De-/Re-constructing Model Transformation Languages

Eugene Syriani

Ph.D. Candidate in the Modelling, Simulation and Design Lab
School of Computer Science

McGill University
OVERVIEW

- Context
- De-Constructing Transformation Languages
  - Collection of MT primitives
- Re-Constructing Transformation Languages
  - FUJABA
  - More esoteric features
- MoTif-Core: a re-construction example
  - MoTif
  - GReAT
- Conclusion
THE BIG PICTURE
IN THE CONTEXT

• Many different model transformation languages (MTLs)
  – Features [1]: atomicity, sequencing, branching, looping, non-determinism, recursion, parallelism, back-tracking, hierarchy, time
  – Transformation rule: matching + rewriting + validation

• Hard to
  – Compare expressiveness
  – Provide framework for interoperability

• Express MTLs in terms of primitive building blocks
  – De-Construction: small set of most primitive constructs
  – Re-Construction: discover new MTLs + interoperation + optimization

**T-Core Module**

- 8 primitives
- Composition operator
- 3 types of messages
- Exchange of messages through methods
- 3 output states:
  - Success
  - Fail
  - Exception
Matcher

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isSuccess</td>
<td>Boolean</td>
</tr>
<tr>
<td>findAll</td>
<td>Boolean</td>
</tr>
<tr>
<td>condition</td>
<td>PreConditionPattern</td>
</tr>
<tr>
<td>packetIn</td>
<td>Packet</td>
</tr>
<tr>
<td>cancelIn</td>
<td>Void</td>
</tr>
</tbody>
</table>

**Algorithm 1**: Matcher.packetIn(\(\pi\))

1. Find all matches (parameter)
2. Store result in packet
### Rewriter

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isSuccess</td>
<td>Boolean</td>
</tr>
<tr>
<td>exception</td>
<td>Exception</td>
</tr>
<tr>
<td>condition</td>
<td>PostConditionPattern</td>
</tr>
<tr>
<td>packetIn</td>
<td>(Packet): Packet</td>
</tr>
<tr>
<td>cancelIn</td>
<td>(Cancel): void</td>
</tr>
</tbody>
</table>

#### Algorithm 2: Rewriter.packetIn(\(\pi\))

```java
if \(\pi\) is invalid then
  isSuccess \(\leftarrow\) false
  exception \(\leftarrow\) \(\chi(\pi)\)
  return \(\pi\)
end if

apply transformation on \(M\).matchesToRewrite for which \((\text{condition.pre}, M) \in \pi\).matchSets

if transformation failed then
  isSuccess \(\leftarrow\) false
  exception \(\leftarrow\) \(\chi(\pi)\)
  return \(\pi\)
end if

set all modified nodes in \(M\) to dirty
remove \((\text{condition}, M)\) from \(\pi\).matchSets

isSuccess \(\leftarrow\) true
return \(\pi\)
```

1. Check validity of packet
2. Apply transformation
3. Propagate changes in all match sets
4. Consume match

*Exception possible!*
1. Check if match set is not empty
2. Randomly choose a match
1. Push packet onto stack

1. Match set not empty: there are matches left (pass on)
2. No match set: back-track to previous state
1. Conservative check for potential conflict between different matches in match sets (parameter)

2. Customizable resolution function

Exception possible!
1. successIn: add to success set
2. failIn: add to fail set
3. Choose randomly first from success then from fail

Exception possible!
1. **successIn**: add to success set

2. **failIn**: add to fail set

3. Merge only if all threads succeeded

4. Customizable merge function

*Exception possible!*

```
Algorithm 9 Synchronizer.merge()
if |success| = threads then
  if defaultMerge() then
    π ← the merged packet in success
    isSuccess ← true
    success ← \emptyset
    fail ← \emptyset
    return π
  else if customMerge() then
    π ← the merged packet in success
    isSuccess ← true
    success ← \emptyset
    fail ← \emptyset
    return π
  else
    isSuccess ← false
    exception ← χ(π_
    return π_
  end if
else if |success| + |fail| = threads then
  π ← choose from fail
  isSuccess ← false
  return π
else
  isSuccess ← false
  exception ← χ(π_
  return π_
end if
```
1. Meaningfully composes its sub-primitives

