Course Paper
Implementation of a Turing simulator

Jan Engelhardt

March 17, 2005

Max-Plank-Gymnasium Göttingen
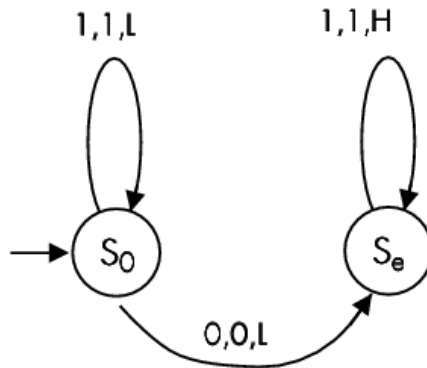Advances CS
Stufe 12 / Abiturjahrgang 2006

Contents

Figure 1: Example for a transition/state graph

1 Introduction

1.1 Limits of today's Turing simulators

There exist a number of commonly used methods to define state tables, also called automaton tables. This includes [static] arrays[1], structures or functions statically put into the executable.

The biggest disadvantage with these methods is that parts of the source code need to be recompiled upon code change, which usually requires the installation of a compiler (or even a complete development environment) if this has not happened yet[2]. If the source code is not even available, the limits of possibilities are already hit.

To circumvent the latter problem, compiled structures / functions could be dynamically post-loaded at runtime. Let alone that not all programming languages support shared libraries[3], this will also require a compiler (or interpreter).

The two starting points satisfy the needs for e.g. short-lived demonstration programs — if one needs more functionality, s/he is redirected to the Internet. As long as programs with static state tables do not comprise the majority on the Internet, the most graphical Turing simulators still got a table or other input methods to define the operations for the Turing machine.

But even these input-table based simulators are not scalable to infinity: with an increasing number of states, a table view becomes more and more unclear, and if you do not have a helper view next to you, you might just give up on searching for the correct state soon enough. This of course also concerns above-mentioned array-based automaton tables.

1.2 Project goal

To solve these problems (by working around them in a smart way), and to generally develop "a different Turing simulator" which is different from the ones present on the Internet, a state graph based view is considerable. Such a simulator has the advantage that states (respectively

---

[1]For an example, see **(author?)** [StatArray]

[2]Unix and Linux systems usually install it by default — or at least have it on the same CD / CD set.

[3]The known files with the `.so` or (even more known) `.dll` extensions.

the geometric shapes with which they are symbolized) can be moved around free-minded in two dimensions, which is not the case with table entries.

In this paper, an own Turing simulator with integrated "Designer" will be developed. The development of the graphical interface will require the most work compared to the data structures for the state graph and the Turing machine, but which all this is inevident.

The finalized Turing simulator is released under the GNU Lesser/Library General Public License**(author?)** [LGPL] and can be downloaded from `http://inai.de/`**(author?)** [jengelh].

2 Basics

2.1 Requirements to the simulator

The primary goal of the simulator is, as already elaborated on, that the definition and display of states is done by drawing them rather than entering the values in a table. The secondary goal will be the chaining of Turing automatons.

Elementary functions, which are necessary for operating the simulator, consist of the possibility to control the Turing machine manually, but also automatically by a clock. Also, it should be possible to save and load [state] graphs, which is also necessary for chaining.

Some optional extra functions only have a slightly higher implementation time compared to them being left out. For example, a CPU preferably executes operations on bytes (and multibytes) rather than single bits[4], and most programming languages follow this principle[5]. It is therefore recommendable that the Turing tape is byte-based rather bit-based. This also enlarges the capacity of the input alphabet.

Two other items that are important for me are, for one, the portability, and for two, the free availability of the code outside of this paper.

2.2 Realization

A graphical interface will be needed for the designer, as described in chapter 3.1. I use **(author?)** [wxWidgets] for this, a portable graphics library which changes its appearance depending on the underlying graphic set. Apart from that I do not know the APIs of **(author?)** [GTK] or **(author?)** [Qt], WX has the most experience in the Other World (Win32). WX is with it since its beginning 1992 **(author?)** [wxIntro], GTK since 1998 **(author?)** [GtkW32] and Qt is expected to be released in mid-2005 **(author?)** [QtW32]. Furthermore, GTK is written in C, which I do not like to use in graphics programming because the lack of object oriented design. Qt (C++) has shown itself to be a heavyweight (when I used **(author?)** [KDE] in the past) in respect to size, memory usage and speed.

---

[4]To change a bit, at least a whole must be read and written back (`var = var ^ 4`)

[5]In addition to that, C also knows bitfields, which work like elementary data types (`fourth_bit = 1`), but which also turns into the CPU instruction "`xor byte ptr [var], 4`" after compilation.

The simulator however, has almost no requirements and could also be implemented as a text-based CLI[6]. Designing a CUI[7] would firstly outgrow the time frame, and secondly, a Turing simulator is not that useful for the main users of CLIs/ CUIs, mainly server administrators. Since there is already a graphic part because of the Designer, the simulator is also implemented in the GUI. The data structures however, will be designed in a way so that a CLI simulator could be added to the source tree without problems, or that other non-graphical applications can use this interface. **(author?)** [libxml] is recommended as a parser loadable files, since I already used it for earlier projects.

