# Some Notes About Reversibility

**Abstract:** A typical computer user knows the difference between what can be undone on a computer, and what cannot. They may be familiar with the "undo" feature of text editors but understand the impossibility of recovering an unsaved document after an emergency shutdown. Creating programs guaranteeing that any action can be undone requires to design and implement reversible programming languages. While such languages come with interesting built-in security features (because any past action can be investigated), they also raise challenges when it comes to concurrency. Indeed, undoing an action that involved synchronization between multiple actors requires all actors to agree to undo said action. This talk offers to discuss current trends in solving the aforementioned problem, and to highlight some of the benefits that could result from well-designed concurrent and reversible programming languages.

## Intro

A computer generally transforms an *input* into an *output*:

| Input | Program | Output |
|---|---|---|
| `html` document | pandoc | `pdf` document |
| $\sqrt{16}$ | calculator | 4 |
| `wav` file | vlc | `flac` file |
| A folder | zip | A zipped folder |
| The pair $(1, 2)$ | projection | 1 |
| A message and a private key | gpg | An encrypted message |

A *reversible* computer is a computer that accepts only programs that can conversely transform the output back into an input. This can be enforced either at the hardware level (the computer *itself* can only perform operations that can be reversed) or at the software level (the program must be capable of being executed both ways). Note that while computer that are reversible at the hardware level can *only* execute reversible programs, "traditional" computers can still execute reversible programs by 1. compiling the source code into two binaries (one corresponding to the forward execution, one to the backward execution), 2. executing both binaries forward.

If pandoc, the calculator or vlc were implemented as reversible programs, then we could get our `html` document back from a `pdf` document, compute the square root of 4, and convert into `wav` from flac "for free", since providing a reversible program from input to output implies that executing it in reverse goes from output to input. Of course, reversing a projection operation sees impossible (information is *lost*), and being able to retrieve the private key from an encrypted message actually sounds dangerous!

"Benett's trick" gives a way of reversing any program, essentially by carrying a copy of the input through the computation: this is highly inefficient, but illustrates that *in principle* any program can be made reversible. However, we are interested in discussing a finer distinction between programs that *should* be reversible (typically, a lossless conversion between formats) and those that *should not* be reversible (any program involving encryption, for instance).

*(Give example of reversible instruction?)*

## Benefits of Reversible Programming Languages

### From a Software-Engineering Position

As we stated, having to write only one source code to obtain a program that can convert from `html` to `pdf` and back is a massive gain of time.

### From a Verification Perspective

Once we obtain a zip program that can unzip by simply executing it the other way around, we also know that it is correct *by design*: no need to prove that zipping then unzipping is the identity, it is enforced by the program itself!

Writing a proper file archiver (like 7-Zip, Ark or TAR) requires to:

- write a program to compress files into an archive,
- write a program to extract files from an archive,
- gain evidence that compressing then extracting files gives back the original files.

If that same program is written using a *reversible* programming language, then

- compressing means executing it forward,
- extracting means executing it backward,
- that compressing then extracting is the identity comes for free!

Another example is pandoc, or, actually, any program that performs lossless compression (frow `wav` to `flac`, but also in cryptography).

*(Discuss benefits from finding out what can be and cannot be reversed?)*

### From a Security Perspective

- Debugging
- Data + program => decision, if we can reverse this this mean that we always know what triggered which decision.
- Built-in forensic.

### From the Hardware

**Low-Energy Computation** https://en.wikipedia.org/wiki/Landauer%27s_principle

Plus verification empirically?

**Quantum Computing** Definition of unitary operation. Is it just on the surface that quantum computation is reversible?

**Bio-computing (chemical reactions)**

**Mixed Systems** Remember that very much like GPU, we could have reversible components embedded in larger device, for particular operations.

## Brief Presentation of Reversible Computation

Distinguish between causal consistent and non-causal consistent. Refer to various surveys about "flavors" of reversible computation.

## Why Is Reversibility of Interest?

- It is connected to quantum computing,
- It is related to bio-computing
- It may open the possibility of low-energy computers,
- It will simplify the development and proof of correctness of some programs,
- It opens new perspective theoretically.

## References