Towards a Step Semantics for Story-Driven Modelling

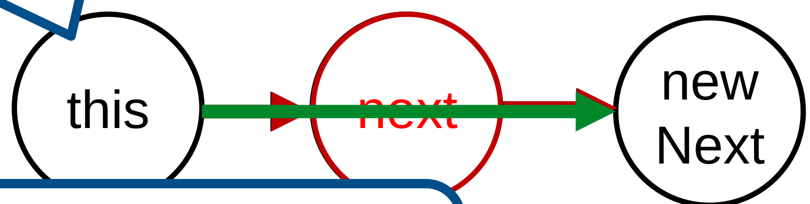
GaM'16, Eindhoven, The Netherlands

Géza Kulcsár (TU Darmstadt, Germany)
Anthony Anjorin (Paderborn University, Germany)

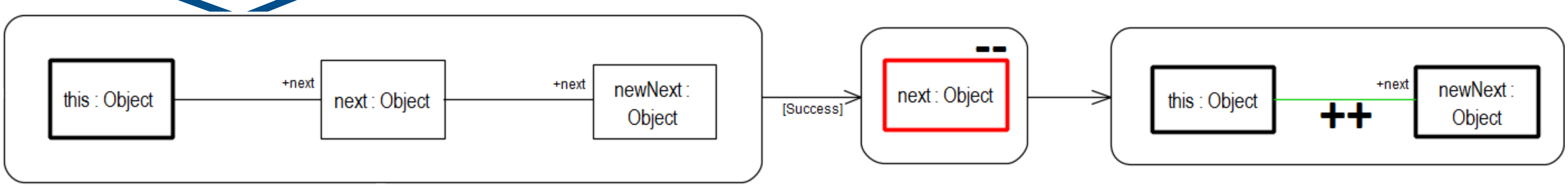
geza.kulcsar@es.tu-darmstadt.de

What is Story-Driven Modelling (SDM)?

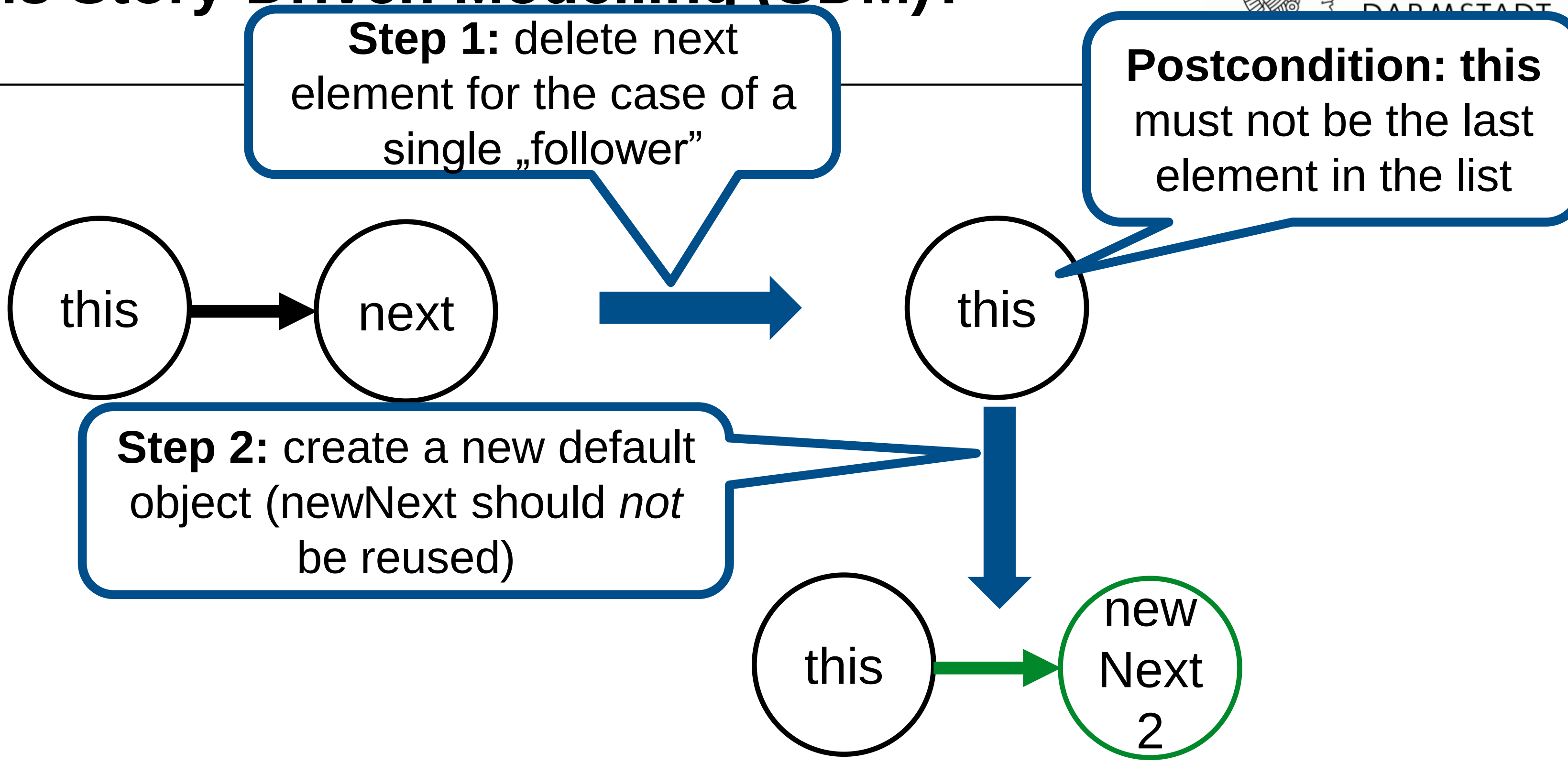
this.deleteNextObject():
delete the next element of an ordered list

