Change Propagation in an Internal Model Transformation Language

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

Promising approach: Internal DSLs

- Inherit tool support from host language
- Let developers stay with their primary language

- Downside: No change propagation, few bidirectional approaches
  - Change Propagation → Performance, Efficiency
  - Bidirectionality → Applicability, Reuse

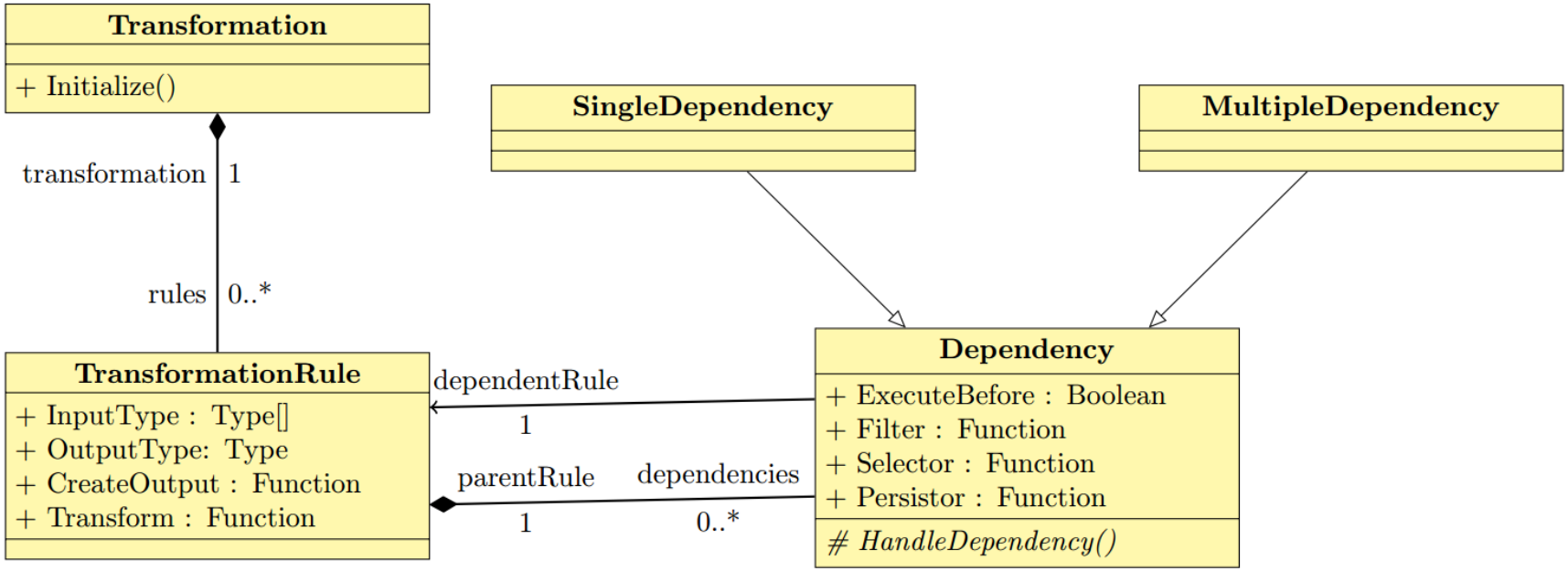
Related Work

- Self-adjusting Computation
  - Explicit [Acar05, Bur+11]
  - Implicit [Che+14]
  - ➔ No abstractions for model transformation
- Internal Model Transformation Languages
  - RubyTL [CMT06]
  - FunnyQT [Hor13]
  - NTL [Hin13]
  - ➔ No Change Propagation, few bidirectionality
- Model Transformation with Change Propagation
  - TGG [GW06]
  - …
  - ➔ No internal Languages, other transformation paradigms

How to obtain Change Propagation?

