An NMF solution to the State Elimination Case at the TTC 2017

Georg Hinkel

Sparse adoption of MDE in industry

- Tool support perceived insufficient [Sta06,Mo+13]
  - Much less manpower in tool development than IDEs such as Visual Studio, IntelliJ, …
- Developers hardly change their primary language [MR13]
  - Project requirements or code reuse

.NET Modeling Framework (NMF)

- Model repository management in .NET
  - Generate code for metamodels
  - Load models
  - Save models
  - (Mostly) Compatible to EMF
- Further tools for Model Transformation, Synchronization, Incrementalization, …
  - Implemented as Internal DSLs
- Open source: http://github.com/NMFCode/NMF

Tool Appropriateness

- NTL: unidirectional batch model transformations
  - Assumption: correspondence relation between source and target elements important

  ➔ not applicable
- NMF Synchronizations: multimode model synchronization
  - Incremental and/or bidirectional model transformations

  ➔ not applicable
- NMF Expressions: Incrementalization system and inverter of model analyses
  - Analyses must be referentially transparent except for object creation

  ➔ not applicable

➔ Solution is based on standard C# but uses generated model API

Loading the model

```
var repository = new ModelRepository();
var transitionGraph = repository.Resolve(path).RootElements[0] as TransitionGraph;
```

Creating a unique initial state with no incoming transitions

```
var initial = transitionGraph.States.FirstOrDefault(s => s.IsInitial);
if (initial.Incoming.Count > 0)
{
  var newInitial = new State { IsInitial = true };
  transitionGraph.Transitions.Add(new Transition
  {
    Source = newInitial,
    Target = initial
  });
  initial = newInitial;
}
```

Creating a unique final state with no outgoing transitions

```
var finalStates = transitionGraph.States.Where(s => s.IsFinal).ToList();
if (finalStates.Count == 1 && finalStates[0].Outgoing.Count == 0)
{
  return finalStates[0];
}
else
{
  var newFinal = new State();
  foreach (var s in finalStates)
  {
    transitionGraph.Transitions.Add(new Transition
    {
      Source = s,
      Target = newFinal
    });
  }
  transitionGraph.States.Add(newFinal);
  return newFinal;
}
```

Considerations on eliminating states

- For each state, the elimination has to create or update $i \cdot o$ transitions where i is the number of incoming transitions and o the number of outgoing transitions
- For $i = n, o = n$ (as suggested in the description), this yields n^2 transitions for each state ➔ $O(n^3)$ runtime
- Avoid creating transitions to reduce complexity (most states have few transitions)
- If we create transitions lazily, for each state, $i \cdot o$ new transitions are generated
  - These new transitions grow the number of transitions to generate in later iterations of the loop!
- Try to reduce creating new transitions by sorting states by $i \cdot o$