The question for the programming language is almost implicitly resolved after the core components that will be used, have been introduced. Java is dropped at the first instant because of lack of flexibility in regard to data structure management[8]. wxWidgets does not provide less than two external bindings for Perl and Python, but my choice is C++, since it is fully compiled[9] and because there is polymorphism[10], overloading and templates in C++. Especially inheritance contributes to shorter code for the invented data structures, compared to when they would be done in ANSI C.

2.3   Comparison with other Turing simulators

The first two chapters already indicated that there are mostly table-based simulators. **(author?)** [DMOZ] has some entries for Alan Turing, but not Turing simulators. Thus, we visit the start page of a named search engine, **(author?)** [Google], and query it about "Turing simulator". The **(author?)** [Google Directory] provides about the same URLs.

On the first result page, **(author?)** [Tril], is a Java applet where the programming is done through a text window (lower left), which is even more error-prone than tables, since you can loose track not only between the lines, but also between the comma-separated fields. The simulator provides a few static examples for demonstration purposes. Single step and automatic run with speed control are implemented. Saving and loading own machine programs is not possible due to the nature of web applications[11].

On another page, **(author?)** [Bertol], we can find another Java applet that implements a certain kind of automaton — according to the URI and its operating behavior, this must be a Busy Beaver[12]. This simulator is also limited in his abilities — probably because it is only a demo version. Only debug functionality ("single" button) is available, reprogramming is not.

These two simulators are actually incomparable with my project, since they are one of these "short-lived" programs from courses (judging by their appearance and functionality) and are not suited for complex tasks. Quite different are the following two simulators which have moved up to page 1 in Google's ranking (sorta Top Ten) whilst working on this paper. One of them is the MPG-Turing-Simulator **(author?)** [MPGTS], a textgraphical (CUI) simulator for DOS16. The menu-driven handling is very intuitive and can also be found out without problems, without actually reading

---

[6]Command Line Interface, see **(author?)** [FourUI].

[7]Textgraphical Command(-line) User Interface, see **(author?)** [FourUI].

[8]Only values and object pointers exist.

[9]Java and Perl may create so-called bytecode, but which is still interpreted.

[10]Use of a derived class where a base class is expected

[11]That would be an ideal case — **(author?)** [mskb]

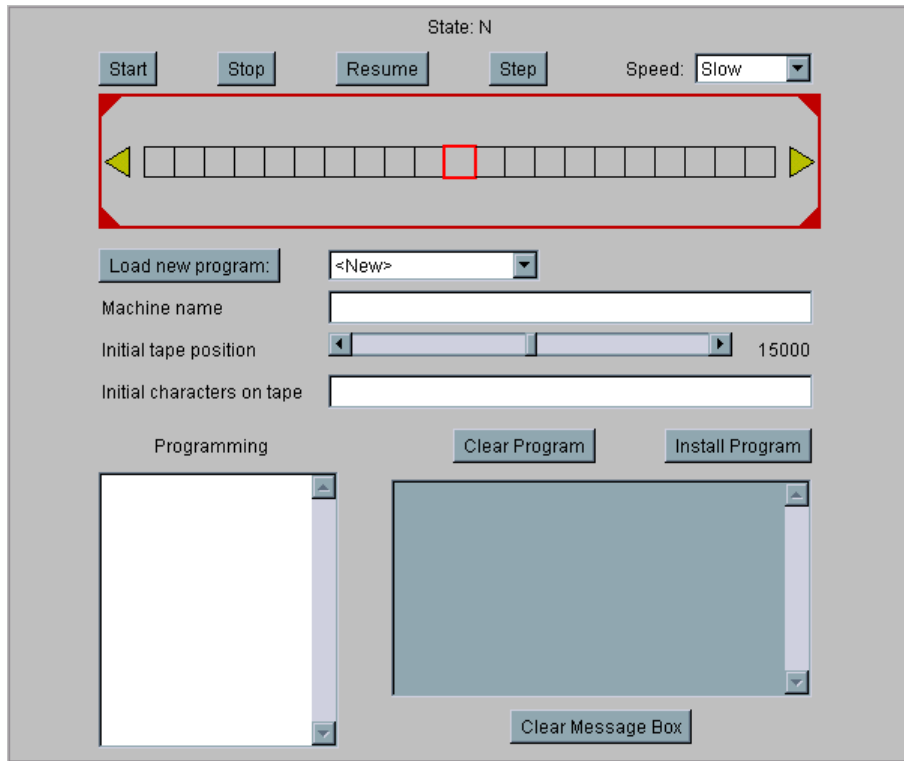[12]Explanation to beavers in **(author?)** [VlinTuring]
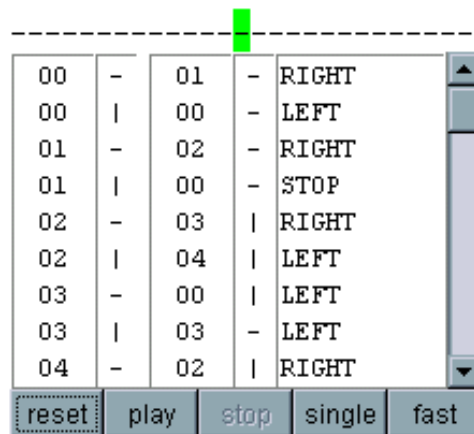
Figure 2: Tril's Turing simulator

