theoretic model of contracts. Such a player can be looked at as a mediator, driving the choices in the retractable points.

Session Types. Session types [77] are one of the formalisms that have been proposed to structure interaction and reason over communicating processes and their behaviour. In a series of works [136–138] reversible monitored semantics for binary [136,138] and multiparty [137] session types is investigated. The novelty of the approach is that monitors are derived by types, and they store all the needed information to bring the system back to previous states. This implies that processes of the system are oblivious to reversibility, as they do not store any information about past computations. A deeper discussion on session types and reversibility can be found in [134].

7 Petri Nets

Petri nets [165] are a mathematical formalism for modelling and reasoning on concurrent systems. In most of the cases, Petri nets are four-tuples containing two finite sets, of active (actions/transitions) and static (places) elements, which are connected by a flow function (or relation) with initial state given by tokens scattered on places. In what follows, by Petri net we mean its most common variant, called place-transition net.

Petri nets support both action-based and state-based approaches (via reachability graphs which are equivalent to transition systems). Reversibility in Petri

nets was always an important notion, however its meaning changed in time. At first, in the seventies, the notion of reversibility referred to nets where each transition has its inverse [54]. Such a notion of local reversibility is very close to the one currently used in other fields, like programming languages or process calculi. This notion of reversible nets (also called symmetric nets [54]) is still occasionally used to define the inverse net [33]. The time complexity of some decision problems in bounded symmetric Petri nets is lower than in the general case of bounded nets. The other meaning of reversibility in Petri nets, also called cyclicity [33], takes a global approach and requires the initial state of the net to be reachable from any other reachable state [147]. Petri nets are called symmetric also in other situations than the described local notion of reversibility [41].

During the four years of the COST Action IC1045, "Reversible Computation - Extending Horizons of Computing", the notion of local reversibility was investigated. One can divide the proposed contributions into three main threads: two of them consider how to reverse a single transition in a Petri net, allowing one to use, respectively, a single reverse transition or a set of reverses. The last thread focuses on modelling reversible semantics in specific models based on Petri nets.

An approach to invert a single transition using a single (strict) reverse was investigated under both the sequential semantics and the true concurrent semantics. The case of sequential semantics was considered in [28]. The strict reverse is added to the net as a fresh transition with arcs copied from the original one, but with the opposite direction. The problem of checking whether the set of reachable markings in a net changes, when a strict reverse for a single transition is added, was proven to be undecidable. The opposite result was shown for the set of all coverable markings. Another important fact shown in [28] is related to cyclicity: introducing a strict reverse in a cyclic net may change the set of reachable markings.

The above problem of checking whether the set of reachable markings in a net changes by adding a strict reverse for a single transition becomes decidable for the bounded nets. Therefore, one can ask a more general question - is it possible to reverse the specified transition while only requiring the resulting net and the given one to have isomorphic behaviour (i.e., isomorphic reachability graph), but allowing one to change the structure of the net? The question has been answered by using well-known techniques from region theory [19]. There are transition systems which are reachability graphs of a bounded Petri net where transitions cannot be inverted by strict reverses, but one can easily combine separate solutions for different transitions to solve the problem [26]. Even in the special case of linear transition systems over binary sets of actions the transitions cannot be always inverted by strict reverses. In such systems, the time complexity of the problem of checking whether the set of reachable markings changes by adding a strict reverse for a single transition is linear [48]. Another special case of bounded nets are occurrence nets, that is 1-safe and acyclic nets without backward conflicts, where one can always use strict reverses. This property of occurrence nets and their infinite extensions was used as an intermediate step in [132], described later on.

Another line of research on strict reverses considers systems under concurrent semantics of action execution. In such systems one can execute more than one action at the same time, including the situation when a single action is executed multiple times (auto-concurrence). Reversing atomic transitions in such systems is discussed in [49]. In simple cases, where auto-concurrence is excluded, one can reduce reversing under the concurrent semantics to the sequential case. However, in the case of true multisets of actions executed simultaneously, one needs to allow mixed reverses (i.e., steps where both forward and backward actions are present) and true concurrent reversing can be reduced to coping with all spikes (i.e., multisets of actions with singleton support).

In a more general setting, in order to invert a single transition, one can allow to define a set of reverses with the opposite effect, called effect reverses [26]. In such a case, the problem of finding a bounded Petri net where each transition can be reversed and with isomorphic behaviour becomes always solvable [26]. Hence, some systems where inverting transitions using strict reverses was impossible become reversible in this setting. Moreover, the price to make any bounded net ready for inverting by the sets of effect reverses is not high - one needs to transform the original net into its complementary version, which doubles the size of the set of places [26].

A similar attempt for unbounded nets is presented in [139]. There are unbounded nets which cannot be inverted even using infinite sets of effect reverses for their transitions. However, if it is possible, then finite sets are enough. The problem of finding a possibly totally different net with isomorphic behaviour that can be reversed was reduced to extending the existing one by new places which do not disable any transitions in any reachable state and checking whether there exists a pair of problematic states. Those pairs of problematic states are strongly structured, with a natural partial order. The set of all minimal pairs of problematic states for a given system is finite, however, the problem of checking whether two given states form a problematic pair is not elementary, while the problem of checking whether there exists at least one such pair is undecidable [139].

A different line of research considers extensions of Petri nets with causal-consistent local reversibility [132]. Such an extension can be obtained for any place transition net by unfolding it into occurrence nets and folding them back to a coloured Petri net with an infinite number of colours. Those colours are used to encode the content of a stack used to reverse the computation. The price to be paid is that coloured Petri nets with infinitely many colours are in general Turing complete.

Another approach to investigate causal-consistent local reversibility, but also out-of-order local reversibility, is the biologically inspired model of reversing Petri nets [155]. There tokens are persistent bases connected by bonds which are relocated by transitions of the net. The greatest limitation of the approach is the requirement of finiteness and acyclicity of the net modelled in this way. On the other hand, one can encode reversing Petri nets into coloured Petri nets with a finite number of colours [27], hence also into classical bounded

place-transition systems. Moreover, reversing Petri nets were successfully applied to the distributed antenna selection problem [156].

Petri net theory has been deeply studied. Cyclic and symmetric systems play quite an important role, however the issue of equipping concurrent systems with reversing mechanisms was not explored. The research conducted as a part of the COST Action IC1405 "Reversible Computation - Extending Horizons of Computing" enriched the theory of Petri nets by exploring some approaches to reverse transitions in existing systems. Although the effect of adding reverses of the actions to the existing system is in general difficult to evaluate (the problem of behaviour preservation is undecidable for place-transition nets), the problem can be solved if one allows unbounded stacks (coloured Petri nets approach) or restricts oneself to bounded models.

8 Automata

Automata theory studies abstract machines, or automata, as mathematical models of computation. They help in understanding limits of computation and the role of various resources – such as time and space – on the computational power. Examples of widely studied classes of automata include finite automata (bounded memory), pushdown automata (infinite memory organised as a stack), counter machine (infinite memory organised as counters), Turing machines (infinite memory tape) and cellular automata (massively parallel regular network of finite automata). These come in several flavours and variations, e.g., with respect to determinism. An automaton is reversible if it preserves information so that its computation can be retraced back in time. All the automata classes above can support reversibility. See [105,143] for details on computation by various models of reversible automata.

8.1 Finite Automata

Reversibility in finite automata has been widely investigated, e.g., [9,162]. The class of languages having a reversible one-way automaton is a proper subclass of the regular one. However, different models have been considered, depending on whether automata are required to have only one initial state and/or only one final state. Languages not having any reversible classical automaton have been characterised in terms of a forbidden pattern in the minimum automaton [73]. In the same paper, an NL-complete method to decide whether the language accepted by a given deterministic finite automaton can also be accepted by some reversible deterministic finite automaton has been derived.

In case the language accepted by a deterministic finite automaton is reversible, the size of the smallest reversible automaton may be exponential with respect to the size of the minimal irreversible one [73]. Recently analyses about the descriptional complexity of reversible deterministic finite automata provided some techniques to simulate these devices in an efficient way [123,125]. Indeed, though converting a deterministic automaton into a reversible one may require

an exponential increase in size, the proposed representation allows to limit this cost by concisely representing the reversible automaton rather than explicitly writing down its description.

Based on the forbidden pattern approach, the degree of irreversibility for a regular language has been studied [13]. The degree is defined to be the minimal number of such forbidden patterns necessary in any deterministic finite automaton accepting the language. It is shown that the degree induces a strict infinite hierarchy of language families. The behaviour of the degree of irreversibility under the usual language operations union, intersection, complement, concatenation, and Kleene star, has been studied, showing tight bounds (some asymptotic) on the degree.