- Combine Self-adjusting Computation with Model Transformation
  - Model Transformation languages provide domain-specific abstractions
  - Self-Adjusting Computation provides techniques for change propagation
- Implementations for both as internal DSLs in .NET Modeling Framework (NMF)
  - Model Transformation: NMF Transformations
  - Self-adjusting Computation: NMF Expressions
  - Reversability extensions for NMF Expressions

NMF Transformations

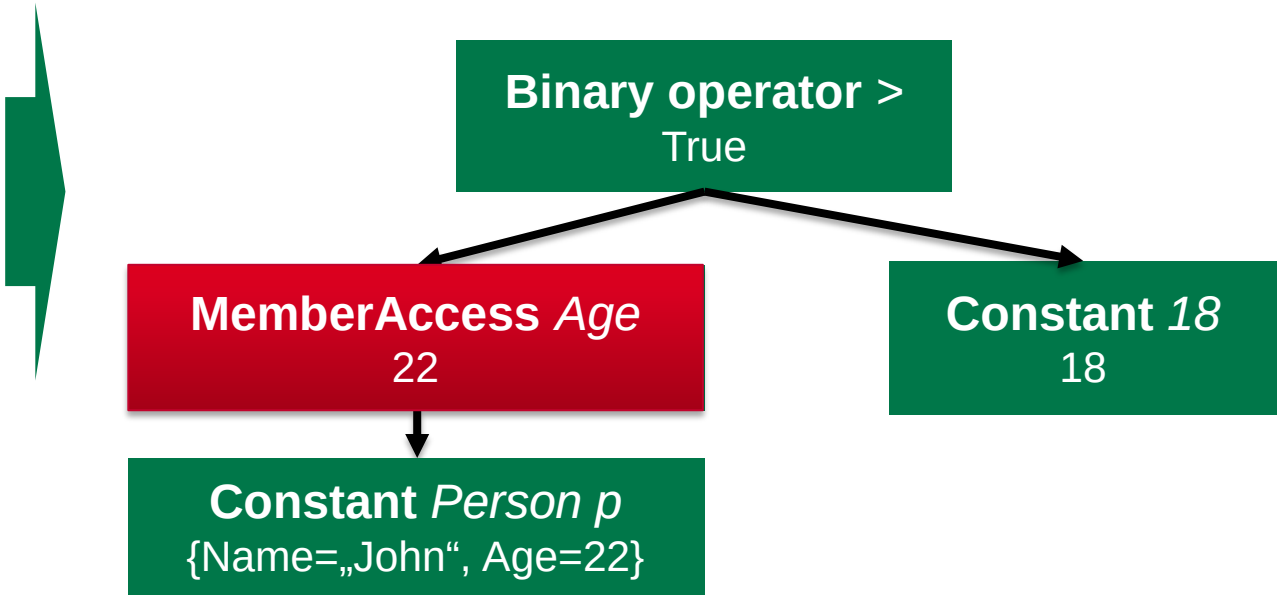


[Hin13]

NMF Expressions

- Self-adjusting computation at instruction level
- Based on C# Expression Trees

- $p.Age > 18$



Putting it all together

- NMF Synchronizations
  - Transformation Engine of NMF Transformations
  - Elementary Change Propagation through NMF Expressions
  - Reversability through Lense-extensions of NMF Expressions

- Internal Model Transformation & Synchronization DSL in C#
- 18 Modes out of a single specification

Transformation Directions

- **LeftToRight**
  - Transformation enforces that model elements exist in RHS if a corresponding element is in LHS
- **LeftToRightForced**
  - LeftToRight + Elements on RHS with no corresponding element in LHS are deleted
- **LeftWins**
  - LeftToRight + Elements on RHS with no corresponding element in LHS are transformed to LHS
- **RightToLeft, RightToLeftForced, RightWins**

