De-/Re-constructing Model Transformation Languages

Eugene Syriani

Ph.D. Candidate in the Modelling, Simulation and Design Lab

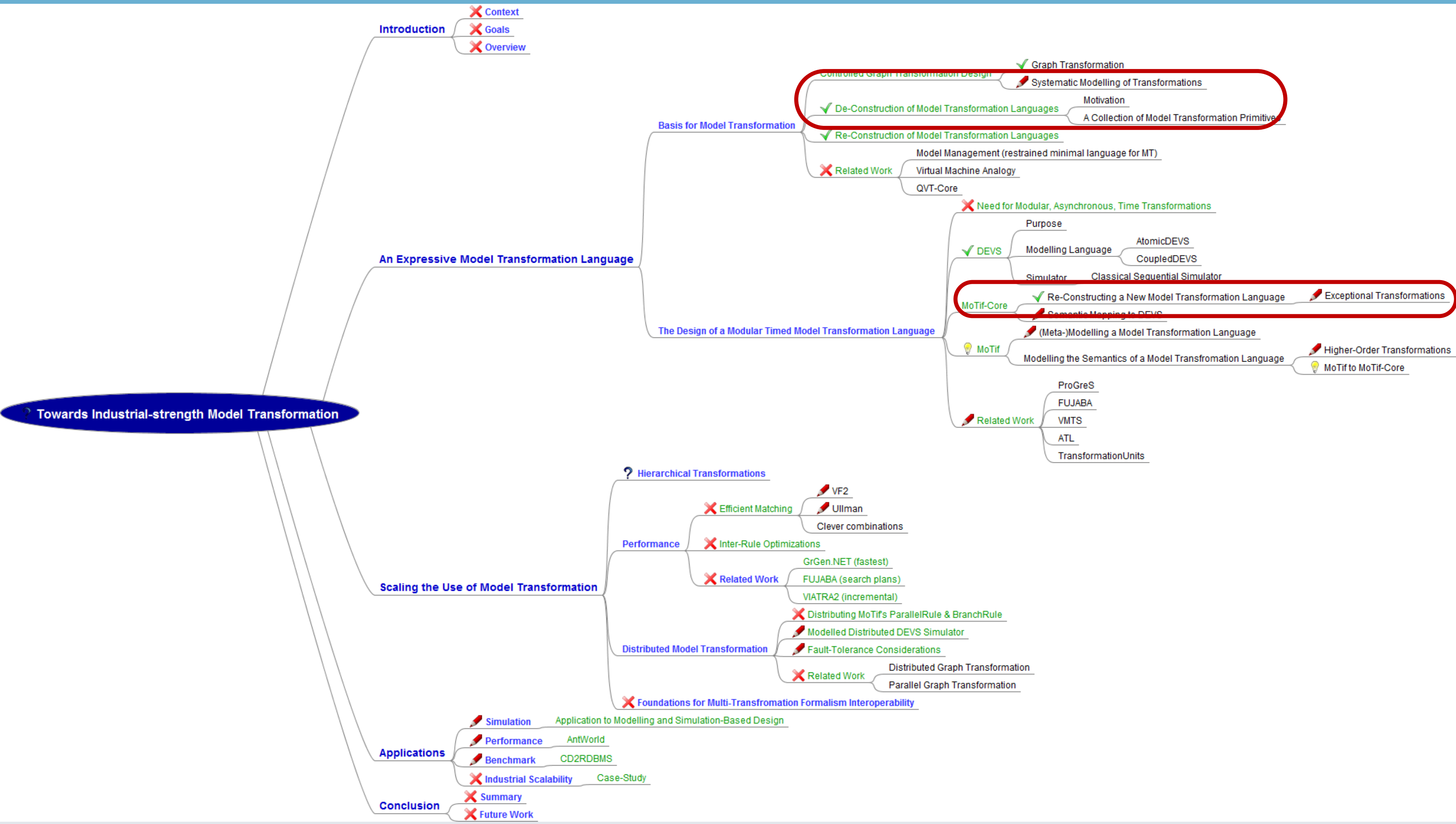
School of Computer Science

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



- Towards Industrial-strength Model Transformation
  - Introduction
    - ✗ Context
    - ✗ Goals
    - ✗ Overview
  - An Expressive Model Transformation Language
    - Basis for Model Transformation
      - Controlled Graph Transformation Design
        - ✓ Graph Transformation
        - ✎ Systematic Modelling of Transformations
      - ✓ De-Construction of Model Transformation Languages
        - Motivation
        - A Collection of Model Transformation Primitive
      - ✓ Re-Construction of Model Transformation Languages
      - ✗ Related Work
        - Model Management (restrained minimal language for MT)
        - Virtual Machine Analogy
        - QVT-Core
    - The Design of a Modular Timed Model Transformation Language
      - ✗ Need for Modular, Asynchronous, Time Transformations
      - ✓ DEVS
        - Purpose
        - Modelling Language
          - AtomicDEVS
          - CoupledDEVS
        - Simulator
          - Classical Sequential Simulator
      - MoTif-Core
        - ✓ Re-Constructing a New Model Transformation Language
          - ✎ Exceptional Transformations
        - ✎ Semantic Mapping to DEVS
      - 💡 MoTif
        - ✎ (Meta-)Modelling a Model Transformation Language
        - Modelling the Semantics of a Model Transfromation Language
          - ✎ Higher-Order Transformations
          - 💡 MoTif to MoTif-Core
      - ✎ Related Work
        - ProGreS
        - FUJABA
        - VMTS
        - ATL
        - TransformationUnits
  - Scaling the Use of Model Transformation
    - ? Hierarchical Transformations
    - Performance
      - ✗ Efficient Matching
        - ✎ VF2
        - ✎ Ullman
        - Clever combinations
      - ✗ Inter-Rule Optimizations
      - ✗ Related Work
        - GrGen.NET (fastest)
        - FUJABA (search plans)
        - VIATRA2 (incremental)
    - Distributed Model Transformation
      - ✗ Distributing MoTif's ParallelRule & BranchRule
      - ✎ Modelled Distributed DEVS Simulator
      - ✎ Fault-Tolerance Considerations
      - ✗ Related Work
        - Distributed Graph Transformation
        - Parallel Graph Transformation
    - ✗ Foundations for Multi-Transfromation Formalism Interoperability
  - Applications
    - ✎ Simulation
      - Application to Modelling and Simulation-Based Design
    - ✎ Performance
      - AntWorld
    - ✎ Benchmark
      - CD2RDBMS
    - ✗ Industrial Scalability
      - Case-Study
  - Conclusion
    - ✗ Summary
    - ✗ Future Work

