Project 2: Remodel

Bruce Spang

December 8, 2013

1 Abstract

Remodel is a re-implentation of make [1] as per [2]. It executes dependencies in parallel, when possible.

See README.md for details on building, testing, and running remodel.

2 Building the Graph

We parse the input file one line at a time

Nodes in the dependency graph correspond to files (e.g. foo.c) or logical groups (e.g. DEFAULT, compile), edges correspond to dependencies, edge labels correspond to a command to execute when traversing that edge.

Each line contains a list of parents, children, and an optional edge label. To build the graph, we add the parent and nodes and then add a labeled edge between each parent and child pair.

To avoid running a command with m parents and n children mn times, we add two nodes: a parent command node and a child command node. When a line has a command specified and has more than one parent or child, we add a node with the parents as dependencies, a node with the children as targets, and an edge between the two new nodes labeled with the command. This ensures that the command is executed once, and that all parents are executed before the command.

We do other graph validation, like enforcing that there can only at most one edge between two nodes, and that there are no cycles.

3 Executing the Graph

To actually execute the commands, we do a parallel topological sort based on [3]. We enqueue each edge into a thread-safe queue, and repeatedly dequeue an edge, execute it, mark it as visited, and enqueue all the edges of its child if the child has no more parents to visit. We use concurrency kit here for its fetch-and-set and fetch-and-add primitives.

When executing an edge, we check if the parents's md5 sum and child's md5 sum matches a cached md5 sum. If it does not or if the parent is marked for execution, we mark the edge and its child for execution. If an edge is marked for execution and is labeled with a command, we execute the command.

4 Notes

We load the dependency graph specified in the arguments. We do not take a target to execute for a few reasons:

1. We cannot make useful guarantees about what will be executed. We still need to execute children of the target, and we may need to execute parents of the target (if they've been modified), and cousins of the target (if their parents were modified). We could guarantee that only the component of the graph containing the production would be executed, but that seems like more trouble than its worth.

2. Since we allow more than one production per rule, it's unclear what the correct behavior when executing a target is: only execute edges that string-match the target, only execute edges that have the same children as the target, execute all edges whose intersection with the target is non-empty, all edges who are a subset of the target, etc...

3. When building small software projects, developers should not be allowed to run part of a a build. Running part of a build usually means that tests are not run (which is bad), or that the tests are insufficient (which is also bad). We do not want to encourage this behavior.

5 Code Structure

- `src/main.c` contains the CLI, and checks whether the graph has cycles.
- `src/file.c` checks whether a file's md5 sum has changed from the cached version
- `src/remodel.c` builds the dependency graph (beginning in `remodel_load_file()`) and executes it executes it (beginning in `remodel_execute()`)
- `src/parse.y, src/scan.l` contain the lexer and parser specification
- `include/` contains headers.
- `test/` contains tests.

References

[1] Gnu make. http://www.gnu.org/software/make/.

[2] Project 2: Remodel: Make done right.
http://plasma.cs.umass.edu/emery/project-2.

[3] Jochem Schutte. Implementing parallel topological sort in a java graph library. http://fmt.cs.utwente.nl/files/sprojects/64.pdf.