VP Theory of Computation

Bernhard Kast - 0120422
Horst Stadler - 0120373
Andreas Wagner - 0120552

July 13, 2006

Contents

1 Introduction

This is a follow-up project of the reMOTEable project[1] of the Embedded Software Engineering class in winter term 2005/2006. Hence a short summary of the reMOTEable Project should provide the appropriate background.

1.1 reMOTEable

The reMOTEable project is a distributed systems that supports multiple agents in a network consisting of multiple nodes that are connected via wireless communication. The system supports mobile code, the source code, the whole state and the memory of an agent can be transmitted, hence it provides strong mobility.
Each node consists of one mote[2] that runs a virtual machine that can support numerous agents.

The development of the reMOTEable system drew up the question about how to realize routing in such an "environment". A so called route acquiring agent (RAA) was developed, which is an agent that floods the network at each node it replicates for every unvisited neighour as long as it does not reached the target yet or detects a circle.
It should be noted, that traditional routing schemes are not appropriate for remote sensor networks.[3]

1.2 Aim

The aim of this project is a formalization of the agent system reMOTEable with the main focus to enhance the command-set, especially to reduce the command-set to the smallest possible amount necessary to implement routing algorithms.

Besides the formalization of the network, we focused on building a formal model of the agent and the necessary computation model (a formalized Virtual Machine).

To prove our computation model we have implemented sample routing algorithms in order to show that our system provides the sufficient operations, necessary to conduct such operations.

1.3 Different Approaches

During the project we tried many different approaches, whereas only a few were explored in more depth, these were:

1.3.1 Automaton Approach

At the beginning of the project the idea emerged to reduce the problem of route finding into the formal model of stack automatons. With the following assumptions:

- The network with its nodes was considered as a language.
- The agent is a stack automaton.
- The possible routes are the accepted words of the automaton.

Several Problems showed up with is approach (e.g. modeling the minmal required instruction-set).

[1] http://www.cs.uni-salzburg.at/~ck/wiki/index.php?n=ESE-Winter-2005.ReMOTEable
[2] A mote is a hardware device with very limited processing and storing capabilities, but can connect with others motes via wireless communication.
[3] Braginsky, David; Estrin, Deborah: Rumor Routing Algorithm For Sensor Networks.

1.3.2 Reverse Formalization Approach

Out of the consisting reMOTEable project a formalization and reduction of the instruction-set was done. Starting with a formalization of the network, neighbours, ... a model for the agent has been formalized with the minimal instruction-set required for routing algorithms. As a definition of the semantics of each command a formal virtual machine was build.
This paper will focus on this approach.

2 Syntactical Definitions

2.1 Network

A network Nw (N, C) is a 2-tuple, where:

- N is the set of all nodes,
- C are the connections: $C := \{\{a, b\} | a, b \in \mathbf{N} \wedge a \text{ is connected with } b\}$.

2.2 Agent

An agent is an 4-tuple (PGM, PC, MEM, $CNODE$), where:

- PGM is the program of the agent,
- PC is the programcounter: $PC \in \mathbb{N}$,
- MEM is the memory of the agent,
- $CNODE \in \mathbf{N}$: the node on which the agent is currently residing. An agent can only be active in exactly one node at a certain time.

A is the set of all agents.

2.2.1 PGM - Program

A program is a Graph $G = (CMD\text{s}, TRANSITIONS)$, where the vertices ($CMD$s) are the set of commands and the edges represent the transitions to the next command. The CMDs are instances of the $INSTRUCTION$ type set and the $TRANSITIONS$ are the set of all possible transitions in the graph where every vertex has exactly one outgoing edge except:

- the last vertex, which has only incoming edge,
- for all vertices of the type BEQ, which always have two outgoing edges.

2.2.2 PC - Programcounter

PC is the programcounter: $PC \in \mathbb{N}$.
$0 \leq PC \leq n$, n is the number of program commands.

2.2.3 CMD - Command

A CMD is a 2-tuple ($INSTRUCTION$, $ARGUMENTS$), where:

- $INSTRUCTION$ is the set of all elementary instructions,
- $ARGUMENTS$ is the tuple of arguments.

2.2.4 *INSTRUCTION*

The set *INSTRUCTION* defines the minimal required instruction set of the agent. We provided two different sets, once which enables the use of replication (for flooding agents) and once which provides an intelligent instruction that enables traversing the whole net without flooding it. Hence it is possible to choose the more appropriate instruction-set and implement different algorithms with it.

Possible Sets:

- $INSTRUCTION_1$:= { PUSH, POP, MOVE, CMPJMP, LOADNEXT } without replication.
- $INSTRUCTION_2$:= { PUSH, POP, MOVE, CMPJMP, MOVEREPLICATE } for flooding.

2.2.5 *MEM* - Memory

MEM consists of three parts:

- pathMem,
- tempMem,
- targetMem.

Where all parts are strings and each character is representing a node. [4]

2.3 ★ - Neighbour

$\bigstar(x) := \{y \in \mathbf{N} | x \in \mathbf{N} \wedge \{x, y\} \in \mathbf{C}\}$
Predicate $visited(a, n)$ is true if the agent a, has visited n.