Because of the narrowness of the power of reversible finite automata with respect to the irreversible ones, the definition of reversibility has been relaxed, by considering finite automata whose computations can be reversed, at any point, by accessing the last k symbols read from the input, for a fixed k. These devices are said to be "weakly irreversible". Characterisations of languages accepted by weakly irreversible automata and languages not having any weakly irreversible automaton ("strongly irreversible" languages) have been given [124].

Another treatment of a relaxed definition of reversibility concerns nondeterminism. It turned out that reversible nondeterministic finite automata are more powerful compared to their reversible deterministic counterparts, but still cannot accept all regular languages [74]. The two notions of relaxed reversibility have been compared and closure properties of the language family induced by these devices have been derived.

8.2 Pushdown Automata

Reversible classical pushdown automata have been introduced in [107]. Their computational capacity turned out to lie properly in between the regular and deterministic context-free languages. In the same paper, it is shown that a deterministic context-free language cannot be accepted reversibly if more than real-time is necessary for acceptance. Closure properties as well as decidability questions for reversible pushdown automata are studied and it is shown that the problem to decide whether a given nondeterministic or deterministic pushdown automaton is reversible is P-complete, whereas it is undecidable whether the language accepted by a given nondeterministic pushdown automaton is reversible.

One extension of finite automata in order to enlarge the underlying language class as well as to preserve many positive closure properties and decidable questions is represented by input-driven pushdown automata. Such automata share many desirable properties with finite automata, but still are powerful enough to describe important non-regular behaviour. Basically, for such devices the operations on the pushdown store are determined by the input symbols. With respect to reversibility they have been studied in [110]. So, the sub-family of the context-free languages that share the two important properties of being accepted by an input-driven pushdown automaton as well as of being accepted by a reversible pushdown automaton are considered. This intersection can be defined on the

underlying language families or on the underlying machine classes. It turned out that the latter class is properly included in the former. The relationships between the language families obtained in this way and to reversible context-free languages as well as to input-driven languages are studied. In general, a hierarchical inclusion structure within the real-time deterministic context-free languages is obtained. Finally, the closure properties of these families under the standard operations are investigated and it turned out that all language families introduced are anti-AFLs (that is, they are not closed under any of the operations required to be an Abstract Family of Languages).

Since reversible finite automata do not accept all regular languages and reversible pushdown automata do not accept all deterministic context-free languages, it is of significant interest both from a practical and theoretical point of view to close these gaps. Therefore these reversible models have been extended by a preprocessing unit which is basically a reversible injective and length-preserving sequential transducer [16]. It turned out that preprocessing the input using such weak devices increases the computational power of reversible deterministic finite automata to the acceptance of all regular languages. On the other hand, for reversible pushdown automata the accepted family of languages lies strictly in between the reversible deterministic context-free languages and the real-time deterministic context-free languages. Moreover, it has been derived that the computational power of both types of machines is not changed by allowing the preprocessing sequential transducer to work irreversibly.

Two-pushdown automata where the input is placed in one pushdown and that perform computations by inspecting and rewriting words at the top of the pushdowns are of particular interest as the deterministic variant is known to characterise the class of Church-Rosser languages when the rewriting is length-reducing. Such reversible two-pushdown automata are studied in [14]. A separation of the deterministic and reversible variants are obtained as well as the incomparability with the (deterministic) context-free languages. However, their properties of emptiness, (in)finiteness, universality, inclusion, equivalence, regularity, and context-freeness are not even semi-decidable.

8.3 Finite State and Pushdown Transducers

Computational models are not only interesting from the viewpoint of accepting some input, but also from the more applied perspective of transforming some input into some output. Transductions that are computed by different variants of transducers are studied in detail in the book of Berstel [31].

Reversibility in transducing devices has been investigated recently in [47,111] for deterministic finite state transducers. In [111], the families of transductions computed are classified with regard to three types of length-preserving transductions as well as to the property of working reversibly. It is possible to settle all inclusion relations between these families of transductions even with injective witness transductions. Furthermore, the standard closure properties and decidability questions have been investigated. It turned out that the non-closure under almost all operations can be shown, whereas all decidability questions

can be answered in polynomial time. Finally, the strict concept of reversibility is relaxed and an infinite and dense hierarchy with respect to the grade of reversibility is obtained.

Deterministic pushdown transducers have also been introduced, and analysed with respect to their ability to compute reversible transductions [66]. Now, the families of transductions computed are classified with regard to four types of length-preserving transductions as well as to the property of working reversibly. It turns out that accurate to one case separating witness transductions can be provided. For the remaining case it is possible to establish the equivalence of both families by proving that stationary moves can always be removed in length-preserving reversible pushdown transductions.

8.4 Queue Automata and Limited Automata

A further natural and well-studied extension of finite automata are queue automata, where the extension is by a storage media of type queue. Their reversible variant has been studied in [109]. In contrast to, for example, finite or pushdown automata, it has been shown that any queue automaton can be simulated by a reversible one. So, reversible queue automata are as powerful as Turing machines. Therefore it is of interest to impose time restrictions on queue automata. Quasi real-time and real-time computations have been considered. It has been shown that every reversible quasi real-time queue automaton can be sped up to real-time. On the other hand, under real-time conditions reversible queue automata are less powerful than general queue automata. Furthermore, a lower bound of $\Omega\left(\frac{n^2}{\log(n)}\right)$ time steps for real-time queue automata witness languages to be accepted by any equivalent reversible queue automaton has been exhibited. The closure properties of reversible real-time queue automata are similar as for reversible deterministic pushdown automata. Moreover, all commonly studied decidability questions such as emptiness, finiteness, or equivalence are not semi-decidable for reversible real-time queue automata. Furthermore, it is not semi-decidable whether an arbitrary given real-time queue automaton is reversible.

A k-limited automaton is a linear bounded automaton that may rewrite each tape square only in the first k visits, where $k \geq 0$ is a fixed constant. It is known that these automata accept context-free languages only. The deterministic k-limited automata have been investigated towards their ability to perform reversible computations [112]. It turned out that, for all $k \geq 0$, sweeping k-limited automata accept regular languages only. In contrast to reversible finite automata, all regular languages are accepted by sweeping 0-limited automata. Then the computational power gained in the number k of possible rewrite operations has been studied. It has been shown that the reversible 2-limited automata accept regular languages only and, thus, are strictly weaker than general 2-limited automata. Furthermore, a proper inclusion between reversible 3-limited and 4-limited automata languages has been obtained. The next levels of the hierarchy are separated between every k and $k + 3$ rewrite operations. Finally,

it turned out that all k-limited automata accept Church-Rosser languages only, that is, the intersection between context-free and Church-Rosser languages contains an infinite hierarchy of language families beyond the deterministic context-free languages.

8.5 Cellular Automata

A cellular automaton (CA) is a dynamical system on an infinite grid of cells defined by a local update rule that is applied simultaneously at all cells. More precisely, in the usual rectilinear d-dimensional setting the cells are the elements of $\mathbb{Z}^d$ and each cell stores an element of a finite state set A. The dynamics is specified by a finite neighbourhood $D \subseteq \mathbb{Z}^d$ that gives the relative offsets to neighbours of cells, and a local rule $f : A^D \longrightarrow A$ that gives the new state of a cell based on the previous states in its neighbourhood. A configuration $c : \mathbb{Z}^d \longrightarrow A$, specifying the global state of the system, changes in a single time unit to become the new configuration c' with $c'(\vec{n}) = f(\sigma^{\vec{n}}(c)_{|D})$ for every cell $\vec{n} \in \mathbb{Z}^d$, where $\sigma^{\vec{n}}$ denotes the shift map that translates the configurations so that cell $\vec{n}$ moves to the origin.

By carefully choosing the update rule f, the global dynamics $c \mapsto c'$ can be made information preserving. In this case, an inverse cellular automaton retraces the computation back in time, and the cellular automaton is called reversible (RCA). See [90] for a recent survey on reversible cellular automata. Cellular automata have an important role as providing simple models in microscopic physics, and because of time-reversibility of microscopic dynamics the cellular automata models are also typically reversible [181]. Reversible cellular automata are able to carry out universal computation [180], even in the one-dimensional setting [144].

In the symbolic dynamics nomenclature reversible cellular automata are called automorphisms of the (full) shift. By Hedlund's theorem [69] cellular automata are precisely the transformations $A^{\mathbb{Z}^d} \longrightarrow A^{\mathbb{Z}^d}$ of the configuration space that commute with shifts $\sigma^{\vec{n}}$ and that are continuous under the compact prodiscrete topology on $A^{\mathbb{Z}^d}$. Reversibility then just means that the transformation is a bijection, i.e., a homeomorphism. Automorphisms form a group under composition, and the structure of the automorphism group of the full shift (as well as of its subshifts) is a topic of active research [168]. For example, it is not known if the groups of one-dimensional RCA over two states and over three states are isomorphic with each other.