IN THE CONTEXT

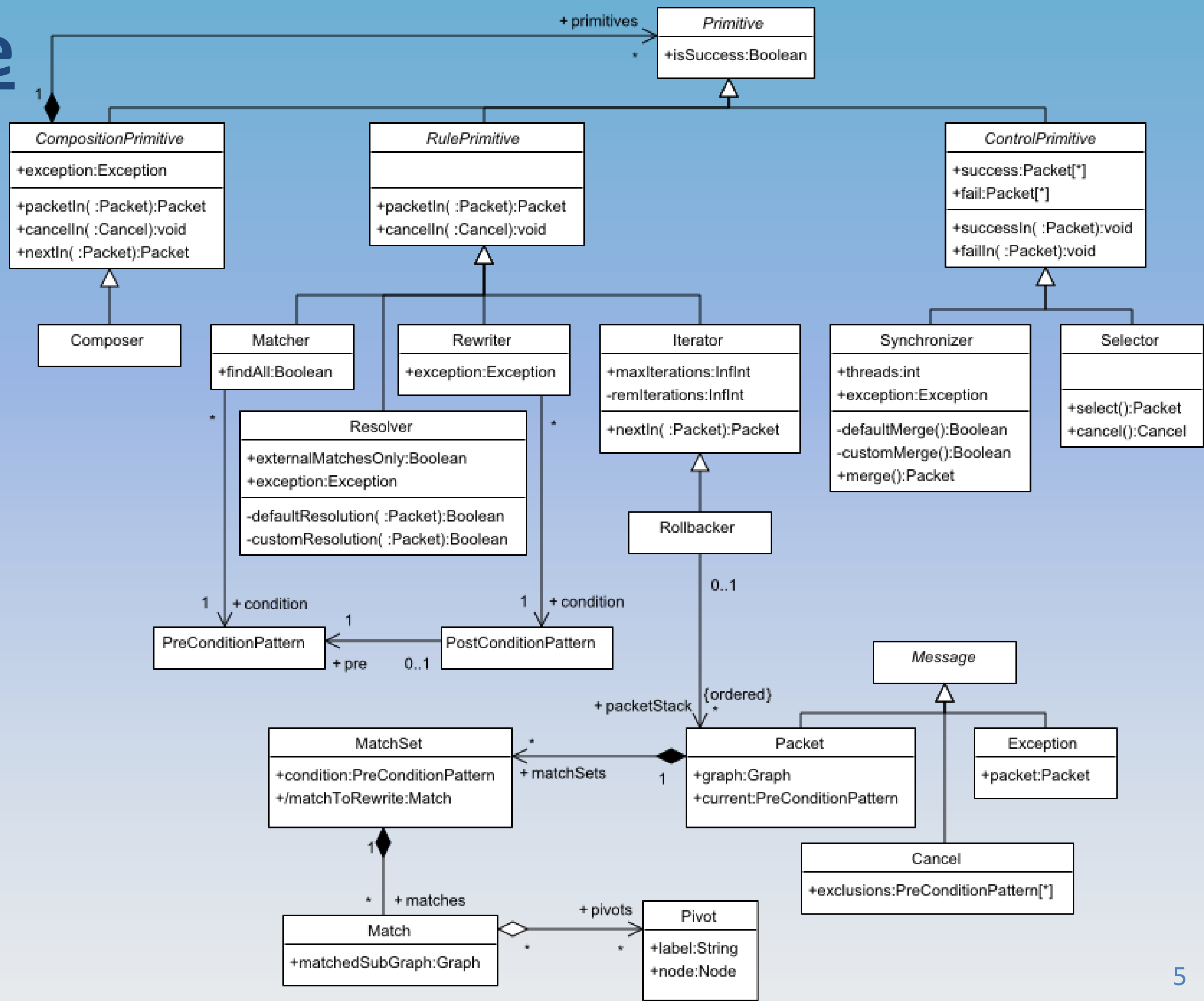
- Many different model transfromation languages (MTLs)
  - Features [1]: atomicity, sequencing, branching, looping, non-determinism, recursion, parallelism, back-tracking, hierarchy, time
  - Transformation rule: matching + rewriting + validation
- Hard to
  - Compare *expressiveness*
  - Provide framework 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

[1] Syriani, E. and Vangheluwe, H. (2009) Matters of model transformation. Technical Report SOCS-TR-2009.2. McGill University, School of Computer Science.

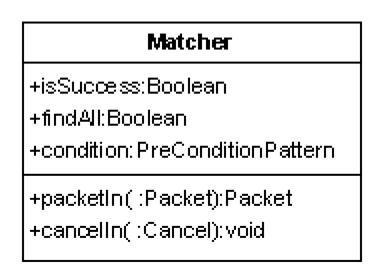
DE-CONSTRUCTING TRANSFORMATION LANGUAGES

T-Core Module

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



DE-CONSTRUCTING TRANSFORMATION LANGUAGES

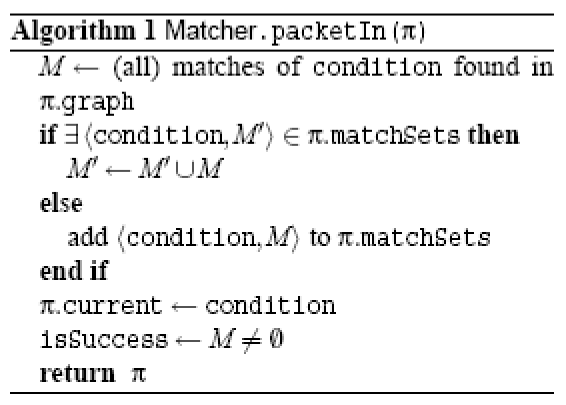


Matcher

+isSuccess:Boolean
+findAll:Boolean
+condition:PreConditionPattern

+packetIn(:Packet):Packet
+cancelIn(:Cancel):void

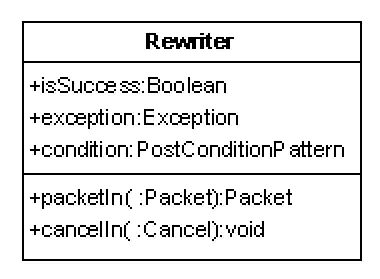
Matcher



```
Algorithm 1 Matcher.packetIn($\pi$)
  $M \leftarrow$ (all) matches of condition found in $\pi$.graph
  if $\exists \langle$condition$, M' \rangle \in \pi$.matchSets then
    $M' \leftarrow M' \cup M$
  else
    add $\langle$condition$, M \rangle$ to $\pi$.matchSets
  end if
  $\pi$.current $\leftarrow$ condition
  isSuccess $\leftarrow M \neq \emptyset$
  return $\pi$
```

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

DE-CONSTRUCTING TRANSFORMATION LANGUAGES

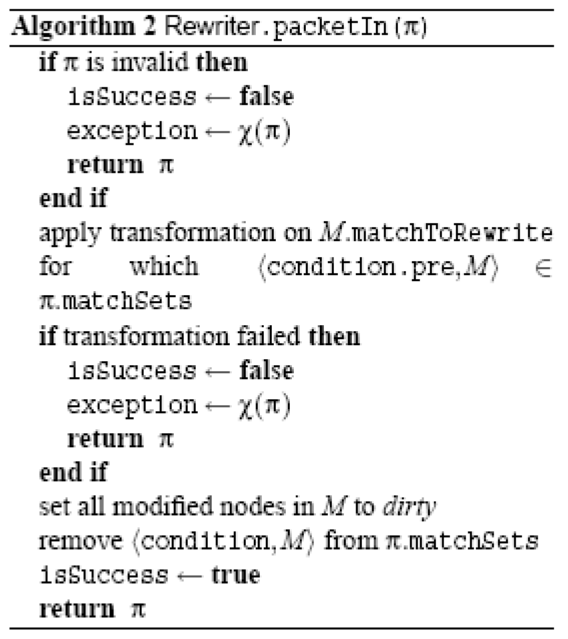


Rewriter

+isSuccess:Boolean
+exception:Exception
+condition:PostConditionPattern

+packetIn(:Packet):Packet
+cancelIn(:Cancel):void

Rewriter



Algorithm 2 Rewriter.packetIn (π)

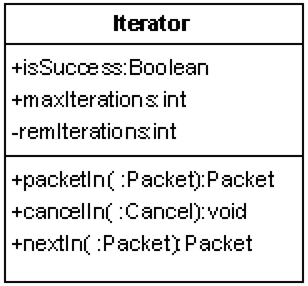
```
if $\pi$ is invalid then
    isSuccess ← false
    exception ← $\chi(\pi)$
    return $\pi$
end if
apply transformation on $M$.matchToRewrite for which ⟨condition.pre,$M$⟩ ∈ $\pi$.matchSets
if transformation failed then
    isSuccess ← false
    exception ← $\chi(\pi)$
    return $\pi$
end if
set all modified nodes in $M$ to *dirty*
remove ⟨condition,$M$⟩ from $\pi$.matchSets
isSuccess ← true
return $\pi$
```

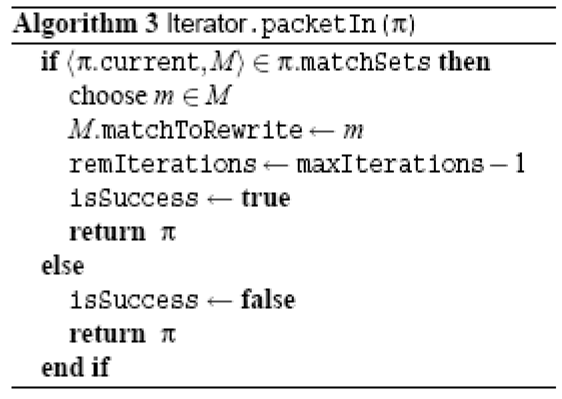
1. **Check validity of packet**
2. **Apply transformation**
3. **Propgate changes in all match sets**
4. **Consume match**

Exception possible!