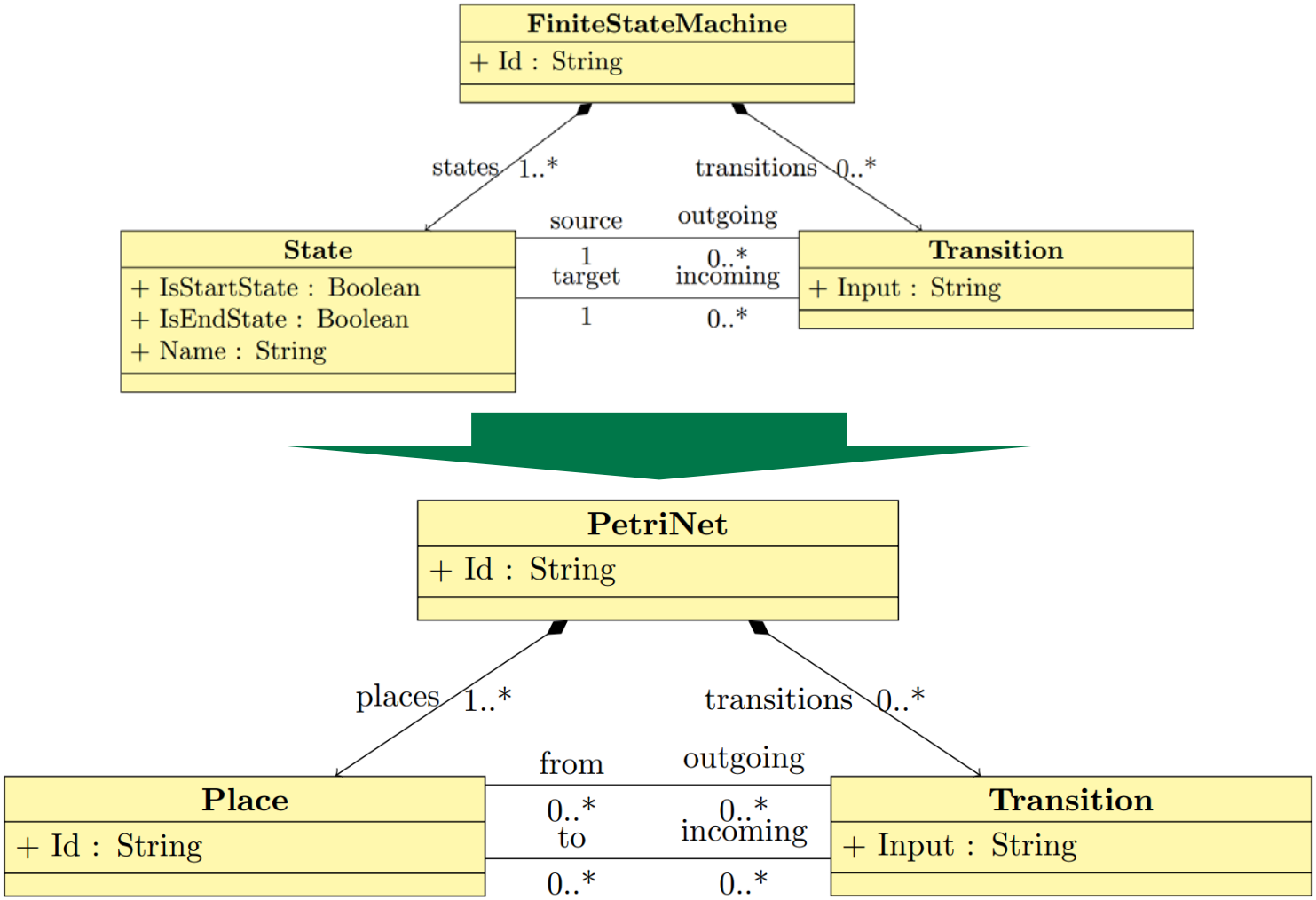
Change Propagation Modes

- **None**
  - Change Propagation is disabled
- **OneWay**
  - Change Propagation from Source to Target
- **TwoWay**
  - Change Propagation in both direction

Decisions for Expressions in NMF Expressions

1. Compile expression and run it normally
2. Reverse expression, compile and run normally
3. Create Dynamic Dependency Graph
   - Receive Incremental Updates
4. Create Reversable Dynamic Dependency Graph
   - Receive Incremental Updates
   - Push Updates backwards through the graph

Finite State Machines and Petri Nets



Finite State Machines to Petri Nets I

```
public class PSM2PN : ReflectiveSynchronization
{
  public class AutomataToNet : SynchronizationRule
          <FiniteStateMachine, PetriNet>
  {...}


  …
}
```

Finite State Machines to Petri Nets II

```
public override void DeclareSynchronization()
{
  SynchronizeMany(SyncRule<StateToPlace>(),
          fsm => fsm.States, pn => pn.Places);
  SynchronizeMany(SyncRule<TransitionToTransition>(),
          fsm => fsm.Transitions,
          pn => pn.Transitions.Where(t => t.To.Count > 0));
  …
  Synchronize(fsm => fsm.Id, pn => pn.Id);
}
```