Figure 3: "Online Turing-Simulator"

```
┌─ VMware Workstation ─────────────────────────────────────────────────────┐
│ File  Edit  View  VM  Power  Snapshot  Windows  Help                      │
├───────────────────────────────────────────────────────────────────────────┤
│  MPG-Turing-Simulator :    Summe.TM              Version 3.2    5.1997     │
│                                                                            │
│     <B>and ändern              <S>peed: 5              <L>ernmodus: ein     │
│                                                                            │
│  8            9              0               1                             │
│                             I I I I I   I I I                              │
│                                    ▲                                       │
│  Ort:      0              <Z>ustand: 000/004    Schritt:      0/      0    │
│                                                                            │
│  ┌── <T>uringtafel ──────┐ ┌──────────── Alphabet ──────────────┐         │
│  │                       │ │ I                                   │         │
│  │  Z000 'I' :  ' ' R Z001│ │ Joker:  ?                          │         │
│  │                       │ │                                     │         │
│  │  Z000 'I' :  ' ' R Z001│ │                                     │         │
│  │  Z001 ' ' :  'I' L Z002│ │                                     │         │
│  │  Z001 'I' :  'I' R Z001│ │ <▲> Vorwärtslauf        <D>  Diskoperationen│
│  │  Z002 ' ' :  ' ' R Z003│ │ <►> Einzelschritt       <N>  Neue Maschine  │
│  │  Z002 'I' :  'I' L Z002│ │ <◄> Einzelschritt zurück <A>  Ausgabe (Tafel)│
│  │  Z003 'I' :  'I' S Z003│ │ <▼> Rücklauf            <F1> Beschreibung   │
│  │                       │ │ <W> Wiederholen         <F2> Farben ändern  │
│  │                       │ │ <F> Fast Run            <M>  Markiere Befehl │
│  │                       │ │ <J> Jump bis *          <PgDn> weitere Befehle│
│  │                       │ │             <Q> Quit                 │         │
│  └───────────────────────┘ └─────────────────────────────────────┘         │
└───────────────────────────────────────────────────────────────────────────┘
```
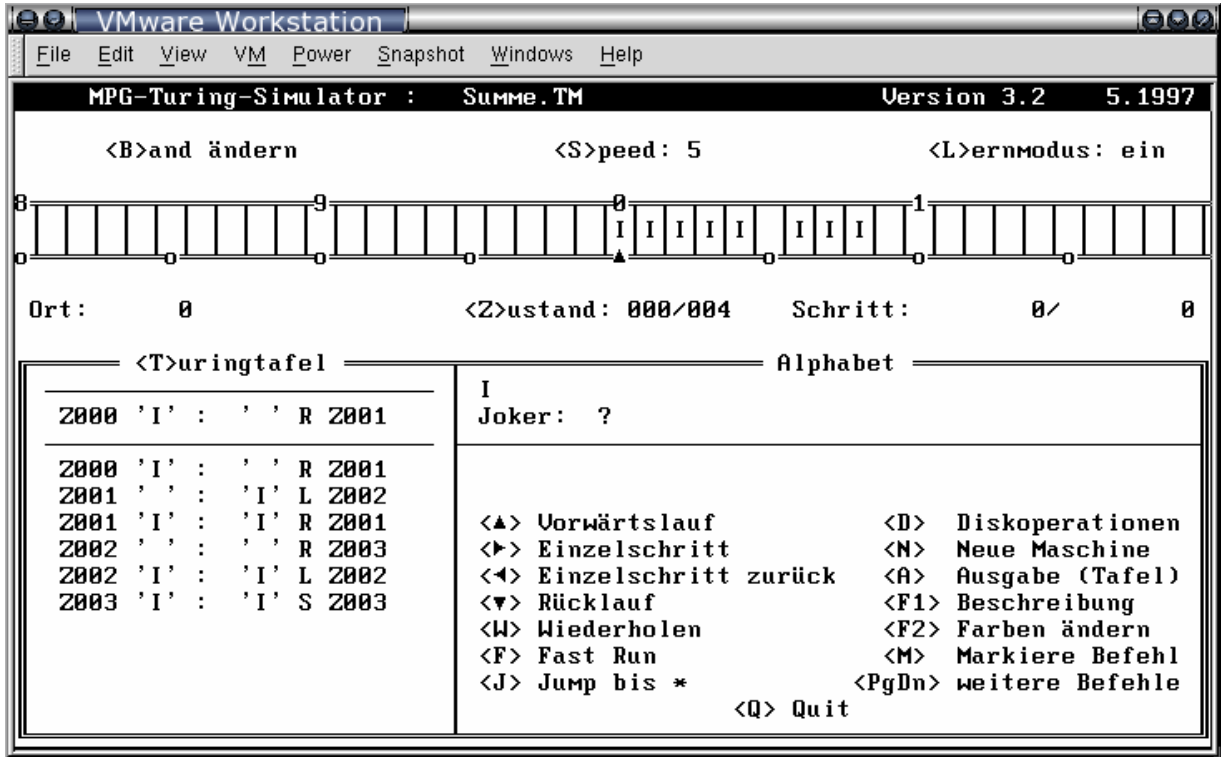
Figure 4: MPG-Turing-Simulator