DE-CONSTRUCTING TRANSFORMATION LANGUAGES

Iterator

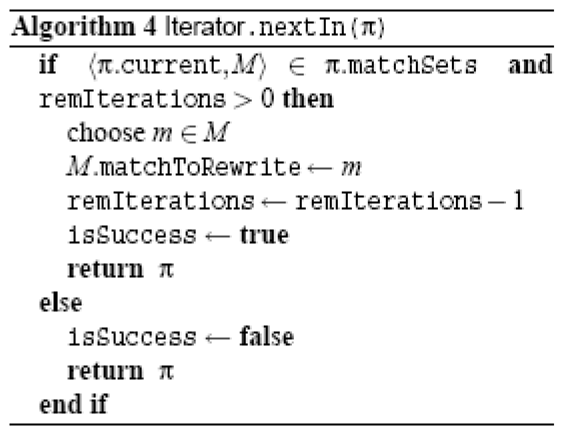




Algorithm 3 Iterator.packetIn(π)

```
if $\langle \pi.\texttt{current}, M \rangle \in \pi.\texttt{matchSets}$ then
    choose $m \in M$
    $M.\texttt{matchToRewrite} \leftarrow m$
    $\texttt{remIterations} \leftarrow \texttt{maxIterations} - 1$
    $\texttt{isSuccess} \leftarrow$ true
    return $\pi$
else
    $\texttt{isSuccess} \leftarrow$ false
    return $\pi$
end if
```

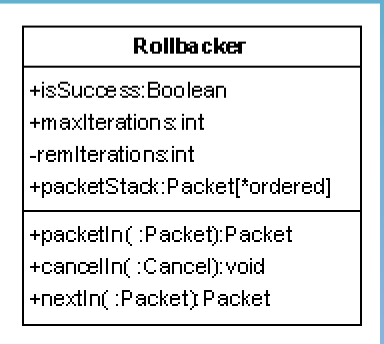
1. Check if match set is not empty
2. Randomly choose a match



Algorithm 4 Iterator.nextIn(π)

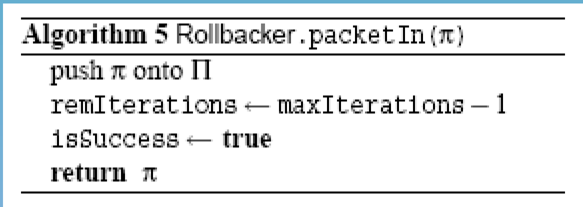
```
if $\langle \pi.\texttt{current}, M \rangle \in \pi.\texttt{matchSets}$ and $\texttt{remIterations} > 0$ then
    choose $m \in M$
    $M.\texttt{matchToRewrite} \leftarrow m$
    $\texttt{remIterations} \leftarrow \texttt{remIterations} - 1$
    $\texttt{isSuccess} \leftarrow$ true
    return $\pi$
else
    $\texttt{isSuccess} \leftarrow$ false
    return $\pi$
end if
```

DE-CONSTRUCTING TRANSFORMATION LANGUAGES



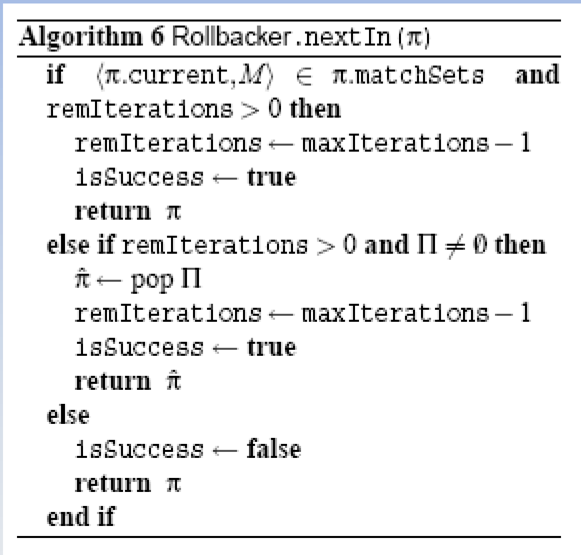
Rollbacker
+isSuccess:Boolean +maxIterations:int -remIterations:int +packetStack:Packet[*ordered]
+packetIn(:Packet):Packet +cancelIn(:Cancel):void +nextIn(:Packet):Packet

Rollbacker



```
Algorithm 5 Rollbacker.packetIn(π)
  push $\pi$ onto $\Pi$
  remIterations ← maxIterations − 1
  isSuccess ← true
  return $\pi$
```

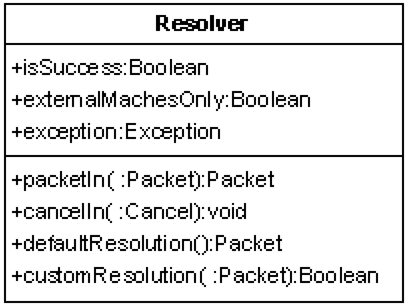
1. Push packet onto stack



```
Algorithm 6 Rollbacker.nextIn(π)
  if ⟨$\pi$.current, $M$⟩ ∈ $\pi$.matchSets and remIterations > 0 then
    remIterations ← maxIterations − 1
    isSuccess ← true
    return $\pi$
  else if remIterations > 0 and $\Pi \neq \emptyset$ then
    $\hat{\pi}$ ← pop $\Pi$
    remIterations ← maxIterations − 1
    isSuccess ← true
    return $\hat{\pi}$
  else
    isSuccess ← false
    return $\pi$
  end if
```

1. Match set not empty:
   there are matches left (pass on)
2. No match set:
   back-track to previous state

DE-CONSTRUCTING TRANSFORMATION LANGUAGES



Resolver

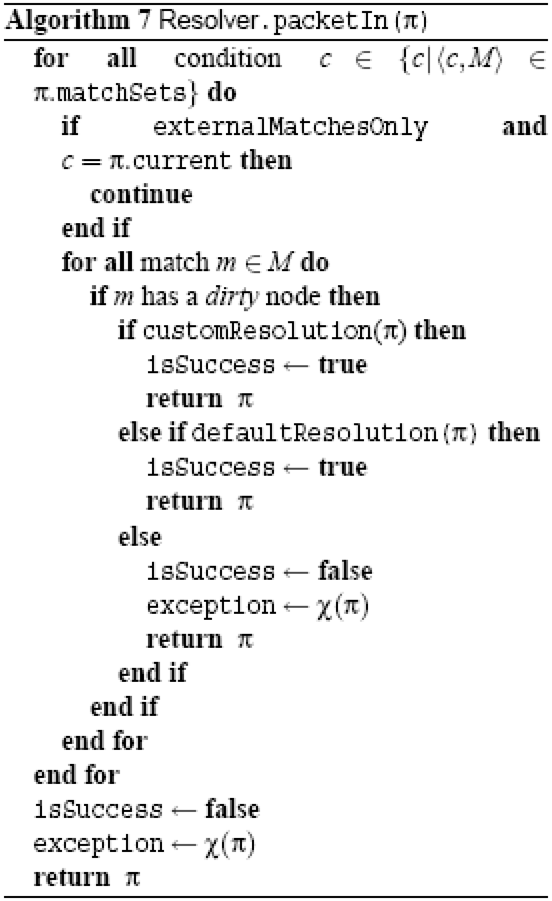
+isSuccess:Boolean
+externalMachesOnly:Boolean
+exception:Exception

+packetIn(:Packet):Packet
+cancelIn(:Cancel):void
+defaultResolution():Packet
+customResolution(:Packet):Boolean

Resolver

1. **Conservative check for potential conflict between different matches in match sets (parameter)**
2. **Customizable resolution function**

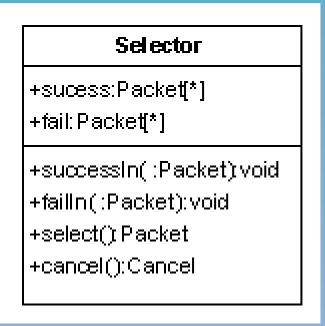
Exception possible!



```
Algorithm 7 Resolver.packetIn(π)
for all condition $c \in \{c|\langle c,M\rangle \in \pi$.matchSets$\}$ do
  if externalMatchesOnly and $c = \pi$.current then
    continue
  end if
  for all match $m \in M$ do
    if $m$ has a dirty node then
      if customResolution($\pi$) then
        isSuccess ← true
        return $\pi$
      else if defaultResolution($\pi$) then
        isSuccess ← true
        return $\pi$
      else
        isSuccess ← false
        exception ← $\chi(\pi)$
        return $\pi$
      end if
    end if
  end for
end for
isSuccess ← false
exception ← $\chi(\pi)$
return $\pi$
```

DE-CONSTRUCTING TRANSFORMATION LANGUAGES

Selector



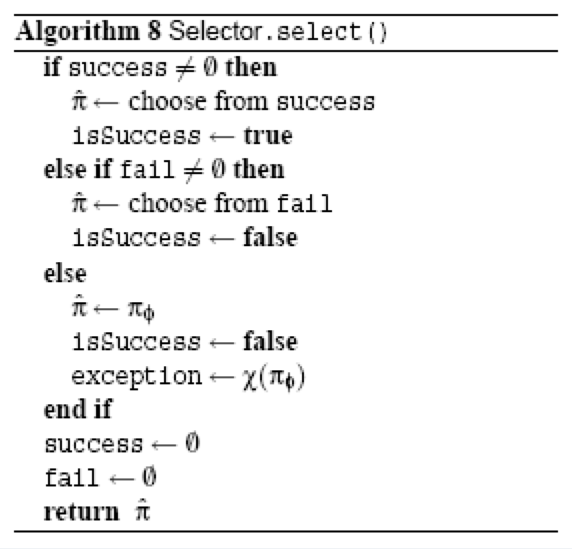
Selector

+sucess:Packet[*]
+fail: Packet[*]

+successIn(:Packet):void
+failIn(:Packet):void
+select():Packet
+cancel():Cancel

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

Exception possible!



