Time and Space Measures for a Complete Graph Computation Model

Brian Courtehoute   Detlef Plump

Department of Computer Science
University of York

GCM, 6 July 2022

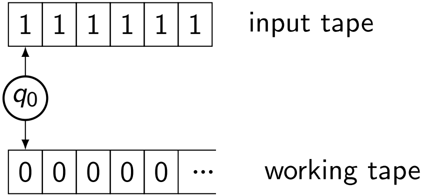
Space Compression

- Kolmogorov-Uspenskii machines (KUMs, 1958) and Schönhage's Storage Modification machines (SMMs, 1980): invented as graph-based Turing-complete models of computation
- Van Emde Boas (1989): A Turing machine of space complexity $\mathrm{O}(s(n) \log s(n))$ can be simulated by an SMM or KUM in space $\mathrm{O}(s(n))$ with quadratic time overhead, where $s(n)$ is an arbitrary function.
- We replicate this result in a Turing-complete model using a subset of GP 2.

GP 2 vs. SMMs

GP 2	*SMMs*
computes relations on graphs	compute relations on strings
pattern-based transformation rules that allow for high-level graph programming	limited set of low-level pointer instructions on a specific set of graphs
structured programs (in the sense of Dijkstra)	branching and "go to" statements
comprehensive theory based on double-pushout graph transformation	no comparable theory (we are aware of)

Off-Line Deterministic Turing Machines



- Tape alphabet: $\{0, 1, 2\}$, where 0 is the blank symbol
- Read-only finite input tape, and one-way infite working tape
- Time complexity: maximum number of transitions for a given input tape size
- Space complexity: number of working tape squares used

Example: A Turing Machine

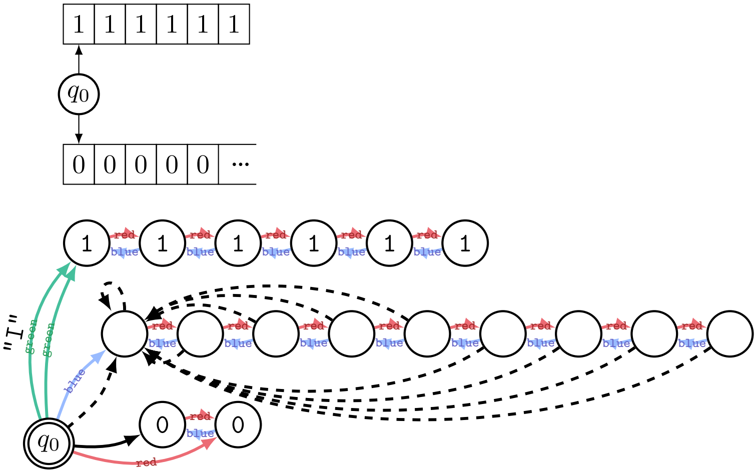
Specification:

- Input: number n represented in unary
- Output: n copies of n in binary on the working tape

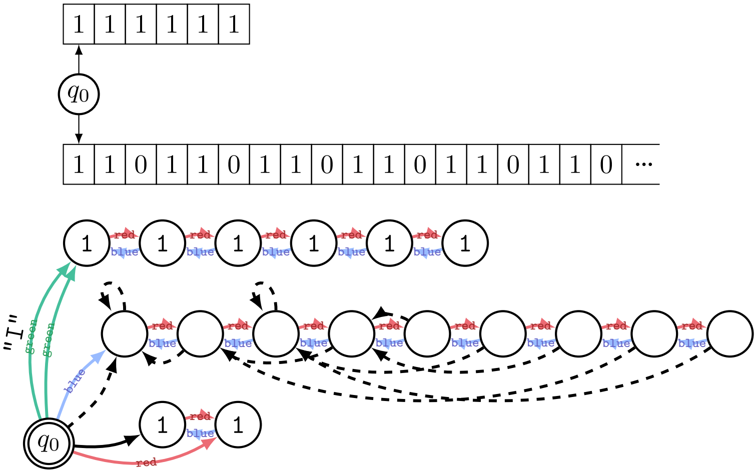
Behaviour of the machine:

- The input is copied in unary onto the working tape for use as a counter
- A binary number to the right of the counter is incremented while traversing the input
- The previous step is repeated while decrementing the counter until it reaches 0
- Tape contents need to be shifted, and 2 can be used for marking tape squares

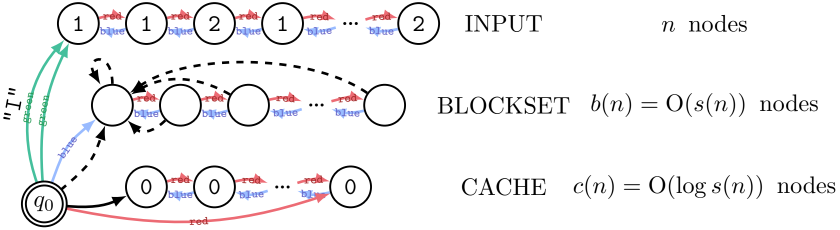
Example: Input



Example: Output

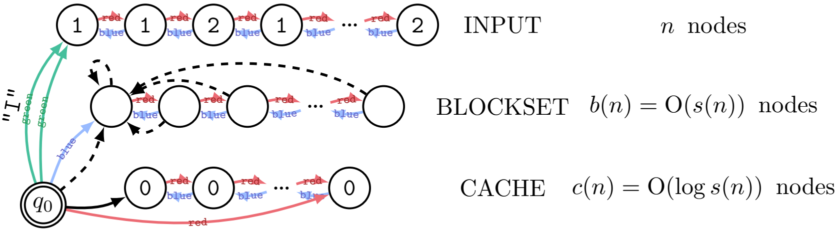


Compressed Turing Machine Configurations in GP 2



- Input tape: INPUT
- Working tape: BLOCKSET and CACHE (active tape section)
- Turing machine space: $\mathrm{O}(s(n) \log s(n))$
- Graph space: $\mathrm{O}(s(n) + \log s(n)) = \mathrm{O}(s(n))$