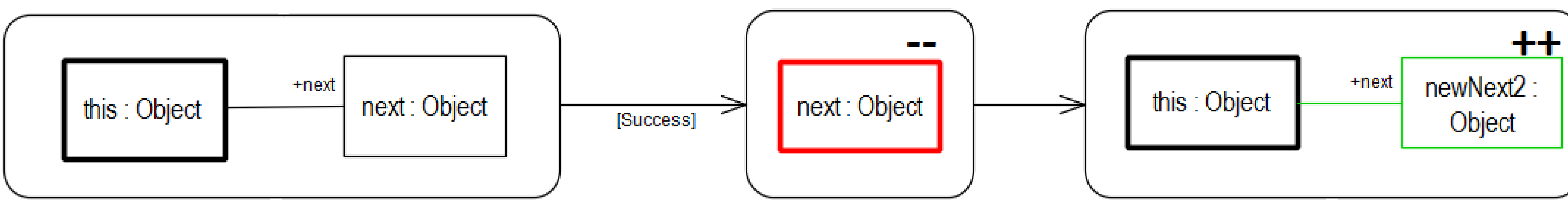


Graph pattern for matching **this** and two subsequent elements



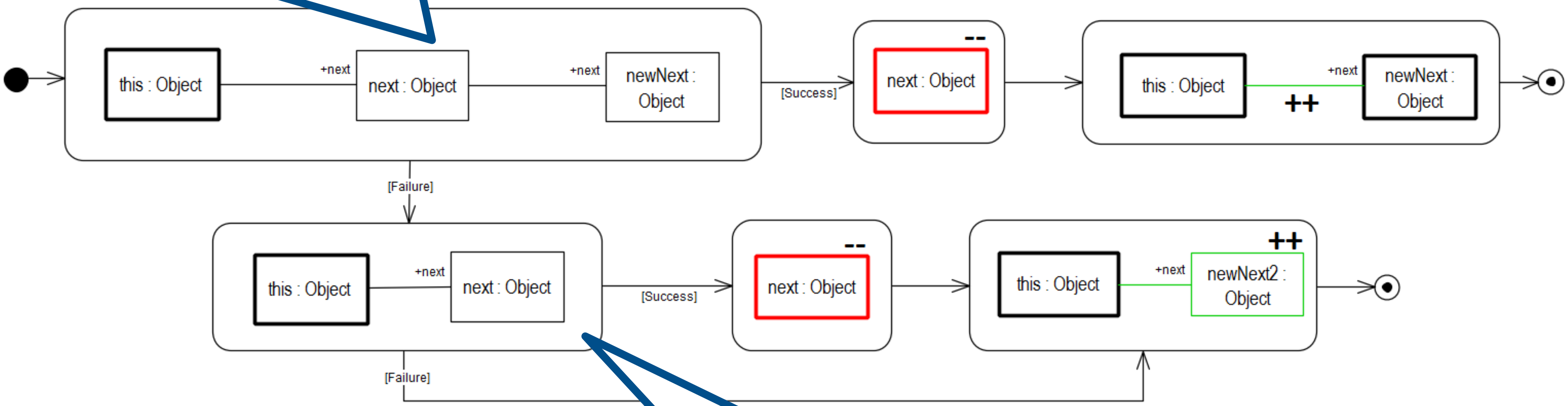
What is Story-Driven Modelling (SDM)?