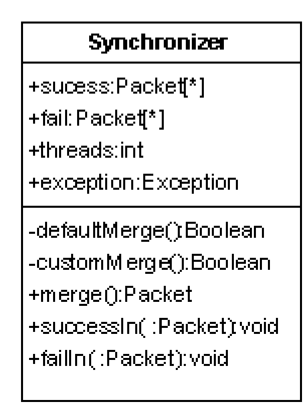
```
Algorithm 8 Selector.select()
  if success $\neq \emptyset$ then
    $\hat{\pi} \leftarrow$ choose from success
    isSuccess $\leftarrow$ true
  else if fail $\neq \emptyset$ then
    $\hat{\pi} \leftarrow$ choose from fail
    isSuccess $\leftarrow$ false
  else
    $\hat{\pi} \leftarrow \pi_\phi$
    isSuccess $\leftarrow$ false
    exception $\leftarrow \chi(\pi_\phi)$
  end if
  success $\leftarrow \emptyset$
  fail $\leftarrow \emptyset$
  return $\hat{\pi}$
```

DE-CONSTRUCTING TRANSFORMATION LANGUAGES

Synchronizer

1. `successIn`: add to `success` set
2. `failIn`: add to `fail` set
3. Merge only if all threads succeeded
4. Customizable merge function

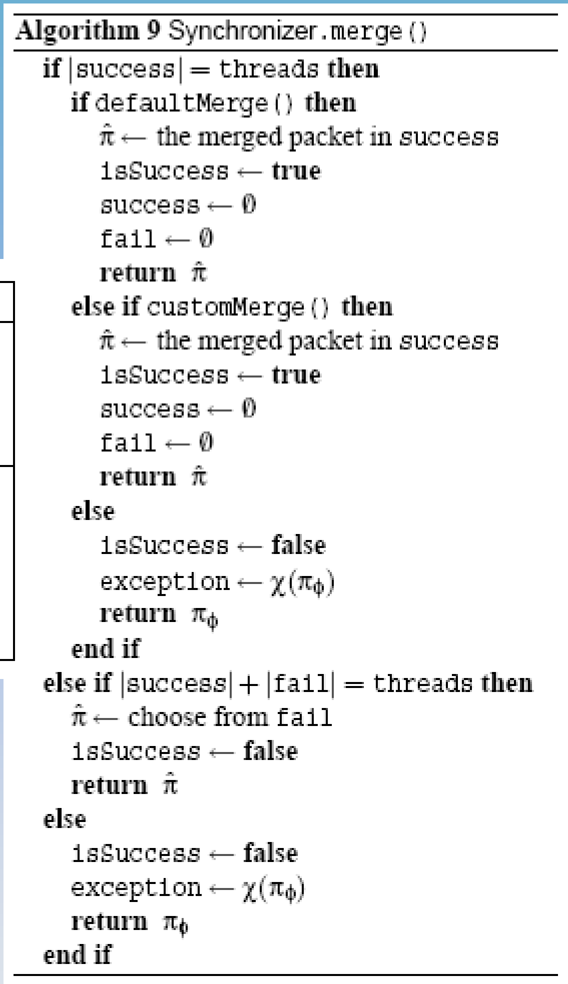
Exception possible!



Synchronizer

+sucess:Packet[*]
+fail:Packet[*]
+threads:int
+exception:Exception

-defaultMerge():Boolean
-customMerge():Boolean
+merge():Packet
+successIn(:Packet):void
+failIn(:Packet):void

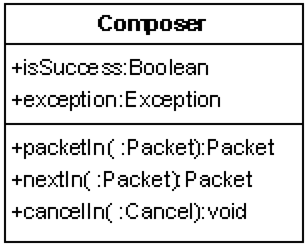


Algorithm 9 Synchronizer.merge()

```
if $|\texttt{success}| = \texttt{threads}$ then
    if defaultMerge() then
        $\hat{\pi} \leftarrow$ the merged packet in success
        isSuccess $\leftarrow$ true
        success $\leftarrow \emptyset$
        fail $\leftarrow \emptyset$
        return $\hat{\pi}$
    else if customMerge() then
        $\hat{\pi} \leftarrow$ the merged packet in success
        isSuccess $\leftarrow$ true
        success $\leftarrow \emptyset$
        fail $\leftarrow \emptyset$
        return $\hat{\pi}$
    else
        isSuccess $\leftarrow$ false
        exception $\leftarrow \chi(\pi_\phi)$
        return $\pi_\phi$
    end if
else if $|\texttt{success}| + |\texttt{fail}| = \texttt{threads}$ then
    $\hat{\pi} \leftarrow$ choose from fail
    isSuccess $\leftarrow$ false
    return $\hat{\pi}$
else
    isSuccess $\leftarrow$ false
    exception $\leftarrow \chi(\pi_\phi)$
    return $\pi_\phi$
end if
```

DE-CONSTRUCTING TRANSFORMATION LANGUAGES

Composer



Composer

+isSuccess:Boolean
+exception:Exception

+packetIn(:Packet):Packet
+nextIn(:Packet):Packet
+cancelIn(:Cancel):void

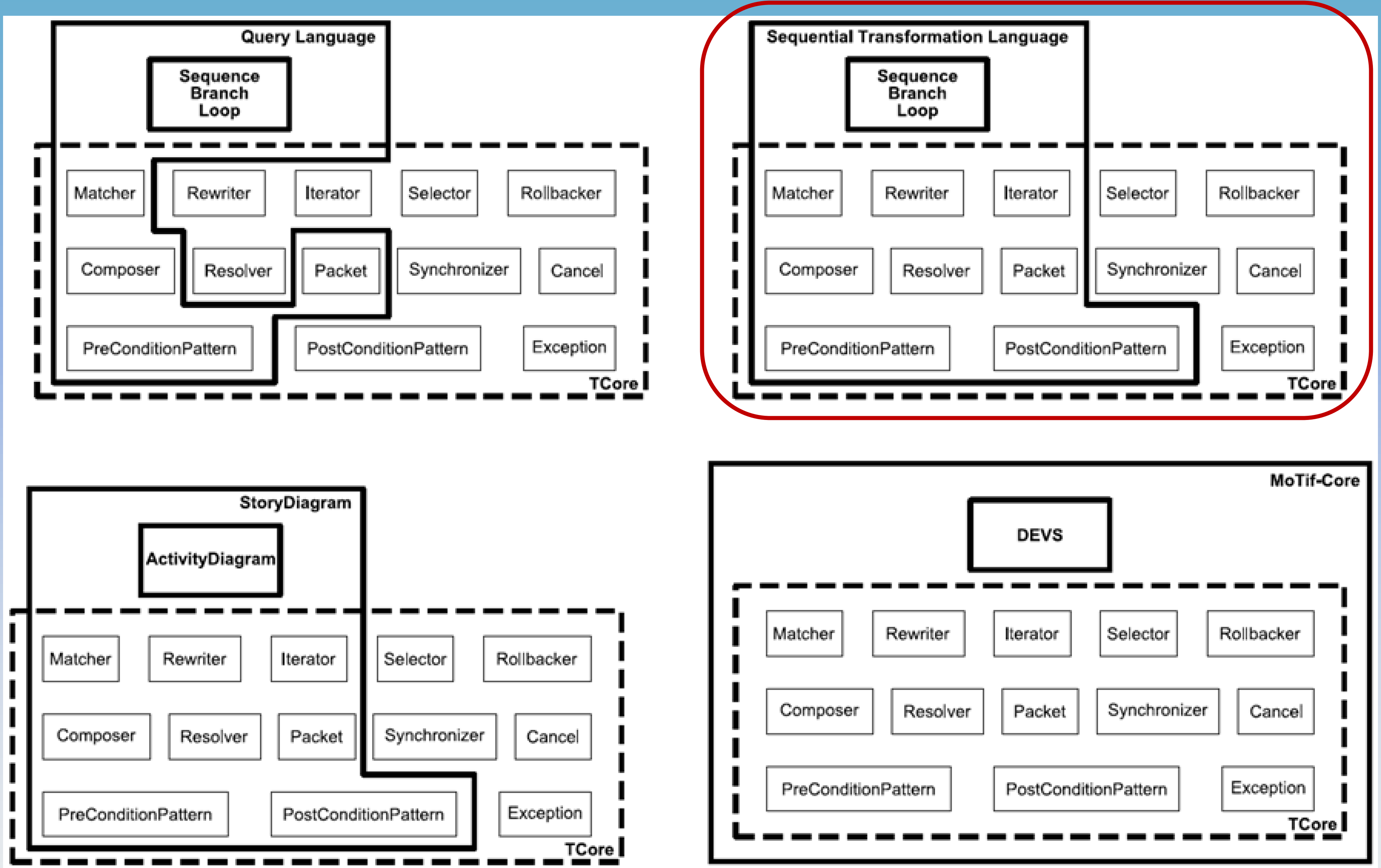
1. **Meaningfully composes its sub-primitives**
2. **User-defined composition**