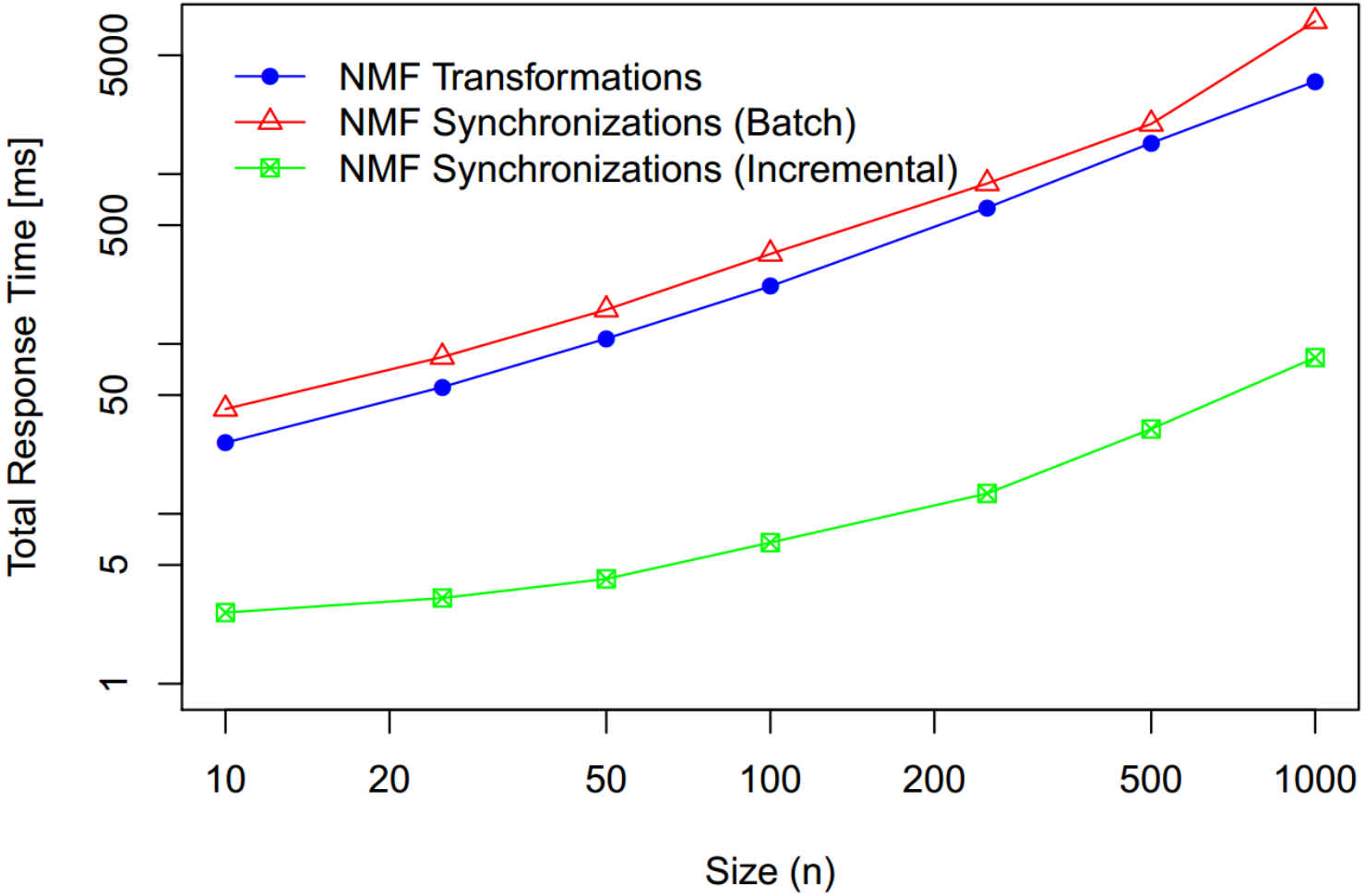
Validation goals

- Validate Correctness
  - Generate Change sequences
  - Propagate changes or retransform input
  - Comparison against NMF Transformations solution
- Validate Performance
  - Response time to apply full change sequence & propagate changes
  - 100 change sequences á 100 elementary changes