Decision Problems. Decision problems concerning reversibility and related properties have been extensively studied. There are efficient algorithms to test one-dimensional cellular automata for reversibility [177] while in higher dimensional cases reversibility is undecidable [88]. It is also undecidable, even in the one-dimensional case, whether a given RCA is periodic [92], that is, whether some iteration of the CA amounts to the identity function. Periodicity among one-sided RCA is not known to be decidable or undecidable at this time, where

one-sidedness refers to the property that the neighbours to the left of a cell have no influence on its next state, nor on the previous state given by the inverse automaton. Periodicity in the one-sided case remains an active research topic due to its link to the finiteness problem of groups generated by Mealy automata [51].

Two dynamical systems are called conjugate if there is a homeomorphism between them that maps orbits to orbits. Conjugate systems are essentially identical. It is undecidable if two given cellular automata are conjugate [81]. This is true even for one-dimensional cellular automata, but if the considered CA are reversible then the undecidability is known in the two- and higher dimensional cases only.

Physical Universality and Glider Automorphisms. A cellular automaton is called physically universal if it can implement any transformation of patterns on any finite domain of cells by suitably choosing the initial states outside the domain. There are reversible cellular automata that are physically universal [170], even in the one-dimensional setting [169]. These automata (reversibly) break the input pattern into fleets of gliders that scatter out of the finite domain. Symmetrically, the inverse automaton breaks the desired output pattern into fleets of inverse gliders. The task of the surrounding gadget is to change the first fleet into the second fleet to implement the desired transformation.

Glider automorphisms that decompose finite configurations into fleets of gliders have been studied in more general subshifts, and they have found applications in understanding the structure of the automorphism groups [100].

Reversible Cellular Automata and Mahler's Problem in Number Theory. If real numbers are written in base pq for some co-primes p and q then there is no carry propagation when numbers are multiplied by constant p. This means that multiplying by p is a local operation, that is, a reversible cellular automaton. Composing such reversible cellular automata yields, for example, an RCA for multiplying numbers in base 6 by constant 3/2.

Mahler's problem asks whether there exists some positive real number ξ such that the fractional part of $\xi \left(\frac{3}{2}\right)^n$ is less than 0.5 for all positive integers n [127]. So the fractional part of the number should remain below one half no matter how many times the number is multiplied by 3/2. The problem is still unsolved. The problem has a very simple interpretation in terms of the RCA that multiplies by 3/2 in base six [89], and using this link it has been proved that for arbitrarily small $\varepsilon > 0$ there is a number $\xi > 0$ and a finite union $U \subseteq [0, 1)$ of intervals of total length ε such that the fractional parts of all $\xi \left(\frac{3}{2}\right)^n$ are in U [91]. The dynamical property of expansivity of the associated reversible cellular automaton plays a central role in the proof. Conversely, there is also a finite union $V \subseteq [0, 1)$ of intervals of total length $1 - \varepsilon$ that does not contain the fractional parts of all $\xi \left(\frac{3}{2}\right)^n$ for any $\xi > 0$.

Asynchronous Updating. In an asynchronous cellular automaton (ACA) only some cells are updated simultaneously. In the one-dimensional setting, one possibility is that states are updated sequentially during a left-to-right (or right-to-left) sweep across the entire infinite line of cells. Such a setup is studied in [93] where the update performed once in each position is given by a reversible block rule $A^n \longrightarrow A^n$ on n consecutive cells. The authors give a precise characterisation of the one-dimensional cellular automata that can be realised by such a sweep. It turns out that not all reversible CA can be realised, while also some non-reversible ones can be obtained. It is decidable whether a CA can be realised that way or not.

Self-Timed Cellular Automata. *Self-Timed Cellular Automata (STCA)* are a form of Asynchronous Cellular Automata where transitions of cells can take place if they are triggered by transitions of the neighbouring cells. *Delay-Insensitive (DI)* circuits are asynchronous circuits which make no assumption about delays within modules or wires of circuits, and where there is no global clock [97]. As a result, logical gates such as NAND and XOR are not Turing-complete when operated in a DI environment. A lot of research went into finding universal sets of DI modules and [145] contributes a solution for reversible DI circuits in terms of STCAs. Serial and parallel DI circuits are simulated with new STCAs that contain rules for signal movement, right and left turn, memory toggle, merge, fork and join, and parallel crossing of signals. In addition to a number of reversibility and determinism properties, including local determinism and local reversibility, the STCAs exhibit direction-reversibility, where reversing the direction of a signal and running a circuit forwards is equivalent to running the circuit in reverse. Benefits of direction-reversibility are discussed, including garbage-less implementation of reversible functions.

Cellular Automata as Language Acceptors. From the perspective of language recognition, real-time bounded cellular automata which are reversible on the core of computation, that is, from initial configuration to the configuration given by the time complexity, have been studied in [106]. The question whether for a given real-time CA working on finite configurations with fixed boundary conditions there exists a reverse real-time CA with the same neighbourhood has been addressed. It has been shown that real-time reversibility is undecidable, which contrasts the general case, where reversibility is decidable for one-dimensional devices. Moreover, the undecidability of emptiness, finiteness, infiniteness, inclusion, equivalence, regularity, and context-freedom has been proved. First steps towards the exploration of the computational capacity have been done and closure under Boolean operations have been shown.

Similar investigations for real-time one-way cellular automata have been done in [108]. In this case, it turned out that the standard model with fixed boundary conditions is quite weak in terms of reversible information processing, since it accepts exactly the regular languages reversibly. The extension that allows the information to flow circularly from the leftmost cell into the rightmost cell does

not increase the computational power in the general case, but does increase it for reversible computations. On the other hand, the model is less powerful than real-time reversible two-way cellular automata. Additionally, it has been derived that the corresponding language class is closed under Boolean operations, and the undecidability of several decidability questions has been proved. Finally, it turned out that the reversibility of an arbitrary real-time circular one-way cellular automaton is undecidable as well.

8.6 Turing Machines

Turing machines (TM) are a classical model of computation where a finite state control unit, the head, moves along a bi-infinite tape of cells, each containing a tape symbol. The head reads and writes symbols on the tape, changes its internal state, and moves to neighbouring cells at discrete time steps as instructed by a fixed transition rule, the program of the TM. A suitable choice of the program makes the machine reversible (RTM). Turing machines are traditionally viewed as language acceptors, but one can also incorporate outputs in the model so that the machine becomes a transducer that computes a (partial) function. In [12] the authors investigate RTM under the strict function semantics that requires that at the end of the computation only the output remains on the tape, and they develop a rigorous foundational theory of reversible computation of functions in this semantics, including the appropriate concept of universality and a design of a universal machine.

Turing machines with bi-infinite tape contents are also discrete dynamical systems (on a compact space) under two possible viewpoints [104]: in the moving tape view (TMT) the position of the head is fixed but the entire tape shifts left or right depending on the current instruction, while in the usual moving head view (TMH) one needs to allow configurations without a head to make the configuration space compact. In [38] the authors present a reversible TMT with the rather surprising property that it has no halting or temporally periodic configurations, thus answering positively a conjecture made in [92]. The machine, dubbed "SMART", is small (4 internal states, 3 tape symbols) and nicely symmetric in both time and space. It possesses the good dynamical properties of transitivity and minimality. The machine is further applied to settle another conjecture made in [92]: it is undecidable whether a given complete reversible Turing machine has a periodic orbit.

The class of RTM dynamical systems becomes more robust if the head is allowed to view and modify locally blocks of several tape symbols at once. In particular, compositions of machines and inverse machines are now in the same class so that reversible Turing machines with any fixed states and tape symbols form a group under composition. The structure of this group and algorithmic questions concerning the group are studied in [25]. The paper also introduces a number of natural subgroups. The model includes multidimensional Turing machines where the tape cells are indexed by $\mathbb{Z}^d$ for dimension d, and both the moving head and the moving tape viewpoints can be taken.

Finally, reversible Turing machines with a working tape and a one-way or two-way read-only input tape are considered as language recognisers [15]. In particular, the classes of languages acceptable by such devices with small time bounds in the range between real time and linear time, that is, with time bounds of the form $n+r(n)$ where $r \in o(n)$ is a sublinear function, have been considered. It has been shown that there exist infinite time hierarchies of separated complexity classes in that range. The question of whether reversible Turing machines in the range of interest are weaker than general ones or not is answered positively by proving that there are languages accepted by irreversible one-way Turing machines in real time that cannot be accepted by any reversible one-way machine in less than linear time.