State Elimination

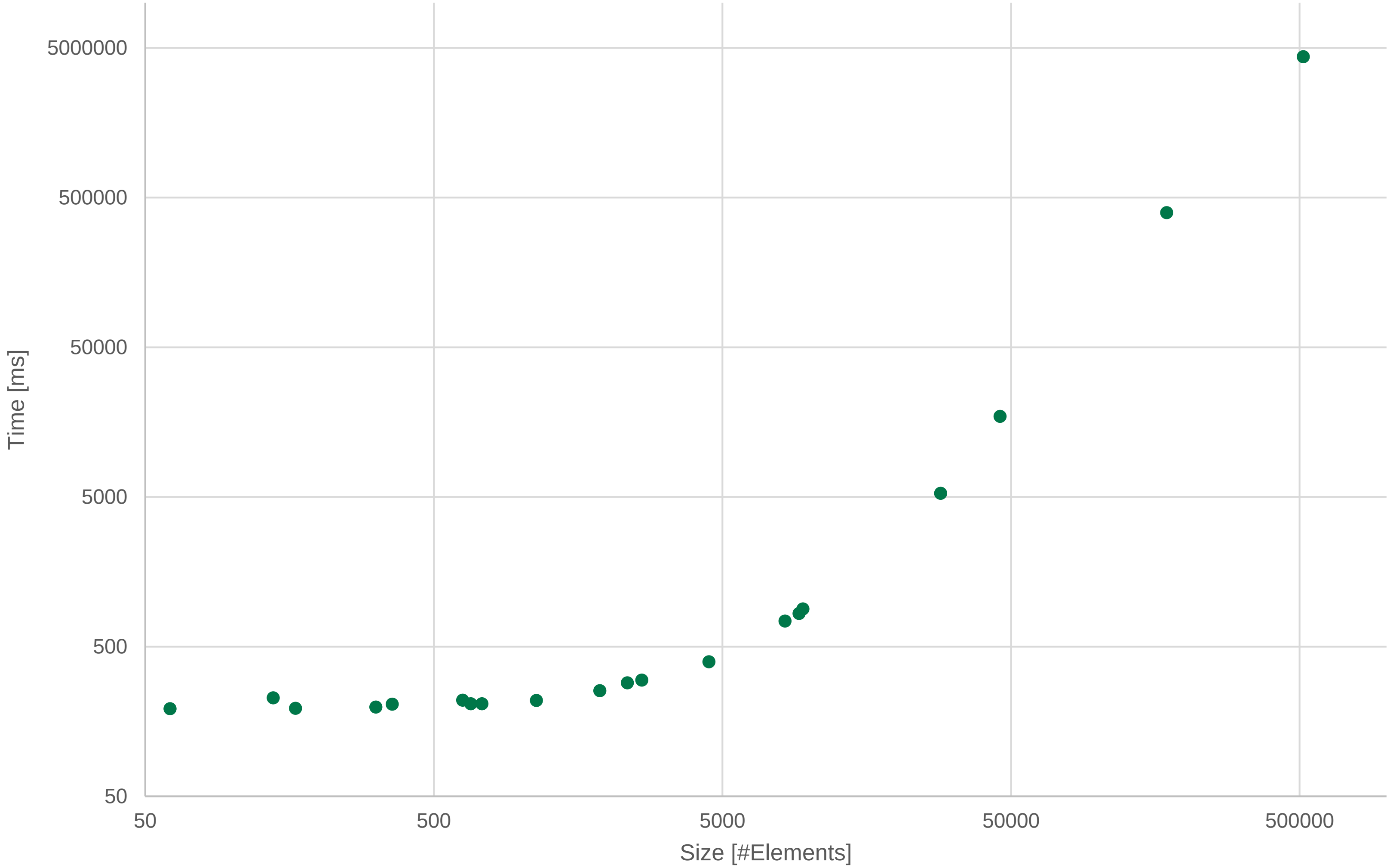
```
foreach (var s in transitionGraph.States.OrderBy(s => s.Incoming.Count * s.Outgoing.Count).ToArray())
{
  if (s == initial || s == final) continue;

  var selfEdge = string.Join("+", from edge in s.Outgoing
                                  where edge.Target == s
                                  select edge.Label);

  if (!string.IsNullOrEmpty(selfEdge)) selfEdge = string.Concat("(", selfEdge, ")*");

  foreach (var incoming in s.Incoming.Where(t => t.Source != s))
  {
    if (incoming.Source == null) continue;
    foreach (var outgoing in s.Outgoing.Where(t => t.Target != s))
    {
      if (outgoing.Target == null) continue;
      var transition = incoming.Source.Outgoing.FirstOrDefault(t => t.Target == outgoing.Target);
      if (transition == null)
      {
        transitionGraph.Transitions.Add(new Transition
        {
          Source = incoming.Source,
          Target = outgoing.Target,
          Label = incoming.Label + selfEdge + outgoing.Label
        });
      }
      else
      {
        transition.Label = string.Concat("(", transition.Label, "+", incoming.Label,
                                         selfEdge, outgoing.Label, ")");
      }
    }
  }

  s.Delete();
}
```

Evaluation

Model	# Elements	Regex size	Time to transform (ms)	Correct	JFLAP (ms)
leader3_2	61	33	192	✓	90
leader3_3	166	95	193	✓	490
leader3_4	359	210	206	✓	4,370
leader3_5	672	398	207	✓	58,600
leader3_6	1,135	675	218	✓	461,640
leader3_8	2,631	1,571	298	✓	–
leader4_2	139	76	227	✓	140
leader4_3	630	354	219	✓	57,780
leader4_4	1,881	1,067	253	✓	4,786,580
leader4_5	4,492	2,558	395	✓	–
leader4_6	9,221	5,262	831	✓	–
leader5_2	315	172	197	✓	3,460
leader5_3	2,344	1292	286	✓	–
leader5_4	9,513	5,267	890	✓	–
leader5_5	28,544	15,833	5,277	✓	–
leader6_2	735	398	207	✓	143,120
leader6_3	8,248	4,487	739	✓	–
leader6_4	45,865	24,979	17,210	✓	–
leader6_5	173,194	94,408	395,616	✓	–
leader6_6	515,077	280,865	4,356,603	✓	–
leader6_8	2,886,813	–	–	–	–

- Conciseness: 102 lines of code (31 empty or only braces)

Evaluation II



Conclusion

- Insights
  - Model transformation technologies are the wrong tool here
  - Key improvements algorithmic

- Key advantages of the solution
  - Concise (about as concise as external languages)
  - Solution easily integrates into C# ➔ good tool support
  - Very good performance
  - Very good scalability

hinkel@fzi.de

THANK YOU FOR YOUR ATTENTION