the README file. (Preconditioned that one knows what a Turing simulator expects.) It does not contain any static programs, but comes with a lot (132) of example/s [files], which even go beyond the bits taught in Grade 12. Even multiple tapes can be inserted, but the core element still remains a table — but which is the best possible option with this kind of interface.

The last compared simulator, Visual Turing **(author?)** [VisTuring], also tries to display states and actions graphically, similar to my imagination, but does so in a very unusual way. My interpretation of the interface says that a step consists of ⟨move, read, write⟩ (in this order) in Visual Turing, which is a substantial difference to the graph shown in figure 1, which is arranged as ⟨read, write, move⟩ Automatic and manual tick are possible, saving and loading graphs is no problem, the debug controls also include backward stepping. There are behavior similarities to S-, I- and LNodes; chaining of automatons is also possible, and the GUI is able to "dive" into them and show the chained automaton's graph. The simulator however, is a deception package, because it has 26 variables, in which chars can be written to and read from independently of the state.

3 Implementation

3.1 User interface

The first question that comes up is what kind of interface type to use, and if the respective time complexity is justified or can be done in the given timeslice. Eventually, the choice came down to a GUI. The main reason is that the text console is simply not suited for a *designer* — a drawing
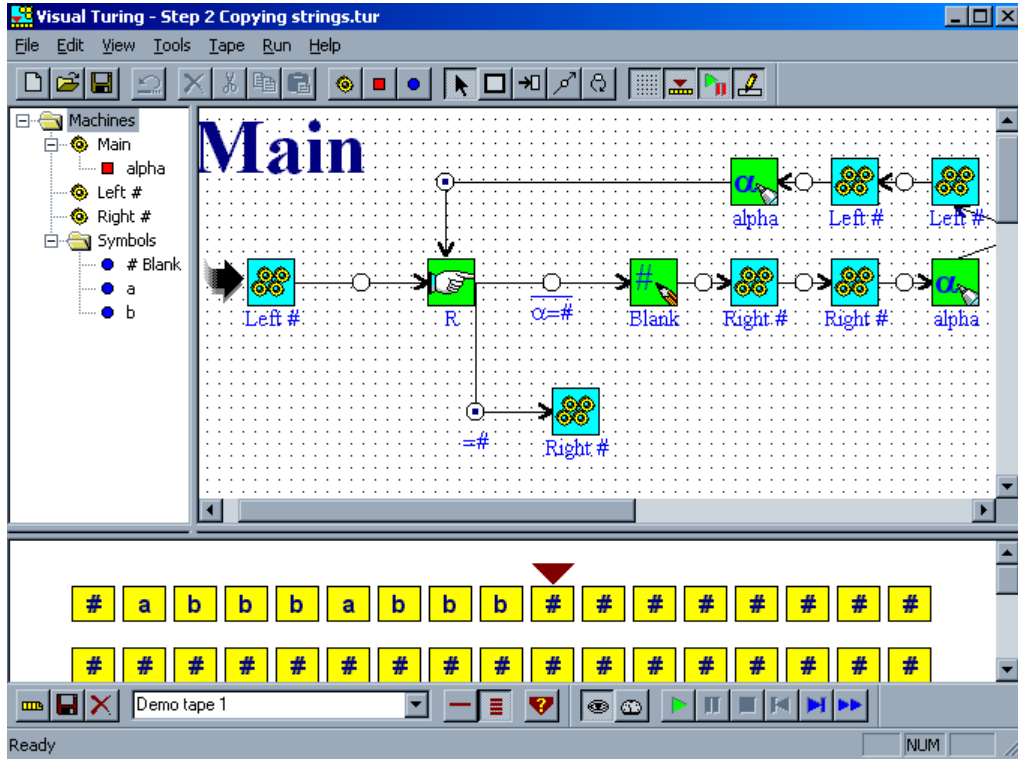
Figure 5: Visual Turing

environment. With a height / width of 80x25 (max. 132x60), it is simply too small for a state graph. A circle with sequence number is at least 3x3 cells in size.

Right at the beginning, scrollbars are implemented into the GUI to allow for a little more space, since already a simple graph like `adder.xml` fills the viewport if the application window is not resized. The virtual size[13] for the drawpad[14] is set to 4096x4096 pixels, but can be extended without further changes in the code or stored files (`de_pad.cpp`, `DA_WIDTH`).