9 Quantum Formal Verification and Quantum Machine Learning

Large-scale, fault-tolerant quantum computers are still under development and, despite a recent major push for "quantum supremacy" by companies like IBM, Google and Intel, it is not clear when they will become a reality. On the other hand there is much recent interest in using Noisy Intermediate Scale Quantum (NISQ) computers to provide a "quantum advantage". This involves the use of existing or near-term quantum computers to solve valuable problems, faster, cheaper, or more efficiently than any available classical solution. Potential application areas include simulation of many-body physics, quantum chemistry, optimisation and quantum machine learning. Airbus has issued its Quantum Computing Challenge to tackle aerospace flight physics problems using quantum computers. Many companies such as IBM, Microsoft, D-Wave, Rigetti and Xanadu are developing full-stack solutions for implementing quantum algorithms. This typically starts from a high-level programming language and a compiler, down to an assembly language and quantum hardware. These resources are usually accessible via the cloud. Much of these developments will need guarantees regarding security and correctness. Formal verification, which has been used successfully in classical computing for a number of years, could be extremely valuable in increasing confidence in quantum systems.

Quantum cryptography aims to overcome the limitations of classical cryptography by providing unconditional security, which is not dependent on the difficulty of inverting a particular computation. Quantum Key Distribution protocols have been implemented in commercial products by Id Quantique, MagiQ, NEC and Toshiba, amongst others, and have been used in practical applications, e.g. the Geneva election ballot count. Various QKD networks have been built, including the DARPA Quantum Network in Boston, the SeCoQC network around Vienna and the Tokyo QKD Network. China has launched a dedicated satellite "Micius" for quantum communication. On the theoretical side, quantum key distribution protocols such as BB84 [29] have been proved to be unconditionally secure. It is important to understand, however, that this is an

information-theoretic proof, which does not necessarily guarantee that *implemented systems* are unconditionally secure. This area is also where approaches such as those based on formal methods could be useful in analysing behaviour of implemented systems.

The paper [32] presents a novel framework for modelling and verifying quantum protocols and their implementations using the proof assistant Coq. It provides a Coq library for quantum bits (qubits), quantum gates, and quantum measurement. As a step towards verifying practical quantum communication and security protocols such as Quantum Key Distribution, it supports multiple qubits, communication and entanglement. These concepts are illustrated by modelling the Quantum Teleportation Protocol, which communicates the state of an unknown quantum bit using only a classical channel. In more recent work, a Quantum IO monad has been implemented in Coq for the specification of the protocols. In addition to quantum operations and measurement, the monad gives us a lightweight process calculus which supports sequencing of operations and keeping of state. This monad has the necessary properties. The process simulation function that gives the QIO monad its semantics has also been written. Current work concerns proving properties of simple quantum protocols.

In [10], the authors present CCSq, a concurrent language for describing quantum systems, and develop verification techniques for checking equivalence between CCSq processes. CCSq has well-defined operational and superoperator semantics for protocols that are functional, in the sense of computing a deterministic input-output relation for all interleavings arising from concurrency in the system. They have implemented QEC (Quantum Equivalence Checker), a tool that takes the specification and implementation of quantum protocols, described in CCSq, and automatically checks their equivalence. QEC is the first fully automatic equivalence checking tool for concurrent quantum systems. For efficiency purposes, the approach is restricted to Clifford operators in the stabiliser formalism, but it is able to verify protocols over all input states. A collection of interesting and practical quantum protocols, ranging from quantum communication and quantum cryptography to quantum error correction, have been specified and verified.

In other recent work, a version of the quantum process calculus CQP has been implemented. The implementation, which has the working title qtpi and is available from `github.com/mdxtoc/qtpi`, uses symbolic rather than numeric probability calculation. Programs are checked statically, before they run, to ensure that they obey real-world restrictions on the use of qbits (e.g. no cloning, no sharing). Qtpi has been used to simulate some simple protocols such as teleportation, and some more involved ones including QKD. It is early days in the development of the tool, but it can already simulate well over 1M qbit transfers per minute.

Quantum machine learning is the aspect of quantum computing concerned with the design of algorithms capable of generalised learning from labelled training data by effectively exploiting quantum effects. The undertaken work makes various contributions to this emerging area; in particular it has pursued the

issue of classification error within a standard quantum computational setting, and explored the congruence of Kernel Methods with the topological quantum computational setting (a congruence that will be developed further in future work).

Specifically, the following have been achieved:

In [52] the authors present a novel approach to computing Hamming distance and its kernelisation within Topological Quantum Computation. This approach is based on an encoding of two binary strings into a topological Hilbert space, whose inner product yields a natural Hamming distance kernel on the two strings. Kernelisation forges a link with the field of Machine Learning, particularly in relation to binary classifiers such as the Support Vector Machine (SVM). This makes our approach of potentially wide interest to the quantum machine learning community.

In [183], the authors set out a strategy for quantising attribute bootstrap aggregation to enable variance-resilient quantum machine learning. To do so, they utilise the linear decomposability of decision boundary parameters in the Rebentrost et al. Support Vector Machine [164] to guarantee that stochastic measurement of the output quantum state will give rise to an ensemble decision without destroying the superposition over projective feature subsets induced within the chosen SVM implementation. It achieves a linear performance advantage, $O(d)$, in addition to the existing $O(\log(n))$ advantages of quantisation as applied to Support Vector Machines. The approach extends to any form of quantum learning giving rise to linear decision boundaries.

Error-correcting output codes (ECOC) are a standard setting in machine learning for efficiently rendering the collective outputs of a binary classifier, such as the support vector machine, as a multi-class decision procedure. Appropriate choice of error-correcting codes further enables incorrect individual classification decisions to be effectively corrected in the composite output. In [182], the authors propose an appropriate quantisation of the ECOC process, based on the quantum support vector machine. They show that, in addition to the usual benefits of quantising machine learning, this technique leads to an exponential reduction in the number of logic gates required for effective correction of classification error.

10 Conclusion

We gave in the previous sections an overview of the status and recent developments of different research threads on the foundations of reversible computation. While many interesting results have been found, we notice that the field is still very heterogeneous. For instance, while process calculi, Petri nets and cellular automata are all models of concurrent systems, they come equipped with different notions of reversibility. Cellular automata are considered reversible if the global dynamics is bijective (similarly to what is done in sequential reversible models), Petri nets if reverse transitions can be added without changing the behaviour of the net, while process calculi are mainly based on the notion of causal-consistent reversibility. Some initial cross-fertilisation results came thanks

to the COST Action, e.g. there have been works applying causal-consistent reversibility to Petri nets [132] and related models [27,155]. We also remark that some of the developments described in this chapter have been instrumental to better understand reversibility in programming languages and to advance on a number of application areas, as discussed in the rest of the book.

References

1. Abramsky, S.: Retracing some paths in process algebra. In: Montanari, U., Sassone, V. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 1–17. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61604-7_44
2. Abramsky, S.: A structural approach to reversible computation. Theoret. Comput. Sci. **347**(3), 441–464 (2005)
3. Abramsky, S., Coecke, B.: A categorical semantics of quantum protocols. In: Logic in Computer Science, LICS 2004, pp. 415–425. IEEE (2004)
4. Abramsky, S., Haghverdi, E., Scott, P.: Geometry of interaction and linear combinatory algebras. Math. Struct. Comput. Sci. **12**(5), 625–665 (2002)
5. Agrigoroaiei, O., Ciobanu, G.: Dual P systems. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2008. LNCS, vol. 5391, pp. 95–107. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-95885-7_7
6. Agrigoroaiei, O., Ciobanu, G.: Reversing computation in membrane systems. J. Logic Algebraic Program. **79**(3–5), 278–288 (2010)
7. Aman, B., Ciobanu, G.: Reversibility in parallel rewriting systems. J. Univers. Comput. Sci. **23**(7), 692–703 (2017)
8. Aman, B., Ciobanu, G.: Controlled reversibility in reaction systems. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) CMC 2017. LNCS, vol. 10725, pp. 40–53. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73359-3_3
9. Angluin, D.: Inference of reversible languages. J. ACM **29**(3), 741–765 (1982)
10. Ardeshir-Larijani, E., Gay, S.J., Nagarajan, R.: Automated equivalence checking of concurrent quantum systems. ACM Trans. Comput. Logic **19**(4), 28:1–28:32 (2018)
11. Axelsen, H.B., Glück, R.: Reversible representation and manipulation of constructor terms in the heap. In: Dueck, G.W., Miller, D.M. (eds.) RC 2013. LNCS, vol. 7948, pp. 96–109. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38986-3_9
12. Axelsen, H.B., Glück, R.: On reversible turing machines and their function universality. Acta Inf. **53**(5), 509–543 (2016)
13. Axelsen, H.B., Holzer, M., Kutrib, M.: The degree of irreversibility in deterministic finite automata. Int. J. Found. Comput. Sci. **28**, 503–522 (2017)
14. Axelsen, H.B., Holzer, M., Kutrib, M., Malcher, A.: Reversible shrinking two-pushdown automata. In: Dediu, A.-H., Janoušek, J., Martín-Vide, C., Truthe, B. (eds.) LATA 2016. LNCS, vol. 9618, pp. 579–591. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30000-9_44
15. Axelsen, H.B., Jakobi, S., Kutrib, M., Malcher, A.: A hierarchy of fast reversible turing machines. In: Krivine, J., Stefani, J.-B. (eds.) RC 2015. LNCS, vol. 9138, pp. 29–44. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20860-2_2
16. Axelsen, H.B., Kutrib, M., Malcher, A., Wendlandt, M.: Boosting reversible pushdown machines by preprocessing. In: Devitt, S., Lanese, I. (eds.) RC 2016. LNCS, vol. 9720, pp. 89–104. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40578-0_6

17. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press, Cambridge (1998)
18. Bacci, G., Danos, V., Kammar, O.: On the statistical thermodynamics of reversible communicating processes. In: Corradini, A., Klin, B., Cîrstea, C. (eds.) CALCO 2011. LNCS, vol. 6859, pp. 1–18. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22944-2_1
19. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. TTCSAES. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47967-4
20. Barbanera, F., de'Liguoro, U.: A game interpretation of retractable contracts. In: Lluch Lafuente, A., Proença, J. (eds.) COORDINATION 2016. LNCS, vol. 9686, pp. 18–34. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39519-7_2
21. Barbanera, F., Dezani-Ciancaglini, M., de'Liguoro, U.: Compliance for reversible client/server interactions. In: Workshop on Behavioural Types, BEAT 2014. EPTCS, vol. 162, pp. 35–42 (2014)
22. Barbanera, F., Dezani-Ciancaglini, M., de'Liguoro, U.: Reversible client/server interactions. Formal Asp. Comput. **28**(4), 697–722 (2016)
23. Barbanera, F., Dezani-Ciancaglini, M., Lanese, I., de'Liguoro, U.: Retractable contracts. In: Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES 2015. EPTCS, vol. 203, pp. 61–72 (2015)
24. Barbanera, F., Lanese, I., de'Liguoro, U.: Retractable and speculative contracts. In: Jacquet, J.-M., Massink, M. (eds.) COORDINATION 2017. LNCS, vol. 10319, pp. 119–137. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59746-1_7
25. Barbieri, S., Kari, J., Salo, V.: The group of reversible turing machines. In: Cook, M., Neary, T. (eds.) AUTOMATA 2016. LNCS, vol. 9664, pp. 49–62. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39300-1_5
26. Barylska, K., Erofeev, E., Koutny, M., Mikulski, Ł., Piątkowski, M.: Reversing transitions in bounded Petri nets. Fund. Inf. **157**(4), 341–357 (2018)
27. Barylska, K., Gogolińska, A., Mikulski, Ł., Philippou, A., Piątkowski, M., Psara, K.: Reversing computations modelled by coloured Petri nets. In: Workshop on Algorithms & Theories for the Analysis of Event Data. CEUR Workshop Proceedings, vol. 2115, pp. 91–111. CEUR-WS.org (2018)
28. Barylska, K., Koutny, M., Mikulski, Ł., Piątkowski, M.: Reversible computation vs. reversibility in Petri nets. Sci. Comput. Program. **151**, 48–60 (2018)
29. Bennett, C.H., Brassard, G.: Quantum cryptography: public key distribution and coin tossing. In: Conference on Computers, Systems & Signal Processing, CSSP 1984, pp. 175–179 (1984)
30. Berry, G., Boudol, G.: The chemical abstract machine. Theor. Comput. Sci. **96**(1), 217–248 (1992)
31. Berstel, J.: Transductions and Context-Free Languages. Teubner, Stuttgart (1979)
32. Boender, J., Kammüller, F., Nagarajan, R.: Formalization of quantum protocols using Coq. In: Workshop on Quantum Physics and Logic, QPL 2015, pp. 71–83 (2015)
33. Bouziane, Z., Finkel, A.: Cyclic Petri net reachability sets are semi-linear effectively constructible. In: Workshop on Verification of Infinite State Systems, INFINITY 1997, ENTCS, pp. 15–24. Elsevier (1997)
34. Bowman, W.J., James, R.P., Sabry, A.: Dagger traced symmetric monoidal categories and reversible programming. In: Reversible Computation, RC 2011, pp. 51–56. Ghent University (2011)

35. Cardelli, L., Laneve, C.: Reversibility in massive concurrent systems. Sci. Ann. Comp. Sci. **21**(2), 175–198 (2011)
36. Cardelli, L., Laneve, C.: Reversible structures. In: Computational Methods in Systems Biology, CMSB 2011, pp. 131–140. ACM (2011)
37. Carothers, C.D., Perumalla, K.S., Fujimoto, R.: Efficient optimistic parallel simulations using reverse computation. ACM Trans. Model. Comput. Simul. **9**(3), 224–253 (1999)
38. Cassaigne, J., Ollinger, N., Torres-Avilés, R.: A small minimal aperiodic reversible Turing machine. J. Comput. Syst. Sci. **84**, 288–301 (2017)
39. Clavel, M., et al.: Maude: specification and programming in rewriting logic. Theor. Comput. Sci. **285**(2), 187–243 (2002)
40. Cockett, J.R.B., Lack, S.: Restriction categories I: categories of partial maps. Theoret. Comput. Sci. **270**(1–2), 223–259 (2002)
41. Colange, M., Baarir, S., Kordon, F., Thierry-Mieg, Y.: Crocodile: a symbolic/symbolic tool for the analysis of symmetric nets with bag. In: Kristensen, L.M., Petrucci, L. (eds.) PETRI NETS 2011. LNCS, vol. 6709, pp. 338–347. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21834-7_20
42. Cristescu, I., Krivine, J., Varacca, D.: A compositional semantics for the reversible π-calculus. In: Logic in Computer Science, LICS 2013, pp. 388–397. IEEE Computer Society (2013)
43. Cservenka, M.H., Glück, R., Haulund, T., Mogensen, T.Æ.: Data structures and dynamic memory management in reversible languages. In: Kari, J., Ulidowski, I. (eds.) RC 2018. LNCS, vol. 11106, pp. 269–285. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99498-7_19
44. Danos, V., Krivine, J.: Reversible communicating systems. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 292–307. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28644-8_19
45. Danos, V., Krivine, J.: Transactions in RCCS. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 398–412. Springer, Heidelberg (2005). https://doi.org/10.1007/11539452_31
46. Danos, V., Krivine, J.: Formal molecular biology done in CCS-R. In: Workshop on Concurrent Models in Molecular Biology, BioConcur 2003, vol. 180(3) (2003). Electr. Notes Theor. Comput. Sci., 31–49. Elsevier (2007)
47. Dartois, L., Fournier, P., Jecker, I., Lhote, N.: On reversible transducers. In: International Colloquium on Automata, Languages, and Programming, ICALP 2017. LIPIcs, vol. 80, pp. 113:1–113:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
48. de Frutos Escrig, D., Koutny, M., Mikulski, Ł.: An efficient characterization of Petri net solvable binary words. In: Khomenko, V., Roux, O.H. (eds.) PETRI NETS 2018. LNCS, vol. 10877, pp. 207–226. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91268-4_11
49. de Frutos Escrig, D., Koutny, M., Mikulski, Ł.: Reversing steps in Petri nets. In: Donatelli, S., Haar, S. (eds.) PETRI NETS 2019. LNCS, vol. 11522, pp. 171–191. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21571-2_11
50. De Nicola, R., Hennessy, M.: Testing equivalences for processes. Theor. Comput. Sci. **34**, 83–133 (1984)
51. Delacourt, M., Ollinger, N.: Permutive one-way cellular automata and the finiteness problem for automaton groups. In: Kari, J., Manea, F., Petre, I. (eds.) CiE 2017. LNCS, vol. 10307, pp. 234–245. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58741-7_23