Composition of the change sequence

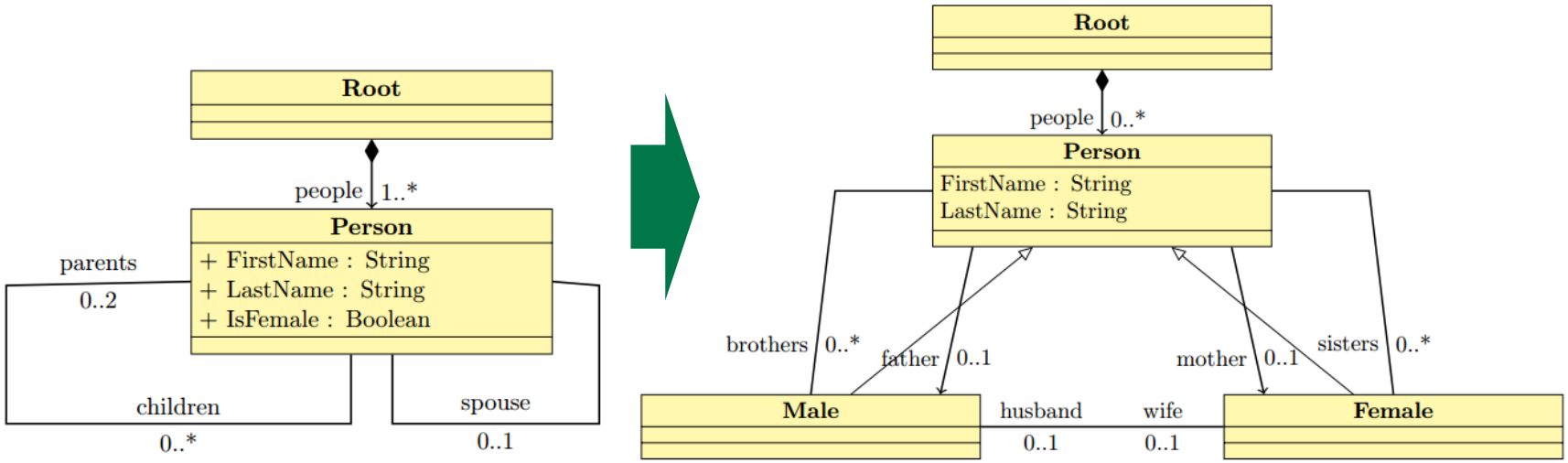
- Add a state to the finite state machine (30%)
- Add a transition to the finite state machine with random start and end state (30%)
- Remove a random state and all of its incoming and outgoing transitions (10%)
- Remove a random transition from the finite state machine (10%)
- Toggle end state of a random state (5%)
- Change the target state of a randomly selected transition to a random other state (5%)
- Rename a state (9%)
- Rename the finite state machine (1%)

Results


NMF Transformations
NMF Synchronizations (Batch)
NMF Synchronizations (Incremental)
Total Response Time [ms]
1
5
50
500
5000
10
20
50
100
200
500
1000
Size (n)

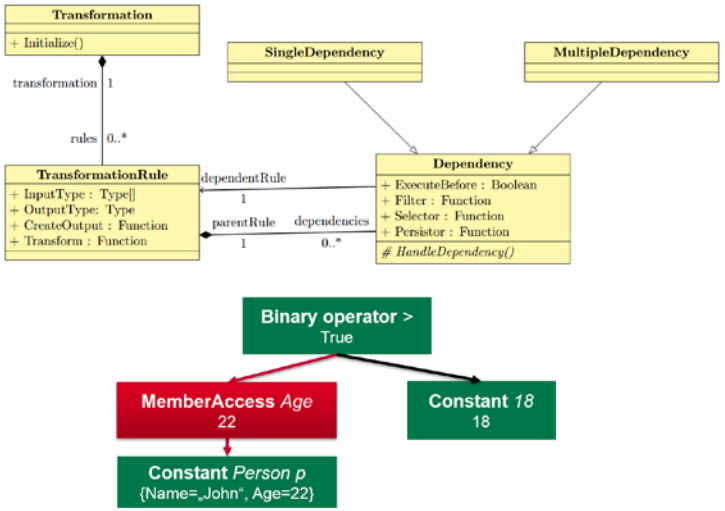
Limitations

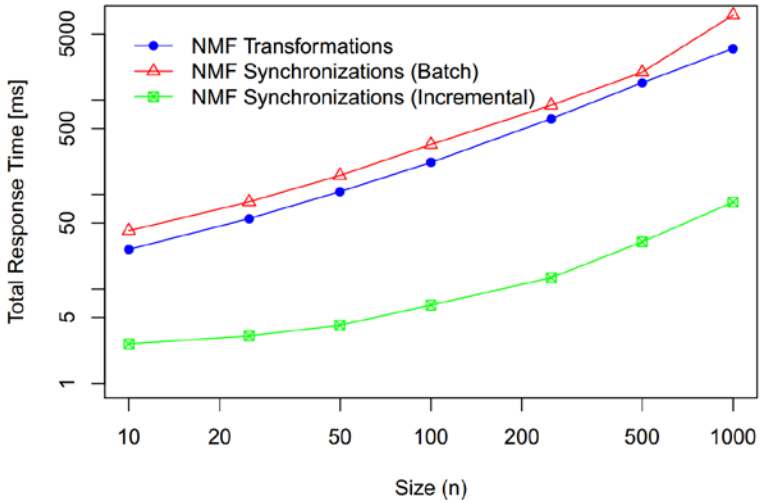
- Correspondences fixed once created
  - Not a general restriction