DE-CONSTRUCTING TRANSFORMATION LANGUAGES

Motivating T-Core

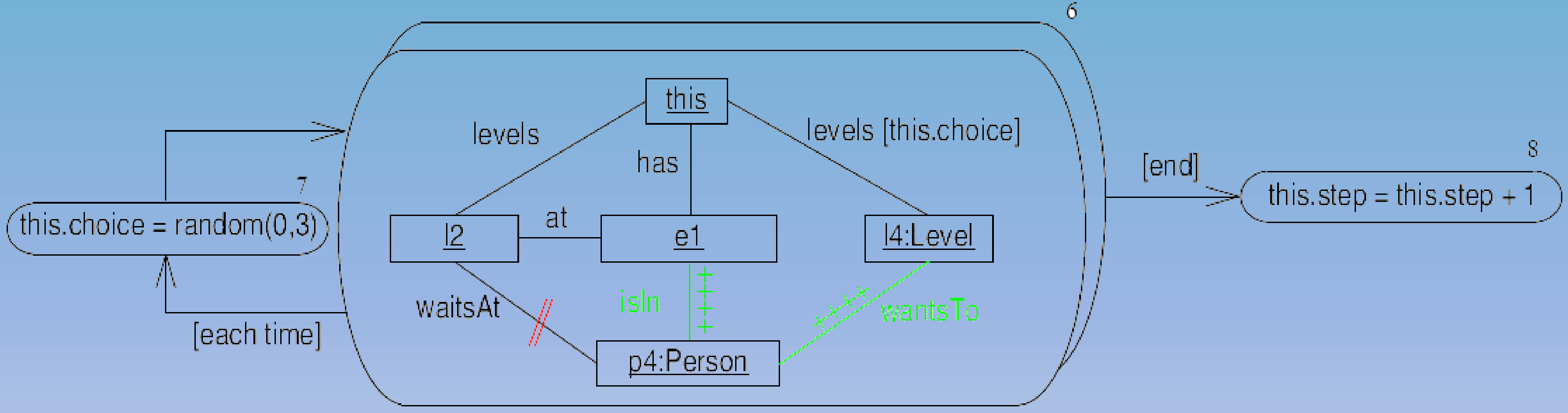
- De-construct up to what level?
- What to include, what to exclude?
  - Pre/PostConditionPattern: rules, bi-directional, functions
  - Separation match/rewrite: queries, nested transformaitons
  - Packet: sufficient info to be processed by each primitive, designed for concurrent transformations
  -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

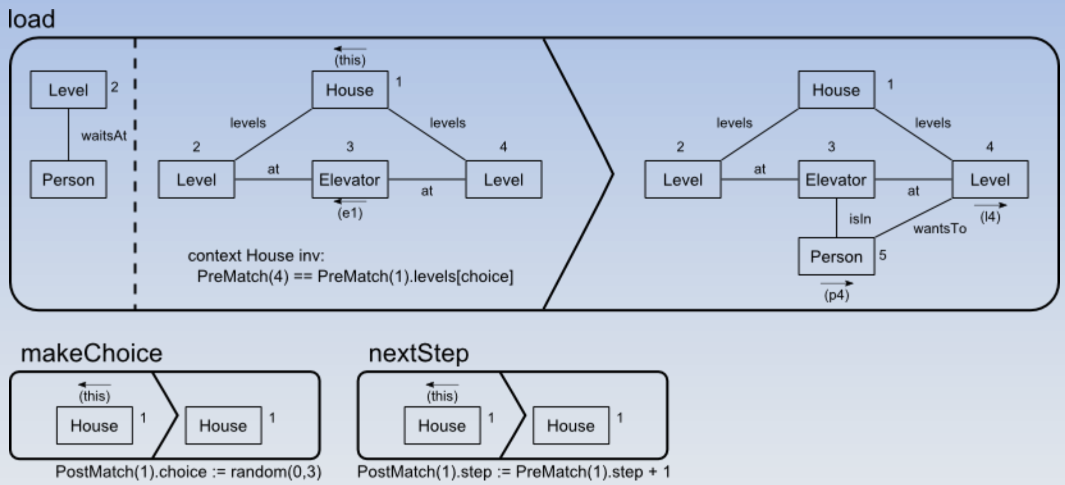
RE-CONSTRUCTING TRANSFORMATION LANGUAGES



RE-CONSTRUCTING TRANSFORMATION LANGUAGES

FUJABA for-all Pattern [2]



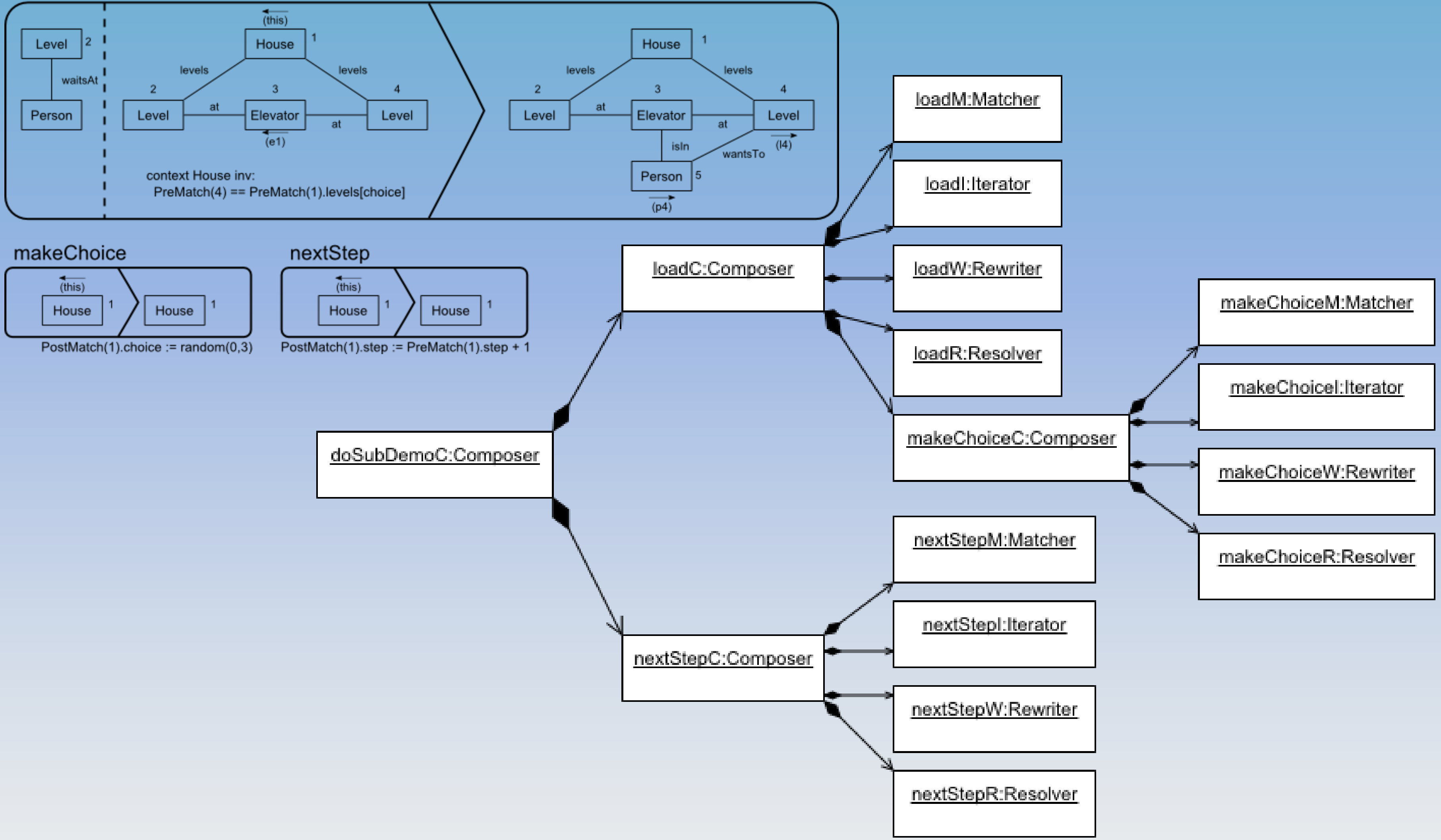


[2] Fischer, T., et. al., (2000) Story diagrams: A new graph rewrite language based on the UML and Java. In Ehrig, H., et al., (eds.), Theory and Application of Graph Transformations, LNCS, 1764, pp. 296–309. Springer-Verlag.