52. Di Pierro, A., Mengoni, R., Nagarajan, R., Windridge, D.: Hamming distance kernelisation via topological quantum computation. In: Martín-Vide, C., Neruda, R., Vega-Rodríguez, M.A. (eds.) TPNC 2017. LNCS, vol. 10687, pp. 269–280. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71069-3_21
53. Ehrenfeucht, A., Rozenberg, G.: Reaction systems. Fund. Inf. **75**(1), 263–280 (2007)
54. Esparza, J., Nielsen, M.: Decidability issues for Petri nets. BRICS Rep. Ser. **1**(8) (1994)
55. Foster, N., Matsuda, K., Voigtländer, J.: Three complementary approaches to bidirectional programming. In: Gibbons, J. (ed.) Generic and Indexed Programming. LNCS, vol. 7470, pp. 1–46. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32202-0_1
56. Giachino, E., Lanese, I., Mezzina, C.A., Tiezzi, F.: Causal-consistent reversibility in a tuple-based language. In: Parallel, Distributed, and Network-Based Processing, PDP 2015, pp. 467–475. IEEE Computer Society (2015)
57. Giachino, E., Lanese, I., Mezzina, C.A., Tiezzi, F.: Causal-consistent rollback in a tuple-based language. J. Log. Algebr. Meth. Program. **88**, 99–120 (2017)
58. Giles, B.G.: An investigation of some theoretical aspects of reversible computing. Ph.D. thesis, University of Calgary (2014)
59. Glück, R., Kaarsgaard, R.: A categorical foundation for structured reversible flowchart languages. In: Mathematical Foundations of Programming Semantics, MFPS 2018. Electronic Notes in Theoretical Computer Science, vol. 341, pp. 155–171. Elsevier (2018)
60. Glück, R., Kaarsgaard, R.: A categorical foundation for structured reversible flowchart languages: soundness and adequacy. Logical Methods Comput. Sci. **14**(3) (2018)
61. Glück, R., Kaarsgaard, R., Yokoyama, T.: Reversible programs have reversible semantics. In: Reversibility in Programming, Languages, and Automata, RPLA 2019. Lecture Notes in Computer Science. Springer (2019, to appear)
62. Glück, R., Yokoyama, T.: A linear-time self-interpreter of a reversible imperative language. Comput. Soft. **33**(3), 108–128 (2016)
63. Glück, R., Yokoyama, T.: A minimalist's reversible while language. IEICE Trans. Inf. Syst. **E100–D**(5), 1026–1034 (2017)
64. Glück, R., Yokoyama, T.: Constructing a binary tree from its traversals by reversible recursion and iteration. Inf. Process. Lett. **147**, 32–37 (2019)
65. Graversen, E., Phillips, I., Yoshida, N.: Towards a categorical representation of reversible event structures. J. Logical Algebraic Methods Program. **104**, 16–59 (2019)
66. Guillon, B., Kutrib, M., Malcher, A., Prigioniero, L.: Reversible pushdown transducers. In: Hoshi, M., Seki, S. (eds.) DLT 2018. LNCS, vol. 11088, pp. 354–365. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98654-8_29
67. Guo, X.: Products, joins, meets, and ranges in restriction categories. Ph.D. thesis, University of Calgary (2012)
68. Haulund, T., Mogensen, T.Æ., Glück, R.: Implementing reversible object-oriented language features on reversible machines. In: Phillips, I., Rahaman, H. (eds.) RC 2017. LNCS, vol. 10301, pp. 66–73. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59936-6_5
69. Hedlund, G.A.: Endomorphisms and automorphisms of the shift dynamical systems. Mathe. Syst. Theor. **3**(4), 320–375 (1969)

70. Heunen, C., Kaarsgaard, R., Karvonen, M.: Reversible effects as inverse arrows. In: Mathematical Foundations of Programming Semantics, MFPS XXXIV. Electronic Notes in Theoretical Computer Science, vol. 341, pp. 179–199. Elsevier (2018)
71. Heunen, C., Karvonen, M.: Monads on dagger categories. Theor. Appl. Categories **31**, 1016–1043 (2016)
72. Hoey, J., Ulidowski, I., Yuen, S.: Reversing parallel programs with blocks and procedures. In: Expressiveness in Concurrency/Structural Operational Semantics. Electronic Proceedings in Theoretical Computer Science, vol. 276, pp. 69–86 (2018)
73. Holzer, M., Jakobi, S., Kutrib, M.: Minimal reversible deterministic finite automata. Int. J. Found. Comput. Sci. **29**(2), 251–270 (2018)
74. Holzer, M., Kutrib, M.: Reversible nondeterministic finite automata. In: Phillips, I., Rahaman, H. (eds.) RC 2017. LNCS, vol. 10301, pp. 35–51. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59936-6_3
75. Hu, Z., Schürr, A., Stevens, P., Terwilliger, J.F.: Bidirectional transformation "bx" (Dagstuhl Seminar 11031). Dagstuhl Reports **1**(1), 42–67 (2011). http://drops.dagstuhl.de/volltexte/2011/3144/
76. Hullot, J.-M.: Canonical forms and unification. In: Bibel, W., Kowalski, R. (eds.) CADE 1980. LNCS, vol. 87, pp. 318–334. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10009-1_25
77. Hüttel, H., et al.: Foundations of session types and behavioural contracts. ACM Comput. Surv. **49**(1), 3:1–3:36 (2016)
78. European COST Action IC1405 on "Reversible Computation - Extending Horizons of Computing". http://www.revcomp.eu/
79. Jacobs, B.: New directions in categorical logic, for classical, probabilistic and quantum logic. Logical Methods Comput. Sci. **11**(3), 1–76 (2015)
80. Jacobsen, P.A.H., Kaarsgaard, R., Thomsen, M.K.: CoreFun: a typed functional reversible core language. In: Kari, J., Ulidowski, I. (eds.) RC 2018. LNCS, vol. 11106, pp. 304–321. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99498-7_21
81. Jalonen, J., Kari, J.: Conjugacy of one-dimensional one-sided cellular automata is undecidable. In: Tjoa, A.M., Bellatreche, L., Biffl, S., van Leeuwen, J., Wiedermann, J. (eds.) SOFSEM 2018. LNCS, vol. 10706, pp. 227–238. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73117-9_16
82. James, R.P., Sabry, A.: Theseus: a high level language for reversible computing. In: Work-in-Progress Report Presented at RC 2014. http://www.cs.indiana.edu/~sabry/papers/theseus.pdf
83. James, R.P., Sabry, A.: Information effects. ACM SIGPLAN Not. **47**(1), 73–84 (2012)
84. Jones, N.D.: Computability and Complexity: From a Programming Language Perspective. Foundations of Computing. MIT Press, Cambridge (1997)
85. Joyal, A., Street, R., Verity, D.: Traced monoidal categories. Math. Proc. Cambridge Philos. Soc. **119**(3), 447–468 (1996)
86. Kaarsgaard, R., Axelsen, H.B., Glück, R.: Join inverse categories and reversible recursion. J. Logical Algebraic Methods Program. **87**, 33–50 (2017)
87. Kaarsgaard, R., Glück, R.: A categorical foundation for structured reversible flowchart languages: soundness and adequacy. Logical Methods Comput. Sci. **14**(3), 1–38 (2018)
88. Kari, J.: Reversibility of 2D cellular automata is undecidable. Physica D **45**(1), 379–385 (1990)