3.2   Data structures for the graph

A typical state graph consists of states (shown as circles) and arrows for the possible ways of handling. The class TGraph (`gr_tgrap.cpp`) represents such a graph. The states are stored in an array, which is the only way, since transition functions must either be statically compiled in, or require a complex parser if they were supplied by the user or loaded from file. An (explicit) initial state should of course not be missing, but is not required from the later-described class TuMach.

Since there will be more than just plain states, the term node is introduced, which covers about everything present in a graph. For the different types of nodes, an abstract base class is required so that polymorphism and virtual calls[15] work.

The most important node type is the SNode (state node), which represents a state. It is displayed

---

[13]Total size, including temporarily invisible areas

[14]Name for the white drawing area in the Designer

[15]Calling a function of a derived class even when the base class is used.

as a red circle on the drawpad. Multiple arrows can originate from an SNode, i.e. it can have many successor states. The table in which these successor states are stored (jump table) is deployed using an A+B/ RB-Tree[16] from **(author?)** [libHX]. The input states of an SNode are each assigned a class ATarget which stores ⟨Output char, Movement, Successor state⟩.

As for the chaining of Turing automatons, a parallel class to SNode would be required, which, instead of a jump table, has exactly one pointer to another graph. For this purpose, the INode is introduced, and stands (in this context[17]) for "import node". It is displayed as a blue rectangle within the GUI. Since flow control is not terminated when reaching the final state in the imported, but continues in the parent graph, there must also exist a successor state for the INode.

To split an arrow in two and add a connection piece, so that it can "go (a)round the corner" (required even for simple automaton graphs), we simply use a node for the detour rather than inventing some arrow-specific data structure. An LNode (link node) serves alone the purpose of having a pointer to the next node.

3.3 Data structures for the Turing machine

Since Turing machines do not require much, do not have registers and their sole memory is their state, they can be implemented in few code lines. The header file `tu_mach.hpp`, which is kind of a table of content for the class methods, is very short. Effectively, it only needs two functions required for the elementary operation of a Turing machine of the class `TuMach`: `tick()` and `reset()`, to do a single step and reset the machine back to the initial state, respectively.

Though, the `tick()` function in `tu_mach.cpp` is a little longer than two fulls of screen pages (of each 25 rows). This is dependent on the different implementation of the state tables — in an earlier Turing simulator of mine, which pulls its data from statically compiled state tables, the `tick()` function is just two lines long. What makes the `TuMach::tick()` this long, is due to reading/writing from/to the tape, which was laid out in other function sin the old simulator. The recursive execution of INodes also requires a few lines, as well as the fact, that there is not one, but three node types in the Turing Designer. Coding style and comments add to that.

The class `TuTape` (`tu_tape.cpp`) represents a byte tape, which extends to infinity to the right and is only limited by the capacity of your computer. Usually, a read/write head belongs to a Turing machine, but by the principle of object-oriented programming, the head does not read from the tape, but the tape is directed to read itself [and return that]. This will become irrelevant during programming anyway: Incurious if you now use `tape[5]` (with `const char *tape`) or `tape->read(5)` (with `struct tape *` or a class), the *call* to it must be in the Turing machine code.

3.4 Data structures in the GUI

With the introduction of a GUI, further components are necessary to the already present structures. This includes, for example, coordinates and appearance of a node.

---

[16]Short for associative-array-style binary red black tree; this is a modified RB binary tree which also stores value pairs rather than just values; similar to Perl's hashes.

[17]This abbreviation stands for "information node" in the Unix world and describes the content and metadata such as permissions of a file.
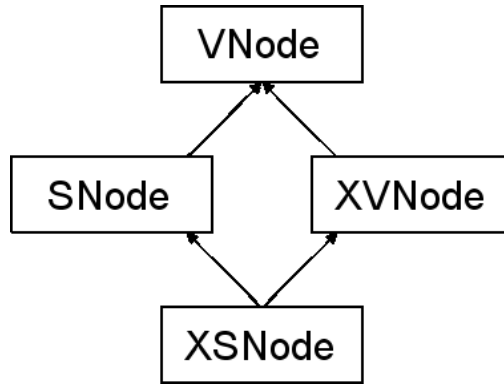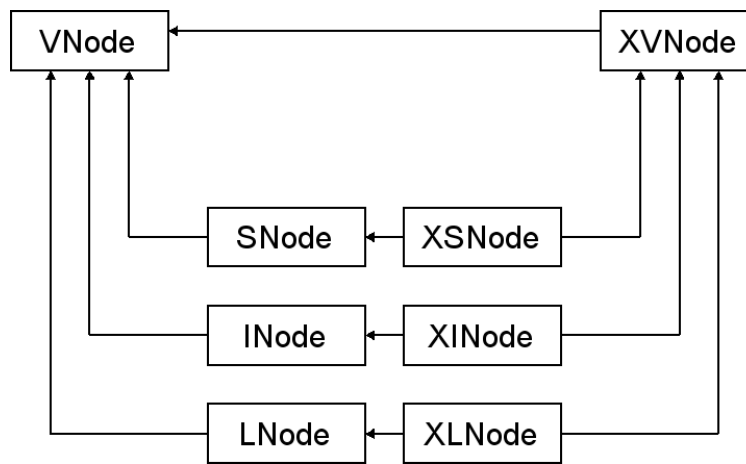
Figure 6: The Dreaded Diamond