RE-CONSTRUCTING TRANSFORMATION LANGUAGES

FUJABA for-all Pattern

load



context House inv:
PreMatch(4) == PreMatch(1).levels[choice]

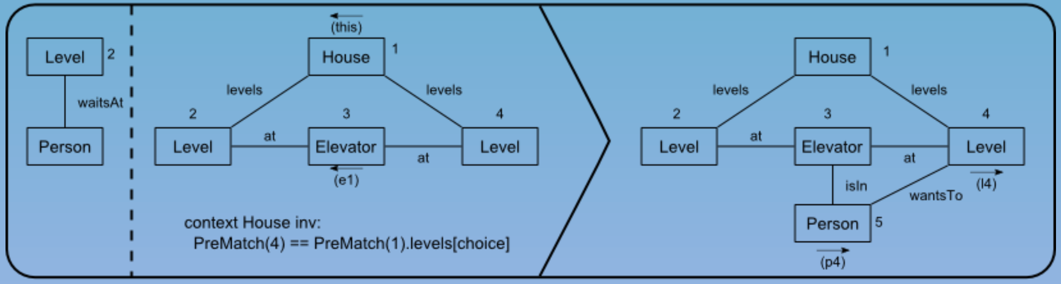
PostMatch(1).choice := random(0,3)

PostMatch(1).step := PreMatch(1).step + 1

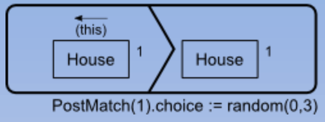
RE-CONSTRUCTING TRANSFORMATION LANGUAGES

FUJABA for-all Pattern

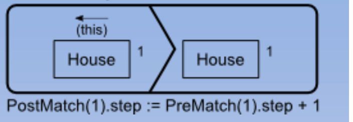
load

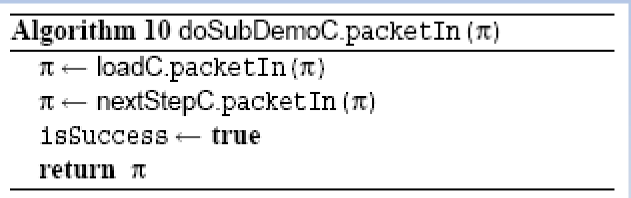


makeChoice



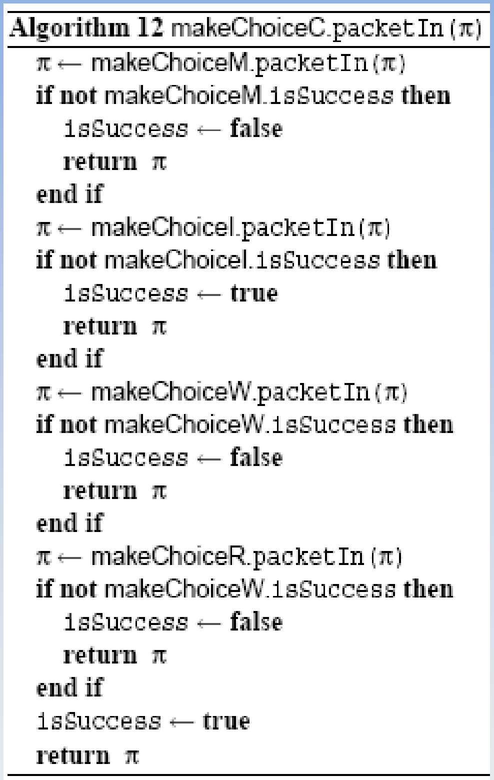
nextStep





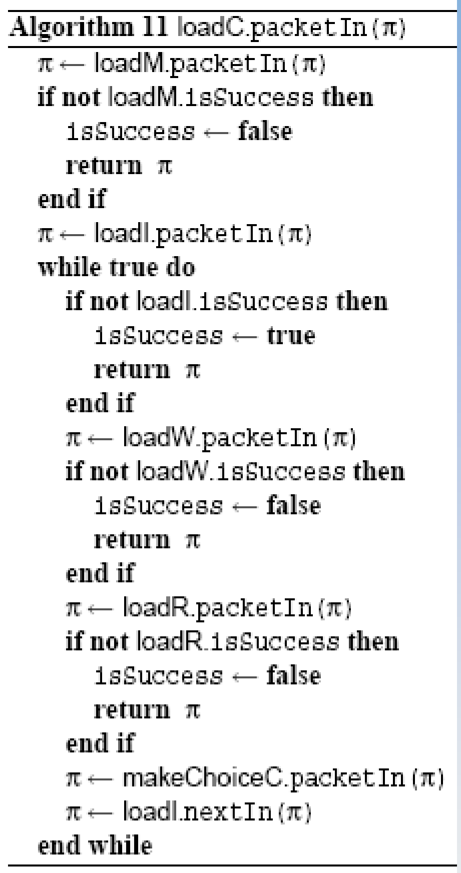
Algorithm 10 doSubDemoC.packetIn(π)

```
π ← loadC.packetIn(π)
π ← nextStepC.packetIn(π)
isSuccess ← true
return π
```



Algorithm 12 makeChoiceC.packetIn(π)

```
π ← makeChoiceM.packetIn(π)
if not makeChoiceM.isSuccess then
    isSuccess ← false
    return π
end if
π ← makeChoiceI.packetIn(π)
if not makeChoiceI.isSuccess then
    isSuccess ← true
    return π
end if
π ← makeChoiceW.packetIn(π)
if not makeChoiceW.isSuccess then
    isSuccess ← false
    return π
end if
π ← makeChoiceR.packetIn(π)
if not makeChoiceW.isSuccess then
    isSuccess ← false
    return π
end if
isSuccess ← true
return π
```



Algorithm 11 loadC.packetIn(π)

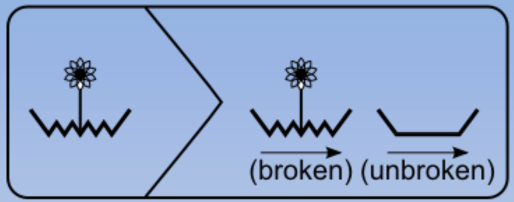
```
π ← loadM.packetIn(π)
if not loadM.isSuccess then
    isSuccess ← false
    return π
end if
π ← loadI.packetIn(π)
while true do
    if not loadI.isSuccess then
        isSuccess ← true
        return π
    end if
    π ← loadW.packetIn(π)
    if not loadW.isSuccess then
        isSuccess ← false
        return π
    end if
    π ← loadR.packetIn(π)
    if not loadR.isSuccess then
        isSuccess ← false
        return π
    end if
    π ← makeChoiceC.packetIn(π)
    π ← loadI.nextIn(π)
end while
```

RE-CONSTRUCTING TRANSFORMATION LANGUAGES

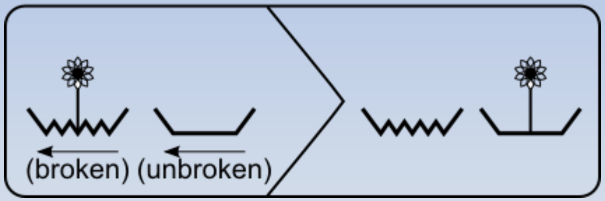
Amalgamation rules: *Repotting the geraniums* [3]