89. Kari, J.: Universal pattern generation by cellular automata. Theoret. Comput. Sci. **429**, 180–184 (2012)
90. Kari, J.: Reversible cellular automata: from fundamental classical results to recent developments. New Generation Comput. **36**(3), 145–172 (2018)
91. Kari, J., Kopra, J.: Cellular automata and powers of p/q. RAIRO - Theor. Inf. Applic. **51**(4), 191–204 (2017)
92. Kari, J., Ollinger, N.: Periodicity and immortality in reversible computing. In: Ochmański, E., Tyszkiewicz, J. (eds.) MFCS 2008. LNCS, vol. 5162, pp. 419–430. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85238-4_34
93. Kari, J., Salo, V., Worsch, T.: Sequentializing cellular automata. In: Baetens, J.M., Kutrib, M. (eds.) AUTOMATA 2018. LNCS, vol. 10875, pp. 72–87. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92675-9_6
94. Karvonen, M.: The way of the dagger. Ph.D. thesis, School of Informatics, University of Edinburgh (2019)
95. Kastl, J.: Inverse categories. In: Algebraische Modelle, Kategorien und Gruppoide. Studien zur Algebra und ihre Anwendungen, vol. 7, pp. 51–60. Akademie-Verlag (1979)
96. Kawabe, M., Glück, R.: The program inverter LRinv and its structure. In: Hermenegildo, M.V., Cabeza, D. (eds.) PADL 2005. LNCS, vol. 3350, pp. 219–234. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30557-6_17
97. Keller, R.: Towards a theory of universal speed-independent modules. IEEE Trans. Comput. **23**(1), 21–33 (1974)
98. Klop, J.W.: Term rewriting systems. In: Abramsky, S., Gabbay, D.M., Maibaum, T.S.E. (eds.) Handbook of Logic in Computer Science, vol. I, pp. 1–112. Oxford University Press (1992)
99. Knowlton, K.C.: A fast storage allocator. Commun. ACM **8**(10), 623–625 (1965)
100. Kopra, J.: Glider automorphisms on some shifts of finite type and a finitary Ryan's theorem. In: Baetens, J.M., Kutrib, M. (eds.) AUTOMATA 2018. LNCS, vol. 10875, pp. 88–99. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92675-9_7
101. Krivine, J.: A verification technique for reversible process algebra. In: Glück, R., Yokoyama, T. (eds.) RC 2012. LNCS, vol. 7581, pp. 204–217. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36315-3_17
102. Kuhn, S., Ulidowski, I.: A calculus for local reversibility. In: Devitt, S., Lanese, I. (eds.) RC 2016. LNCS, vol. 9720, pp. 20–35. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40578-0_2
103. Kuhn, S., Ulidowski, I.: Local reversibility in a calculus of covalent bonding. Sci. Comput. Program. **151**, 18–47 (2018)
104. Kurka, P.: On topological dynamics of Turing machines. Theor. Comput. Sci. **174**(1–2), 203–216 (1997)
105. Kutrib, M.: Reversible and irreversible computations of deterministic finite-state devices. In: Italiano, G.F., Pighizzini, G., Sannella, D.T. (eds.) MFCS 2015. LNCS, vol. 9234, pp. 38–52. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48057-1_3
106. Kutrib, M., Malcher, A.: Fast reversible language recognition using cellular automata. Inf. Comput. **206**, 1142–1151 (2008)
107. Kutrib, M., Malcher, A.: Reversible pushdown automata. J. Comput. Syst. Sci. **78**, 1814–1827 (2012)

108. Kutrib, M., Malcher, A., Wendlandt, M.: Real-time reversible one-way cellular automata. In: Isokawa, T., Imai, K., Matsui, N., Peper, F., Umeo, H. (eds.) AUTOMATA 2014. LNCS, vol. 8996, pp. 56–69. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18812-6_5
109. Kutrib, M., Malcher, A., Wendlandt, M.: Reversible queue automata. Fund. Inf. **148**, 341–368 (2016)
110. Kutrib, M., Malcher, A., Wendlandt, M.: When input-driven pushdown automata meet reversiblity. RAIRO - Theor. Inf. Applic. **50**, 313–330 (2016)
111. Kutrib, M., Malcher, A., Wendlandt, M.: Transducing reversibly with finite state machines. Theor. Comput. Sci. **787**, 111–126 (2019)
112. Kutrib, M., Wendlandt, M.: Reversible limited automata. Fund. Inf. **155**, 31–58 (2017)
113. Landauer, R.: Irreversibility and heat generated in the computing process. IBM J. Res. Dev. **5**, 183–191 (1961)
114. Lanese, I., Lienhardt, M., Mezzina, C.A., Schmitt, A., Stefani, J.-B.: Concurrent flexible reversibility. In: Felleisen, M., Gardner, P. (eds.) ESOP 2013. LNCS, vol. 7792, pp. 370–390. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37036-6_21
115. Lanese, I., Medic, D., Mezzina, C.A.: Static versus dynamic reversibility in CCS. Acta Informatica (2019)
116. Lanese, I., Mezzina, C.A., Schmitt, A., Stefani, J.-B.: Controlling reversibility in higher-order Pi. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 297–311. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23217-6_20
117. Lanese, I., Mezzina, C.A., Stefani, J.-B.: Reversing higher-order Pi. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 478–493. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15375-4_33
118. Lanese, I., Mezzina, C.A., Stefani, J.-B.: Controlled reversibility and compensations. In: Glück, R., Yokoyama, T. (eds.) RC 2012. LNCS, vol. 7581, pp. 233–240. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36315-3_19
119. Lanese, I., Mezzina, C.A., Stefani, J.-B.: Reversibility in the higher-order π-calculus. Theor. Comput. Sci. **625**, 25–84 (2016)
120. Lanese, I., Mezzina, C.A., Tiezzi, F.: Causal-consistent reversibility. Bull. EATCS **114** (2014)
121. Lanese, I., Nishida, N., Palacios, A., Vidal, G.: CauDEr: a causal-consistent reversible debugger for Erlang. In: Gallagher, J.P., Sulzmann, M. (eds.) FLOPS 2018. LNCS, vol. 10818, pp. 247–263. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-90686-7_16
122. Laursen, J.S., Schultz, U.P., Ellekilde, L.: Automatic error recovery in robot assembly operations using reverse execution. In: Intelligent Robots and Systems, IROS 2015, pp. 1785–1792. IEEE (2015)
123. Lavado, G.J., Pighizzini, G., Prigioniero, L.: Minimal and reduced reversible automata. J. Automata, Lang. Comb. **22**(1–3), 145–168 (2017)
124. Lavado, G.J., Pighizzini, G., Prigioniero, L.: Weakly and strongly irreversible regular languages. In: Automata and Formal Languages, AFL 2017. EPTCS, vol. 252, pp. 143–156 (2017)
125. Lavado, G.J., Prigioniero, L.: Concise representations of reversible automata. Int. J. Found. Comput. Sci. **30**(6–7), 1157–1175 (2019)

126. Lienhardt, M., Lanese, I., Mezzina, C.A., Stefani, J.-B.: A reversible abstract machine and its space overhead. In: Giese, H., Rosu, G. (eds.) FMOODS/FORTE -2012. LNCS, vol. 7273, pp. 1–17. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30793-5_1
127. Mahler, K.: An unsolved problem on the powers of 3/2. J. Australian Math. Soc. **8**(2), 313–321 (1968)
128. Matsuda, K., Hu, Z., Nakano, K., Hamana, M., Takeichi, M.: Bidirectionalization transformation based on automatic derivation of view complement functions. In: International Conference on Functional Programming, ICFP 2007, pp. 47–58. ACM (2007)
129. McNellis, J., Mola, J., Sykes, K.: Time travel debugging: root causing bugs in commercial scale software. CppCon talk (2017). https://www.youtube.com/watch?v=l1YJTg_A914
130. Medić, D., Mezzina, C.A.: Static VS dynamic reversibility in CCS. In: Devitt, S., Lanese, I. (eds.) RC 2016. LNCS, vol. 9720, pp. 36–51. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40578-0_3
131. Medic, D., Mezzina, C.A., Phillips, I., Yoshida, N.: A parametric framework for reversible pi-calculi. In: Workshop on Expressiveness in Concurrency and Workshop on Structural Operational Semantics, EXPRESS/SOS 2018. EPTCS, vol. 276, pp. 87–103 (2018)
132. Melgratti, H., Mezzina, C.A., Ulidowski, I.: Reversing P/T Nets. In: Riis Nielson, H., Tuosto, E. (eds.) COORDINATION 2019. LNCS, vol. 11533, pp. 19–36. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22397-7_2
133. Mezzina, C.A.: On reversibility and broadcast. In: Kari, J., Ulidowski, I. (eds.) RC 2018. LNCS, vol. 11106, pp. 67–83. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99498-7_5
134. Mezzina, C.A., et al.: Software and reversible systems: a survey of recent activities. In: Ulidowski, I., et al. (eds.) Reversible Computation. LNCS 12070, pp. 41–59. Springer, Cham (2020)
135. Mezzina, C.A., Koutavas, V.: A safety and liveness theory for total reversibility. In: Theoretical Aspects of Software Engineering, TASE 2017, pp. 1–8. IEEE Computer Society (2017)
136. Mezzina, C.A., Pérez, J.A.: Reversible sessions using monitors. In: Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES 2016. EPTCS, vol. 211, pp. 56–64 (2016)
137. Mezzina, C.A., Pérez, J.A.: Causally consistent reversible choreographies: a monitors-as-memories approach. In: Principles and Practice of Declarative Programming, PPDP 2017, pp. 127–138. ACM (2017)
138. Mezzina, C.A., Pérez, J.A.: Reversibility in session-based concurrency: a fresh look. J. Log. Algebr. Meth. Program. **90**, 2–30 (2017)
139. Mikulski, Ł., Lanese, I.: Reversing unbounded Petri nets. In: Donatelli, S., Haar, S. (eds.) PETRI NETS 2019. LNCS, vol. 11522, pp. 213–233. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21571-2_13
140. Milner, R.: A Calculus of Communicating Systems. LNCS, vol. 92. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10235-3
141. Mogensen, T.Æ.: RSSA: a reversible SSA form. In: Mazzara, M., Voronkov, A. (eds.) PSI 2015. LNCS, vol. 9609, pp. 203–217. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41579-6_16
142. Mogensen, T.Æ.: Reversible garbage collection for reversible functional languages. New Gener. Compu. **36**(3), 203–232 (2018)