Figure 7: Inheritance between the different node types

First, a generic graphic node XVNode (`de_data.hpp`) is required, which derives from VNode and adds graphic-specific data. From XVNode, three further classes will be derived, one for each of S-, I- and L-Nodes, which will have the properties of XVNode as well as, e.g. SNode. Concluding, XSNode must inherit from both SNode and XVNode[18]. The problem of the "dreaded diamond" **(author?)** [CFL], where a base class, VNode in this case, is inherited more than once, can be solved by adding one more keyword to the code. (This does not pose any problems, but occasional criticism is unjustified **(author?)** [CFL, 25.3].) The complete inheritance graph is shown in figure 7.

As for the GUI components of the Turing tape, I have once again made use of multiple inheritance. On the one hand, the class SimTape (`si_tape.cpp`) should have the same properties as a TuTape, and on top of that, shall instantly display changes in the GUI. The other way would be to delegate the methods to the parent or a sibling via a pointer. This however, is not necessary. SimTape just inherits from wxPanel as a second base, which makes it a packable graphic object (a so-called widget).

The class Simulator (`si_sim.cpp`) is just a packable container, like MainWindow (`xf_main.cpp`), for all the widgets and extra functions that are not provided or executed by a Turing machine itself. This includes the clock and e.g. "only X steps" ("<steps" button) and "until state X".

---

[18]A mechanism similar to Java's "implements xyz" would cause more effort in C.
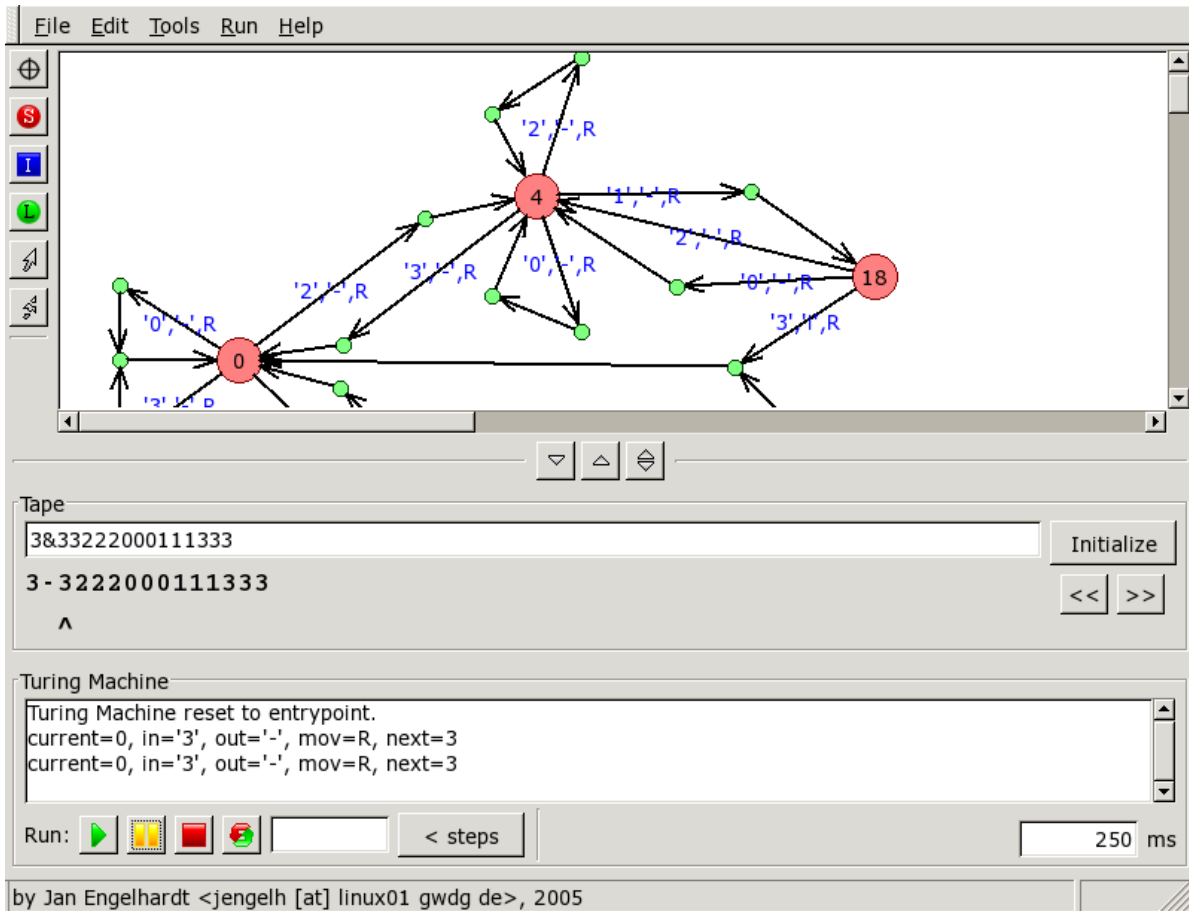
Figure 8: Turing Designer, v0.50 / March 14 2005