2. User-defined composition
DE-CONSTRUCTING TRANSFORMATION LANGUAGES
Motivating T-Core

• De-construct up to what level?
• What to include, what to exclude?
  – Pre/PostConditionPattern: rules, bi-directional, functions
  – Separation match/rewrite: queries, nested transformaitions
  – Packet: sufficient info to be processed by each primitive, designed for concurrent transformations
  – Composition: scaling for large model transformations
  – T-Core module: open for more building blocks, extendable
RE-CONSTRUCTING TRANSFORMATION LANGUAGES
RE-CONSTRUCTING TRANSFORMATION LANGUAGES

FUJABA for all Pattern
Algorithm 10 doSubDemoC.packetIn(π)

\[
\begin{align*}
\pi & \leftarrow \text{loadC.packetIn}(\pi) \\
\pi & \leftarrow \text{nextStepC.packetIn}(\pi) \\
\text{isSuccess} & \leftarrow \text{true} \\
\end{align*}
\]

return π

Algorithm 11 loadC.packetIn(π)

\[
\begin{align*}
\pi & \leftarrow \text{loadM.packetIn}(\pi) \\
\text{if not} & \text{loadM.isSuccess then} \\
\text{isSuccess} & \leftarrow \text{false} \\
\text{return} & \pi \\
\end{align*}
\]

Algorithm 12 makeChoiceC.packetIn(π)

\[
\begin{align*}
\pi & \leftarrow \text{makeChoiceM.packetIn}(\pi) \\
\text{if not} & \text{makeChoiceM.isSuccess then} \\
\text{isSuccess} & \leftarrow \text{false} \\
\text{return} & \pi \\
\end{align*}
\]

Algorithm 13 loadC.packetIn(π)

\[
\begin{align*}
\pi & \leftarrow \text{loadC.packetIn}(\pi) \\
\text{if not} & \text{loadC.isSuccess then} \\
\text{isSuccess} & \leftarrow \text{false} \\
\text{return} & \pi \\
\end{align*}
\]

nextStep

\[
\begin{align*}
\text{PostMatch(1).step} & \leftarrow \text{PostMatch(1).step} + 1 \\
\end{align*}
\]

Algorithm 14 makeChoiceL.packetIn(π)

\[
\begin{align*}
\pi & \leftarrow \text{makeChoiceL.packetIn}(\pi) \\
\text{if not} & \text{makeChoiceL.isSuccess then} \\
\text{isSuccess} & \leftarrow \text{true} \\
\text{return} & \pi \\
\end{align*}
\]

Algorithm 15 loadL.packetIn(π)

\[
\begin{align*}
\pi & \leftarrow \text{loadL.packetIn}(\pi) \\
\text{if not} & \text{loadL.isSuccess then} \\
\text{isSuccess} & \leftarrow \text{false} \\
\text{return} & \pi \\
\end{align*}
\]

Algorithm 16 loadW.packetIn(π)

\[
\begin{align*}
\pi & \leftarrow \text{loadW.packetIn}(\pi) \\
\text{if not} & \text{loadW.isSuccess then} \\
\text{isSuccess} & \leftarrow \text{false} \\
\text{return} & \pi \\
\end{align*}
\]

Algorithm 17 loadR.packetIn(π)

\[
\begin{align*}
\pi & \leftarrow \text{loadR.packetIn}(\pi) \\
\text{if not} & \text{loadR.isSuccess then} \\
\text{isSuccess} & \leftarrow \text{false} \\
\text{return} & \pi \\
\end{align*}
\]
Amalgamation rules: *Repotting the geraniums* [3]

“Repot all flowering geraniums whose pots have cracked”

MOTIF-CORE = DEVS + T-CORE [4]

```
Algorithm 14 primitive. externalTransition(e)
// Some pre-processing ...
if e received from ACancelIn then
  state.cancelIn(e)
end if
if e received from APacketIn then
  state.packetIn(e)
else if e received from ANextIn then
  // if defined
  state.nextIn(e)
end if
```

MOTIF-CORE: TIMED MTLS

MoTif AtomicRule [5]

- Time
- Exceptions

MOTIF-CORE: TIMED MTLS

GReAT Test/Case block [6]

- Asynchrony
- Parallelism

MOTIF-CORE: TIMED MTLS

More Readable: Repotting the geraniums
CONCLUSION

• Collection of MT primitives: T-Core
• Re-construction of existing MTLs (comparable)
• New-Construction of novel MTLs: MoTif-Core

• Future Work
  – Efficiently implement these primitives
  – Compare MoTif-Core with QVT-Core
Let’s discuss