3 Semantics

3.1 Access function for memory

$G(x_1 x_2 \dots x_{n-1} x_n) := x_n$ (G stands for grab)

3.2 PUSH

$$PUSH(A, B) = \{ \quad B : (b_1 b_2 \dots b_n) \longmapsto B : (b_1 b_2 \dots b_n, G(A)), PC \longmapsto PC + 1$$

3.3 POP

$$POP(B) = \{ \quad B : (b_1 b_2 \dots b_n) \longmapsto B : (b_1 b_2 \dots b_{n-1}), PC \longmapsto PC + 1$$

3.4 BEQ

$$BEQ(A, B, c) = \begin{cases} G_n(A) = G_n(B), & PC \longmapsto c \\ otherwise, & PC \longmapsto PC + 1 \end{cases}$$

3.5 LOADNEXT

$$LOADNEXT(B) = \begin{cases} \exists x \in \bigstar(CNODE) \wedge \neg visited(x), & B : (b_1 b_2 \dots b_n) \longmapsto B : (b_1 b_2 \dots b_n x), PC \longmapsto PC + 1 \\ \text{otherwise}, & B : (b_1 b_2 \dots b_n) \longmapsto B : (b_1 b_2 \dots b_{n-1}), PC \longmapsto PC + 1 \end{cases}$$

[4] For the targetMem it is sufficient to have only one character

3.6 MOVE

$$MOVE(B) = \begin{cases} B = \emptyset, & \mathbf{A} \longmapsto \mathbf{A} \setminus \{a_{this}\}, PC \longmapsto PC + 1 \\ \text{otherwise}, & CNODE \longmapsto G_n(B), PC \longmapsto PC + 1 \end{cases}$$

3.7 MOVEREPLICATE

$$MOVEREPLICATE(B) = \begin{cases} \forall x \in \bigstar(CNODE) \wedge \neg visited(x) : \text{generate } a_{new} \in \mathbf{A} \\ \text{with } (a_{old}.PGM, a_{old}.PC, a_{old}.MEM, a_{old}.x) \\ \mathbf{A} \setminus a_{old}, PC \longmapsto PC + 1 \end{cases}$$

4 Algorithms

4.1 Algorithm without Replication

```
010 PUSH      target, targetMem
020 PUSH      currentNode, pathMem
030 BEQ       currentNode, targetMem, 80
040 LOADNEXT  pathMem
050 MOVE      pathMem
060 BEQ       0,0,30
070 PUSH      currentNode, tempMem
080 MOVE      pathMem
090 POP       pathMem
100 BEQ       0,0,10
```

4.2 Algorithm with Replication

```
010 PUSH           target, targetMem
020 PUSH           currentNode, pathMem
030 BEQ            currentNode, targetMem, 60
040 MOVEREPLICATE  pathMem
050 BEQ            0,0,20
060 PUSH           currentNode, tempMem
070 MOVE           pathMem
080 POP            pathMem
090 BEQ            0,0,10
```

5 Conclusion

We formalized a computational model and as a proof we implemented two routing algorithms.

- Algorithm 1 works without replication of the agent. This algorithm finds every route, including the shortest one.
- Algorithm 2 uses replication to flood the network. This algorithm finds at least one route, if there exists one.

For both algorithms five instructions are sufficient. The original reMOTEable agents have 15 low-level instructions. These two instruction sets can not be compared, because the reROUTEable agents are only used for routing. Each of the two algorithms has one complex instruction, requiring more complex interaction with the node.

6 *LOADNEXT*

In order to further improve the model, we are working on the idea of handling the the complex *LOADNEXT* command as an subroutine. In detail we would like to express the *LOADNEXT* command with the other commands. In this paragraph we do not provide a solution in form of code, but we evaluate the needed extentions to realize this idea. Two different solutions are discussed.

- An "bruteforce" compare algorithm which requires additional memory capability: Assume a reduction of the *LOADNEXT* command to a simple command only returning the list of all neigbours of a node. An agent would have to store all vistited nodes in memory and compare this list with the neighbours, if every neighbour of the actual node is already resident in this memory no further unvisited neighbour exists and the Agent would have to step back to the previous node. This algorithm could be implemented with the *BEQ* Instruction.
- Neighbour lists with sorting orders: To avoid additional memory and the countless compare operations an different approach could be implemented. Every node returns at every time it's Neighbour list in the same order. The agent only stores the node it currently returns from on the tempMem (instead of throwing it away like the branch of the current *LOADNEXT* command). When the agent returns to a node, it gets the same list of neighbours and because of the order, it can distinguish from the visited and unvisited neighbours. If there are no more unvisited nodes it steps back one node in its path.

7 Further Work

Interesting questions:

- Is there a simpler model? (Tradeoff memory complexity vs. additional instructions.)
- Evaluate extendability of routing model to all functionalities of reMOTEable.
- What are the limitations of this model?

8 Bibliography

Hofstätter, Alois; Kast, Bernhard; Stadler, Horst: reMOTEable
http://www.cs.uni-salzburg.at/~ck/wiki/index.php?n=ESE-Winter-2005.ReMOTEable

Sipser, Michael: Introduction to the Theory of Computation.

Braginsky, David; Estrin, Deborah: Rumor Routing Algorithm For Sensor Networks.