Finally, a screenshot of the finished Turing Designer:

4 Examples

4.1 Adder

A designer has mostly the advantage that one can insert and move objects as known, or at least, would do. This is not any different in the Turing Designer. After the core elements — ranging from nodes to targets — have been described, it should not be hard to reproduce the graph shown in figure 9 in the Turing Designer. If it is, open `adder.xml` over File ▷ Open to get at the graph. It is about an adder which operates on unary coded numbers **(author?)** [VlinTuring, pg.3] — even if the Turing Designer works on full bytes (usually 8 bits) and that a unary coding would look rather sparse.

Enter, say, "0011101111&100"[19] (2 and 4) and click on "Initialize", to "insert" a new tape. By clicking on the yellow pause button, a tick is generated, the green run button autogenerates ticks.

---

[19]`&` sets the starting position of the head. `&` thus is effectively removed from the input alphabet — which is meaningful, since `&` is used for other things in the graphic functions of wxWidgets (and others).
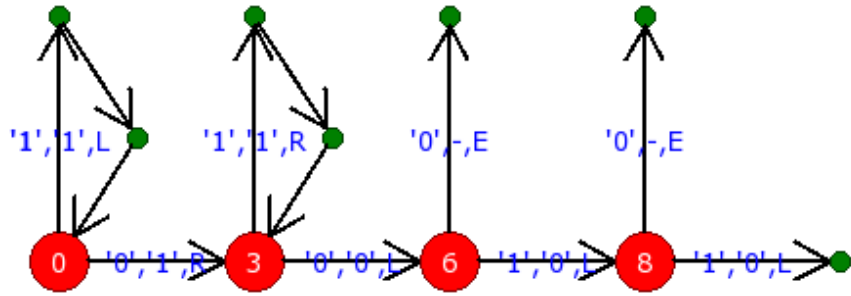
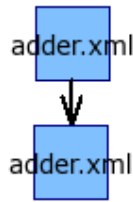Figure 9: Transition graph for the unary adder

Figure 10: Chained adder

The pause between two automatic ticks can be set in the input field in the lower right. For debugging purposes, the current state plus the next action is shown between the ticks. The adder would therefore do the following steps:

Turing tape/head before MOVE:	Turing Machine reset to entrypoint.
001101100	current=0, in='1', out='1', mov=L, next=0
001101100	current=0, in='1', out='1', mov=L, next=0
001101100	current=0, in='0', out='1', mov=R, next=3
001111100	current=3, in='1', out='1', mov=R, next=3
001111100	current=3, in='1', out='1', mov=R, next=3
001111100	current=3, in='0', out='0', mov=L, next=6
001111100	current=6, in='1', out='-', mov=L, next=8
001111000	current=8, in='1', out='0', mov=L, next=10
001110000	Reached exit gate. Machine completed.
	Turing Machine reset to entrypoint.

In the left column, the Turing tape is displayed as it would be before executing the `tick`() function. The byte cell over which the head is hovering is displayed with underline in this document. In the right column, the debug output can be seen as they would happen in the different states.

The movement and the successor state that would be executed with the next call to tick(). This is the important thing with debuggers – errors can be identified one step earlier, data structures be changed and a restart of the debug process is not necessary.

4.2 Chained adder

To build up an example chained adder, only two INodes are required, each of which imports adder.xml as their graph (cf. figure 10). Initialize the tape with three unary coded number groups, e.g. "001101101&100", and let the machine run/step. The result will be shown in one step — "00111&100".

The execution of this graph is already finished after one tick. This is because the tick() function tail-recurses into itself with the successor state if the successor node is not an SNode. Effectively, this makes paths between two SNodes execute in one tick. This is especially useful for LNodes, which only serve the purpose of being a detour. It also applies to INodes, which are thus transparently executed in the run of an tick. With some magic values for the input and output chars, you can immediately achieve an addition if the input char matches a target, for example. Furthermore, the GUI cannot currently "dive" into imported graphs, i.e. show the imported graph from adder.xml when stepping onto an INode, it makes little sense to stop on INodes.

5 Retrospect

Even if the Turing Designer is a functional simulator and fulfills its requirements, there are still some impossible or unrealized things that could not have been implemented in the given time.

For example, you cannot do a single step ("step") in debugging like it is possible with (author?) [VisTuring]. The imported automaton is executed atomically, which is more like a "next".

A generalization of the graph interface (gr_data.hpp), so that it can be used for any kind of automaton, not only for Turing machines, is not possible due to different parameters of automatons. A Turing machine for example would need to store ⟨input char, output char, movement, successor state⟩, while a BCD validator would need ⟨input char, output string, successor state⟩. The common elements are very few so that no coherent class could be built.