Conclusion

- Proof-of-concept: Change Propagation and Bidirectional Transformations possible with Internal Model Transformation Languages
- 18 operation modes from single specification
- Speedups of up to 48 measured





References I

Sta06	M. Staron, “Adopting model driven software development in industry–a case study at two companies,” in *Model Driven Engineering Languages and Systems*, Springer, 2006, pp. 57–72
Mo+13	P. Mohagheghi, W. Gilani, A. Stefanescu, and M. A. Fernandez, “An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases,” *Empirical Software Engineering*, vol. 18, no. 1, pp. 89–116, 2013.
MR13	L. A. Meyerovich and A. S. Rabkin, “Empirical analysis of programming language adoption,” in *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications*, ACM, 2013, pp. 1–18.
Hin13	G. Hinkel, “An approach to maintainable model transformations using an internal DSL”, Master's thesis, Karlsruhe Institute of Technology, October 2013.
Fo+13	Foster, J. Nathan, et al. "Combinators for bi-directional tree transformations: a linguistic approach to the view update problem." *ACM SIGPLAN Notices* 40.1 (2005): 233-246.

References II

GW06	H. Giese and R. Wagner, “Incremental model synchronization with triple graph grammars,” in Model Driven Engineering Languages and Systems, Springer, 2006, pp. 543–557.
Acar05	U. A. Acar, “Self-adjusting computation,” PhD thesis, 2005.
Che+14	Y. Chen, J. Dunfield, M. A. Hammer, and U. A. Acar, “Implicit self-adjusting computation for purely functional programs,” Journal of Functional Programming, vol. 24, no. 01, pp. 56–112, 2014.
CMT06	J. S. Cuadrado, J. G. Molina, and M. M. Tortosa, “Rubytl: a practical, extensible transformation language,” in Model Driven Architecture–Foundations and Applications, Springer, 2006, pp. 158–172.
Hor13	T. Horn, “Model querying with FunnyQT,” in Theory and Practice of Model Transformations, Springer, 2013, pp. 56–57.
GWS13	L. George, A. Wider, and M. Scheidgen, “Type-Safe model transformation languages as internal DSLs in Scala,” in Theory and Practice of Model Transformations, Springer, 2012, pp. 160–175.
Bur+11	S. Burckhardt, D. Leijen, C. Sadowski, J. Yi, and T. Ball, “Two for the price of one: a model for parallel and incremental computation,” in ACM SIGPLAN Notices, ACM, vol. 46, 2011, pp. 427–444.

Reversable Expressions through Lenses

- Algebraic construct consisting of two functions
  - Get $l \searrow: A \rightarrow B$
  - Put $l \nearrow: A \times B \longrightarrow A$

$$\left.\begin{array}{l} l \searrow: A \rightarrow B \\ l \nearrow: A \times B \longrightarrow A \end{array}\right\} \quad A, B \in ob\, \mathcal{C}$$

[Fo+05]

- Specification of Lenses through attributes
  - Allow developers to specify how to revert function calls

```
[ObservableProxy(typeof(ObservableFirstOrDefault<>), "CreateExpression")]
[SetExpressionRewriter(typeof(ObservableFirstOrDefault<>), "CreateSetExpression")]
public static TSource FirstOrDefault<TSource>(this IEnumerableExpression<TSource> source)
```

Finite State Machines to Petri Nets III

```
public override bool ShouldCorrespond
        (FSM.Transition left, PN.Transition right,
        ISynchronizationContext context)
{
  var stateToPlace = SyncRule<StateToPlace>().LeftToRight;
  return left.Input == right.Input
  && right.From.Contains
        (context.Trace.ResolveIn(stateToPlace, left.StartState))
  && right.To.Contains
        (context.Trace.ResolveIn(stateToPlace, left.EndState));
}
```

Finite State Machines to Petri Nets IV

```
public override void DeclareSynchronization()
{
  SynchronizeLeftToRightOnly(SyncRule<StateToPlace>(),
            state => state.IsEndState ? state : null,
            transition => transition.From.FirstOrDefault());
}
```