Exercise 10.1  Colorability

Show that Chaitin’s algorithm finds a $k$-coloring for every $k$-colorable chordal graph.

*Hint:* Every chordal graph with at least 2 nodes has 2 simplicial nodes.

Exercise 10.2  Partitioned Boolean Quadratic Programming

Prove that finding a solution for a PBQP problem to be NP-hard by reducing SAT to PBQP.

*Hint:* Reconsider the NP-hardness proof for instruction selection. First, try to map the boolean formula $(a \land b) \lor \neg b$ from the example in Figure 2 of Koes’ paper\(^1\) to PBQP. Then, you can derive an algorithm to map any SAT problem to PBQP. Generally, to map $a \lor b$ you will need four nodes: one for $a$, one for $b$, one for $\lor$ and an auxiliary node.

Exercise 10.3  PBQP Applied

1. Study the LLVM-IR program below and draw the value graph for the loop body (`for.body`). Include constants, function arguments and PHI nodes from other blocks in the graph. Futhermore, replace the `getelementptr` instruction by appropriate scalar operations (`add/mul`) and fold constant expressions together. Assume the size of an `i32` is 4 bytes.

2. Use the patterns on the PBQP slide 19 and the costs shown below to create a PBQP instance for the graph constructed in part 1. Assume the patterns `AC` and `A` are also available for multiplications (`MC/M`) and that there is a pattern `AR` for accessing an arguments.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>AR</th>
<th>C</th>
<th>P</th>
<th>A</th>
<th>AC</th>
<th>M</th>
<th>MC</th>
<th>L</th>
<th>LA</th>
<th>LAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

3. Use the optimality-preserving reductions and the heuristic reduction to find a solution for the PBQP problem. Write down the order in which edges/nodes are eliminated and the rule that was applied.

```c
1 define i32 @array_sum(i32* %A, i32* B, i32 %N) {
2 entry:
3 br label %for.cond
4
5 for.cond:
6 %iv = phi i32 [ 0, %entry ], [ %iv.inc, %for.body ]
7 %sum = phi i32 [ 0, %entry ], [ %add1, %for.body ]
8 %B.cur = phi i32* [ %B, %entry ], [ %B.idx, %for.body ]
9 %cmp = icmp slt i32 %iv, %N
10 br i1 %cmp, label %for.body, label %for.end
11
12 for.body:
13 %A.idx = getelementptr i32, i32* A, i32 %iv
14 %B.idx = getelementptr i32, i32* B, cur, i32 1
15 %A.val = load i32, i32* A.idx, align 4
16 %B.val = load i32, i32* B.idx, align 4
17 %add1 = add i32 %sum, %A.val
18 %add2 = add i32 %add1, %B.val
19 %iv.inc = add i32 1, %iv
20 br label %for.cond
21
22 for.end:
23 ret i32 %sum
24 }
```

\(^1\)available at https://dl.acm.org/citation.cfm?id=1356065 (from university network)
Project task G Implement Sparse Conditional Constant Propagation

For this project assignment, we introduce an additional compiler switch, --optimize. This switch performs the same operation as --compile and additionally performs optimizations. Implement the following optimization:

• Perform the SCCP\(^2\) analysis.

• If applicable,
  – substitute instructions with constants,
  – substitute conditional branches with unconditional ones,
  – remove dead instructions and
  – remove unreachable blocks.

• If you do not want to implement SSA construction yourself, you may use the LLVM mem2reg pass (see \texttt{llvm::createPromoteMemoryToRegisterPass()}) \textbf{but no other LLVM passes}.

• This is the last project assignment. Information on scheduling your code review meeting with us will be available in the forum.

• The soft deadline for this milestone is 2018-02-02.

• Keep it simple!

\(^2\)consider \url{https://dl.acm.org/citation.cfm?id=103136} for more details if you haven’t already (accessible from the university network).