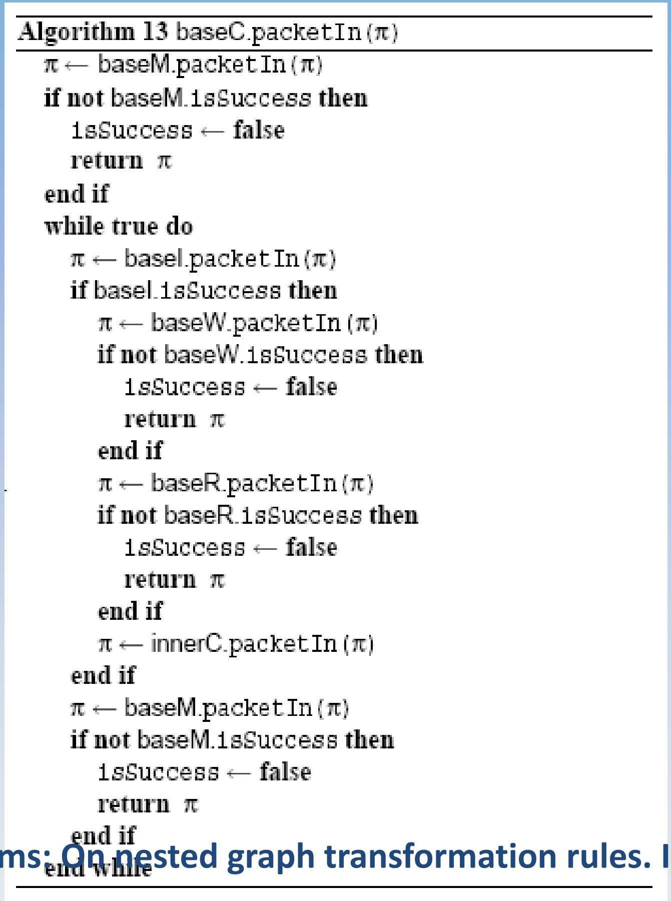
"Repot all flowering geraniums whose pots have cracked"

base



inner

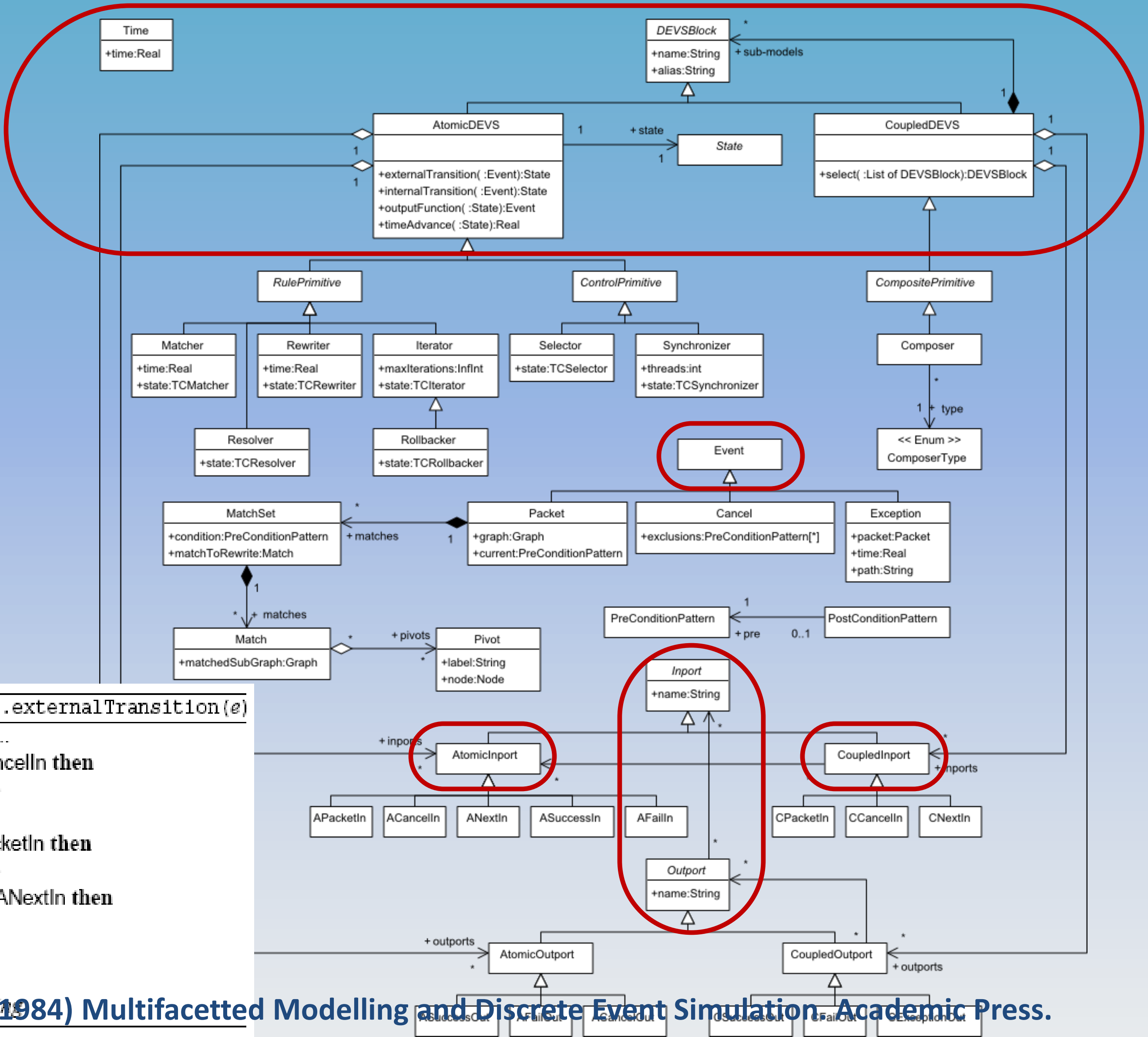




```
Algorithm 13 baseC.packetIn(π)
  π ← baseM.packetIn(π)
  if not baseM.isSuccess then
    isSuccess ← false
    return π
  end if
  while true do
    π ← baseI.packetIn(π)
    if baseI.isSuccess then
      π ← baseW.packetIn(π)
      if not baseW.isSuccess then
        isSuccess ← false
        return π
      end if
      π ← baseR.packetIn(π)
      if not baseR.isSuccess then
        isSuccess ← false
        return π
      end if
      π ← innerC.packetIn(π)
    end if
    π ← baseM.packetIn(π)
    if not baseM.isSuccess then
      isSuccess ← false
      return π
    end if
  end while
```

[3] Rensink, A. and Kuperus, J.-H. (2009) Repotting the geraniums: On nested graph transformation rules. In Margaria, T., Padberg, J., and Taentzer, G. (eds.), GT-VMT'09, EASST.

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



Algorithm 14 `primitive.externalTransition(e)`

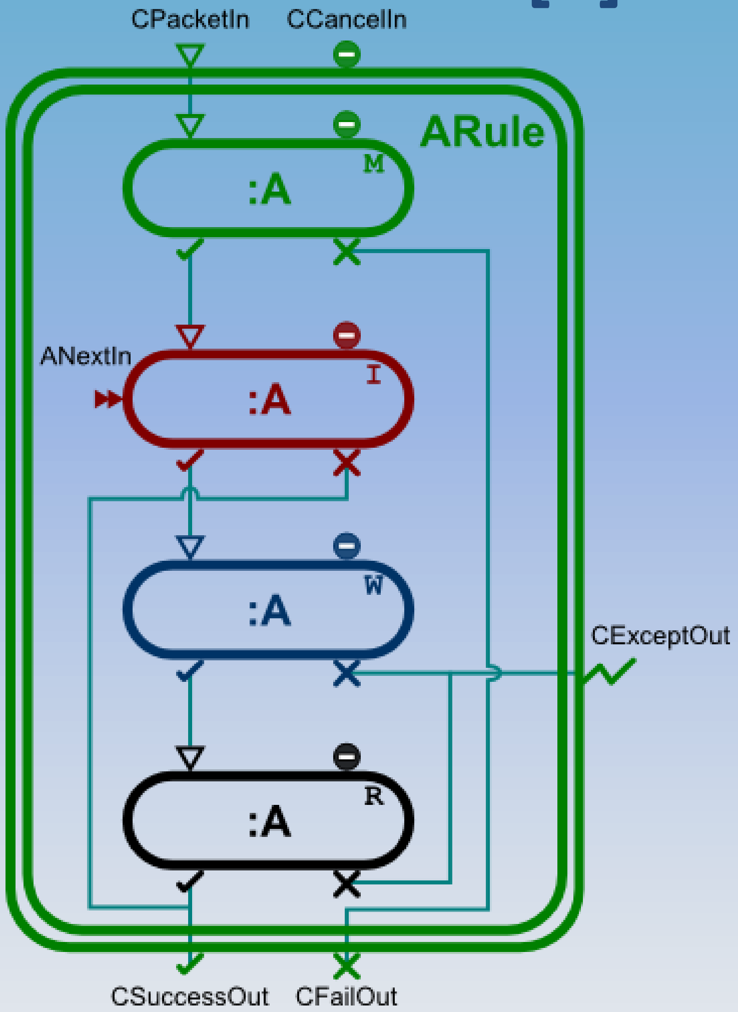
```
// Some pre-processing...
if e received from ACancelIn then
    state.cancelIn(e)
end if
if e received from APacketIn then
    state.packetIn(e)
else if e received from ANextIn then
    // if defined
    state.nextIn(e)
end if
// Some post-processing
```

[4] Zeigler, B. P. (1984) Multifacetted Modelling and Discrete Event Simulation. Academic Press.