Finally, the table display is not available as an alternative in the Turing Designer. Even though I put it as the main problem (and implementation thus would be contradictorily), it would be nice to have multiple ways of input. At the latest when import and export of state graphs is possible, functions for table-like display is necessary.

The set of high-value Turing simulators, to which I count (author?) [MPGTS], (author?) [VisTuring] and also my own Turing Designer, seems to be low. Possible alternatives are only the oodles of Java applets, which have the described problems. What remains are some in-between programs for DOS or Windows. I therefore think that the development of another Turing simulator — this time under Linux — has filled a gap.

References

[Bertol]   Java applet demonstrating a Busy Beaver
           Michael Bertol, Holger Peterson, Horst Prote, August 1996
           http://www.fmi.uni-stuttgart.de/ti/personen/Bertol/beaver/bbb.

html and Turing.html `http://www.fmi.uni-stuttgart.de/ti/personen/Bertol/beaver/Turing.html`

[CFL] C++ FAQ Lite
Marshall Cline, 1999 - today
`http://parashift.com/c++-faq-lite/multiple-inheritance.html#faq-25.8`

[DMOZ] Web catalog of the Open Directory Project
`http://dmoz.org/`

[FourUI] The four kinds of UIs Categorization of programs with user I/O into four types
Jan Engelhardt, 2005
`http://jengelh.hopto.org/coding/uitypes.php`

[Google] Google – Search engine with PageRanking method
`http://google.com/`

[Google Directory] Google's webcatalog directory
`http://directory.google.com/`

[GTK] GIMP Toolkit
Peter Mattis, Spencer Kimball, Josh MacDonald et al
`http://gtk.org/`

[Gtk@wiki] [Gtk@wiki] Wikipedia article for GTK
various authors / no date: repeatedly updated
`http://en.wikipedia.org/wiki/GTK`

[GtkW32] First message about a Win32 port on the GTK mainling list
Tor Lillqvist, August 1998
`http://mail.gnome.org/archives/gtk-devel-list/1998-August/msg00089.html`

[jengelh] jengelh's site – personal homepage of Jan Engelhardt
`http://inai.de/`

[KDE] K Desktop Environment
Matthias Ettrich et al, 1996–today
`http://kde.org/`

[LGPL] [LGPL] GNU Lesser/Library General Public License (version 2.1)
The Free Software Foundation, February 1999
`http://gnu.org/licenses/lgpl.html`

[libHX] [libHX] General purpose library for daily usage
Jan Engelhardt, 1999–today
`http://libhx.sf.net/`

[libxml] [libxml] The XML C parser and toolkit
Daniel Veillard
`http://xmlsoft.org/`

[MPGTS] MPG-Turing-Simulator
Ulrich Mayr, Max-Plank-Gymnasium Trier/Österreich, 1988, '89, '94, '97, 2002
`http://hsg.region-kaiserslautern.de/faecher/inf/material/`
`berechenbar/turing/simulator/mpg/index.php`

[mskb] Java Security Issue Allows Access to ActiveX
Bug report by Microsoft TechNet, June 16 2004
`http://support.microsoft.com/default.aspx?scid=kb;en-us;`
`275609&sd=tech`

[ncurses] New Curses, Screen handling and optimization package
Library for textgraphical drawing
Zeyd M. Ben-Halim, Eric S. Raymond, Thomas E. Dickey (based on pcurses by Pavel Curtis)
`http://gnu.org/software/ncurses/`

[perldata] Perl Data Types - Manpage
various authors / no date: repeatedly updated
`http://perl.com/doc/manual/html/pod/perldata.html`

[Qt] Qt graphics library
Trolltech AS Norway
`http://trolltech.com/products/qt/`

[QtW32] Qt 4.0 Announcement, February 07 2005
Trolltech AS Norway
`http://www.trolltech.com/newsroom/announcements/00000192.html`

[StatArray] Example to static arrays (as part of this paper; not contained in this HTML/PDF version)

[Tril] Trils Turing Simulator
Suzanne Skinner, unknown date
`http://ironphoenix.org/tril/tm/`

[VisTuring] Visual Turing IDE
Christian Cheran, February 2001
`http://cheransoft.com/vturing/`

[VlinTuring] Turing machines
Hans-Georg Beckmann, 2003
`http://vlin.de/material_2/Turingmaschinen.pdf`
pg. 3: unary coding
pg. 36: busy beavers

[wxWidgets] Cross-platform widget library
             Julian Smart, 1992–today
             `http://wxwidgets.org/`

   [wxAPI]   General presentation to wxWidgets
             Slide 8: Layer model of wxWidgets
             Julian Smart, 2003
             `http://wxwidgets.org/fosdem2003/html/talk/img7.html`

  [wxIntro] Introduction to wxWidgets, incl. history
             various authors / no date: repeatedly updated
             `http://wxwidgets.org/intro.htm`

List of figures