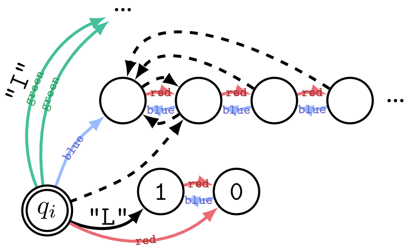
The Turing Machine Simulation



- Tape operations done in CACHE until tape head moves out of bounds, then change blocks
- If working tape too small, restart simulation with a bigger tape

Example: Changing Blocks (1)

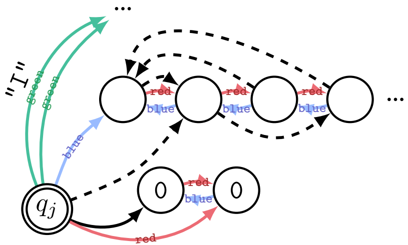
Initially: working tape is “0100 . . . ”, tape head on third square
Next move: write “1” and move tape head left



- Left node in CACHE is relabelled “1”
- Tape head labelled "L" to indicate leftward move

Example: Changing Blocks (2)

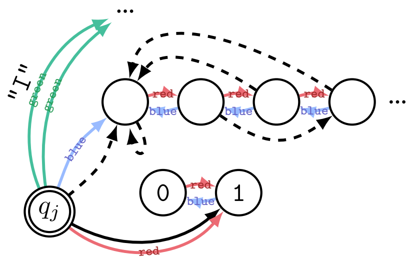
Content of CACHE saved into BLOCKSET



- Active block node: node targeted by dashed edge from node labelled q_j
- CACHE is decremented as a ternary number, while target of dashed edge originating in the active block node is moved to the right

Example: Changing Blocks (3)

Content of new block loaded into CACHE



- Leftmost node becomes new active block node
- While dashed edge originating in the active block node is moved to the left, CACHE is incremented as a ternary number

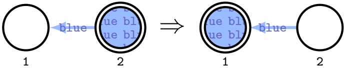
Efficient Rule Matching

- Matching a graph L in a graph G can take $\text{size}(G)^{\text{size}(L)}$ time, which is polynomial
- Constant-time matching is needed

Efficient Rule Matching

A rule is *fast* if each node in the left-hand side is reachable from a root (special node that can be accessed in constant time)

Example of a fast rule:



Theorem: Fast Rule Matching

Matching can be implemented to run in constant time for fast rules, using host graphs with a bounded outdegree that contain a bounded number of roots.

Note: Time is that of a standard machine model such as the RAM (we assume reasonable runtimes for graph data structure operations)

Efficient Backtracking

Expensive backtracking of subcomputations:

- Changes made by critical subprograms (loop bodies and conditions of `if` and `try` statements) may need to be undone
- Undoing can be time- or space-intensive, and so is preferably avoided

Expensive backtracking of nondeterminism:

- Finite sets of rules are called nondeterministically, and rule matching is nondeterministic
- To avoid the cost of backtracking, one can avoid nondeterminism

Efficient Backtracking

Each critical subprogram that fails is null (does not change the host graph).

Each critical subprogram is a rule or rule set followed by a subprogram that cannot fail, such as

```
r1; try r2; r3!
```

The simulation is deterministic:

- Every rule has at most one match.
- In every rule set, at most one rule has a match.

Complexity Results

Theorem: Time Complexity

Every Turing machine M of time complexity $t(n)$ is simulated in time $\mathrm{O}(t^2(n))$, where n is the size of the input.

Theorem: Space Complexity

Every Turing machine M of space complexity $\mathrm{O}(s(n)\log s(n))$ is simulated in graph space $O\left(s(n)\right)$, where n is the size of the input.

Note: graph space is the number of nodes and edges

Uniform and Logarithmic Space Measures

Why is space compression not possible on RAMs, since GP 2 has a C compiler?

- Space compression result uses uniform space measure: unit cost for nodes and edges
- In RAMs: edges are represented by pointers whose size grows with the number of nodes
- Possible alternative logarithmic space measure (Van Emde Boas): a graph of $s(n)$ nodes is assigned a cost of $s(n) \log s(n)$
  Note: this would nullify the space compression!
- A related issue can be found in the time measure of RAM models when programs have to deal with large integers: using logarithmic instead of uniform time may be more realistic

Conclusion

- Turing machines of space complexity $\mathrm{O}(s(n) \log s(n))$ can be simulated in space $\mathrm{O}(s(n))$ with a subset of the DPO-based language GP 2
- A theorem that enables fast matching with graphs and rules different to those in previous theorems
- Alternate approach (not in this paper): efficient simulation of arbitrary SMMs in the GP 2 subset