MOTIF-CORE: TIMED MTLS

MoTif AtomicRule [5]

- Time
- Exceptions

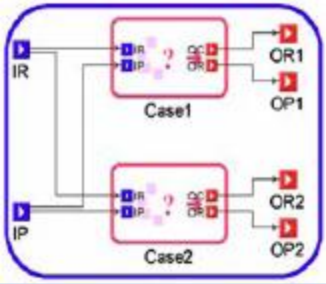


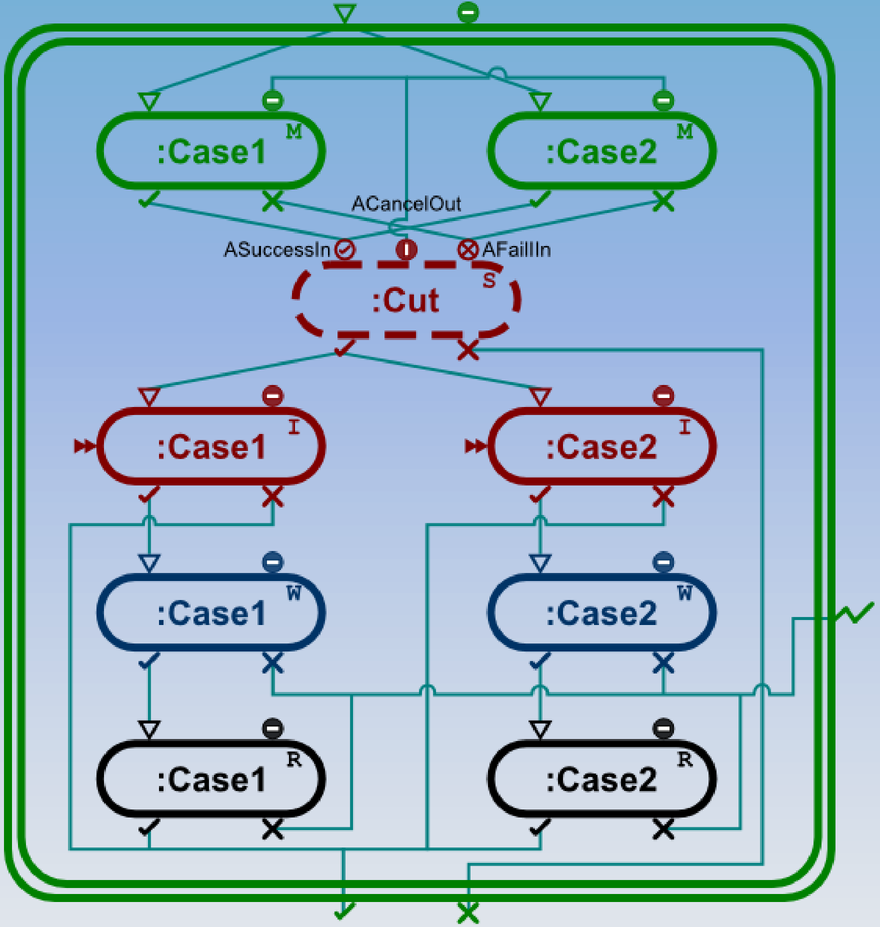
[5] Syriani, E. and Vangheluwe, H. (2009) Discrete-Event Modeling and Simulation: Theory and Applications. CRC Press, Boca Raton (USA).

MOTIF-CORE: TIMED MTLS

GReAT Test/Case block [6]

- Asynchrony
- Parallelism

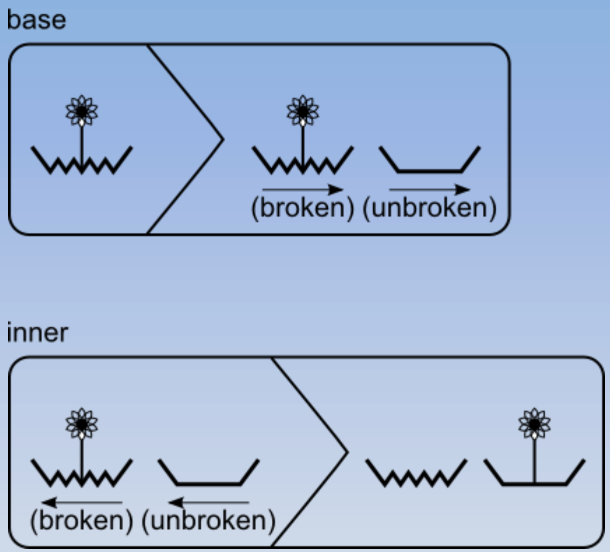


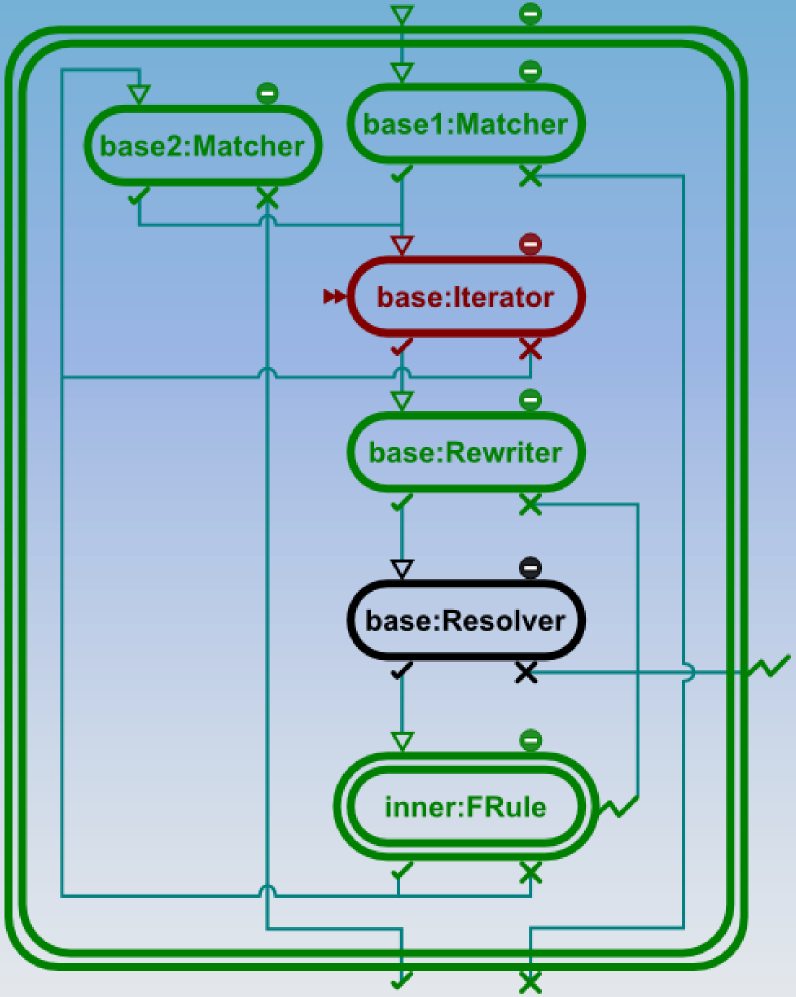


[6] Agrawal, A., Karsai, G., Kalmar, Z., Neema, S., Shi, F., and Vizhanyo, A. (2006) The design of a language for model transformations. SoSym, 5, 261–288.

MOTIF-CORE: TIMED MTLS

More Readable: *Repotting the geraniums*





CONCLUSION

- Collection of MT primtives: 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