143. Morita, K.: Theory of Reversible Computing. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Tokyo (2017). https://doi.org/10.1007/978-4-431-56606-9
144. Morita, K., Harao, M.: Computation universality of one-dimensional reversible (injective) cellular automata. IEICE Trans. **E72**(6), 758–762 (1989)
145. Morrison, D., Ulidowski, I.: Direction-reversible self-timed cellular automata for delay-insensitive circuits. J. Cellular Automata **12**(1–2), 101–120 (2016)
146. Mousavi, M.R., Reniers, M.A., Groote, J.F.: SOS formats and meta-theory: 20 years after. Theor. Comput. Sci. **373**(3), 238–272 (2007)
147. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
148. Nishida, N., Palacios, A., Vidal, G.: Reversible term rewriting. In: Formal Structures for Computation and Deduction, FSCD 2016. LIPIcs, vol. 52. pp. 28:1–28:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016)
149. Nishida, N., Palacios, A., Vidal, G.: A reversible semantics for Erlang. In: Hermenegildo, M.V., Lopez-Garcia, P. (eds.) LOPSTR 2016. LNCS, vol. 10184, pp. 259–274. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63139-4_15
150. Nishida, N., Palacios, A., Vidal, G.: Reversible computation in term rewriting. J. Log. Algebr. Meth. Program. **94**, 128–149 (2018)
151. Nishida, N., Vidal, G.: Program inversion for tail recursive functions. In: Rewriting Techniques and Applications, RTA 2011. LIPIcs, vol. 10, pp. 283–298. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2011)
152. Nishida, N., Vidal, G.: Characterizing compatible view updates in syntactic bidirectionalization. In: Thomsen, M.K., Soeken, M. (eds.) RC 2019. LNCS, vol. 11497, pp. 67–83. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21500-2_5
153. Paolini, L., Piccolo, M., Roversi, L.: A certified study of a reversible programming language. In: Types for Proofs and Programs, TYPES 2018. LIPIcs, vol. 69, pp. 7:1–7:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018)
154. Păun, G.: Computing with membranes. J. Comput. Syst. Sci. **61**(1), 108–143 (2000)
155. Philippou, A., Psara, K.: Reversible computation in Petri nets. In: Kari, J., Ulidowski, I. (eds.) RC 2018. LNCS, vol. 11106, pp. 84–101. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99498-7_6
156. Philippou, A., Psara, K., Siljak, H.: Controlling reversibility in reversing Petri nets with application to wireless communications. In: Thomsen, M.K., Soeken, M. (eds.) RC 2019. LNCS, vol. 11497, pp. 238–245. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21500-2_15
157. Phillips, I., Ulidowski, I.: Reversibility and asymmetric conflict in event structures. J. Log. Algebr. Meth. Program. **84**(6), 781–805 (2015)
158. Phillips, I., Ulidowski, I., Yuen, S.: A reversible process calculus and the modelling of the ERK signalling pathway. In: Glück, R., Yokoyama, T. (eds.) RC 2012. LNCS, vol. 7581, pp. 218–232. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36315-3_18
159. Phillips, I., Ulidowski, I., Yuen, S.: Modelling of bonding with processes and events. In: Dueck, G.W., Miller, D.M. (eds.) RC 2013. LNCS, vol. 7948, pp. 141–154. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38986-3_12
160. Phillips, I., Ulidowski, I.: Reversing algebraic process calculi. In: Aceto, L., Ingólfsdóttir, A. (eds.) FoSSaCS 2006. LNCS, vol. 3921, pp. 246–260. Springer, Heidelberg (2006). https://doi.org/10.1007/11690634_17

161. Phillips, I.C.C., Ulidowski, I.: Reversing algebraic process calculi. J. Log. Algebr. Program. **73**(1–2), 70–96 (2007)
162. Pin, J.-E.: On reversible automata. In: Simon, I. (ed.) LATIN 1992. LNCS, vol. 583, pp. 401–416. Springer, Heidelberg (1992). https://doi.org/10.1007/BFb0023844
163. Pinna, G.M.: Reversing steps in membrane systems computations. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) CMC 2017. LNCS, vol. 10725, pp. 245–261. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73359-3_16
164. Rebentrost, P., Mohseni, M., Lloyd, S.: Quantum support vector machine for big data classification. Phys. Rev. Lett. **113**, 130503 (2014)
165. Reisig, W.: Petri Nets: An Introduction. EATCS Monographs on Theoretical Computer Science, vol. 4. Springer, Heidelberg (1985). https://doi.org/10.1007/978-3-642-69968-9
166. Rensink, A., Vogler, W.: Fair testing. Inf. Comput. **205**(2), 125–198 (2007)
167. Sabry, A., Valiron, B., Vizzotto, J.K.: From symmetric pattern-matching to quantum control. In: Baier, C., Dal Lago, U. (eds.) FoSSaCS 2018. LNCS, vol. 10803, pp. 348–364. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89366-2_19
168. Salo, V.: Groups and monoids of cellular automata. In: Kari, J. (ed.) AUTOMATA 2015. LNCS, vol. 9099, pp. 17–45. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47221-7_3
169. Salo, V., Törmä, I.: A one-dimensional physically universal cellular automaton. In: Kari, J., Manea, F., Petre, I. (eds.) CiE 2017. LNCS, vol. 10307, pp. 375–386. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58741-7_35
170. Schaeffer, L.: A physically universal cellular automaton. In: Innovations in Theoretical Computer Science, ITCS 2015, pp. 237–246. ACM (2015)
171. Schordan, M., Oppelstrup, T., Jefferson, D., Barnes Jr., P.D.: Generation of reversible C++ code for optimistic parallel discrete event simulation. New Gener. Comput. **36**(3), 257–280 (2018)
172. Schordan, M., Oppelstrup, T., Thomsen, M.K., Glück, R.: Reversible languages and incremental state saving in optimistic parallel discrete event simulation. In: Ulidowski, I., et al. (eds.) Reversible Computation. LNCS 12070, pp. 187–207. Springer, Cham (2020)
173. Schultz, U.P., Axelsen, H.B.: Elements of a reversible object-oriented language. In: Devitt, S., Lanese, I. (eds.) RC 2016. LNCS, vol. 9720, pp. 153–159. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40578-0_10
174. Selinger, P.: Dagger compact closed categories and completely positive maps. In: Workshop on Quantum Programming Languages, QPL 2005. Electronic Notes in Theoretical Computer Science, vol. 170, pp. 139–163 (2005)
175. Selinger, P.: A survey of graphical languages for monoidal categories. New Structures for Physics. Lecture Notes in Physics, vol. 813, pp. 289–355. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-12821-9_4
176. Slagle, J.R.: Automated theorem-proving for theories with simplifiers, commutativity and associativity. J. ACM **21**(4), 622–642 (1974)
177. Sutner, K.: De Bruijn graphs and linear cellular automata. Complex Syst. **5**(1), 19–30 (1991)
178. Terese: Term Rewriting Systems, Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press (2003)
179. Tiezzi, F., Yoshida, N.: Reversible session-based pi-calculus. J. Log. Algebr. Meth. Program. **84**(5), 684–707 (2015)

180. Toffoli, T.: Computation and construction universality of reversible cellular automata. J. Comput. Syst. Sci. **15**(2), 213–231 (1977)
181. Toffoli, T., Margolus, N.: Cellular Automata Machines: A New Environment for Modeling. MIT Press, Cambridge (1987)
182. Windridge, D., Mengoni, R., Nagarajan, R.: Quantum error-correcting output codes. Int. J. Quantum Inf. **16**(8), 1840003 (2018)
183. Windridge, D., Nagarajan, R.: Quantum bootstrap aggregation. In: de Barros, J.A., Coecke, B., Pothos, E. (eds.) QI 2016. LNCS, vol. 10106, pp. 115–121. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52289-0_9
184. Yokoyama, T., Axelsen, H.B., Glück, R.: Towards a reversible functional language. In: De Vos, A., Wille, R. (eds.) RC 2011. LNCS, vol. 7165, pp. 14–29. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29517-1_2
185. Yokoyama, T., Axelsen, H.B., Glück, R.: Fundamentals of reversible flowchart languages. Theoret. Comput. Sci. **611**, 87–115 (2016)