What is Story-Driven Modelling (SDM)?

Complete specification in concrete syntax (historically, UML-like visual syntax, textual is also possible)

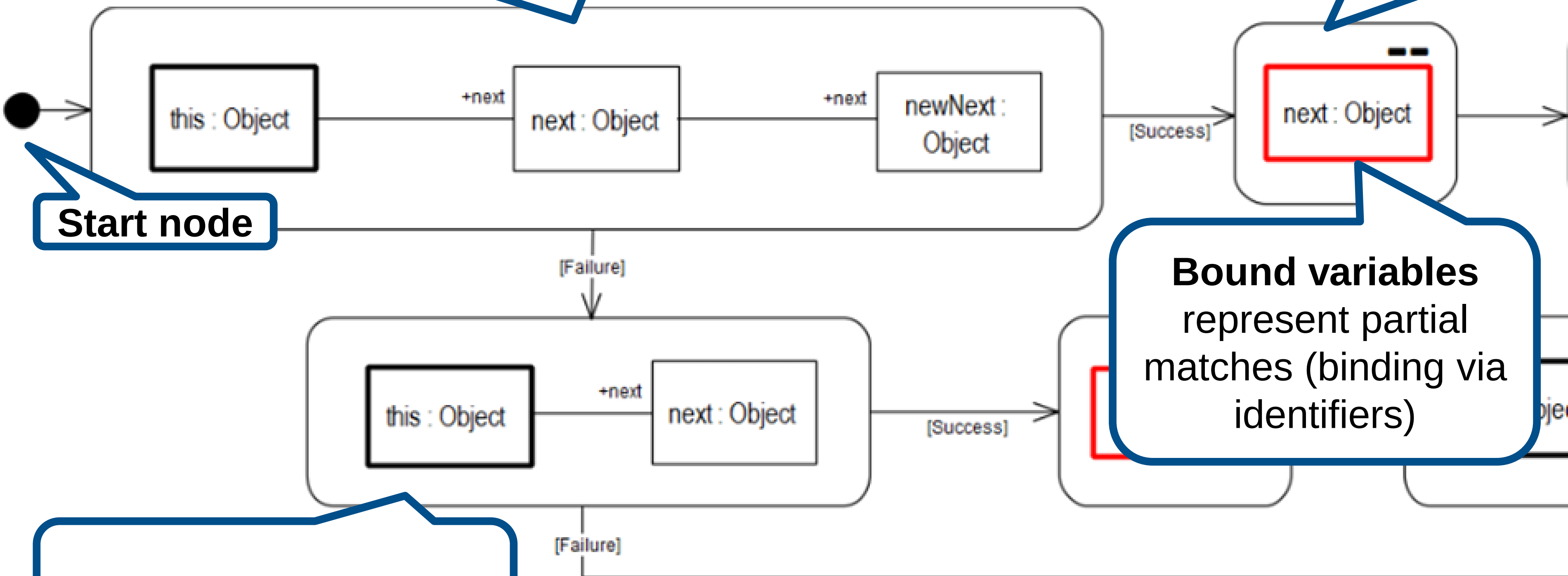


Uses simplified activity diagrams with typical imperative constructs (sequences, conditionals, ...)

What is Story-Driven Modelling (SDM)?

Story nodes might contain graph patterns which should be matched in the graph...

...but also (SPO) graph transformation rules to modify the graph

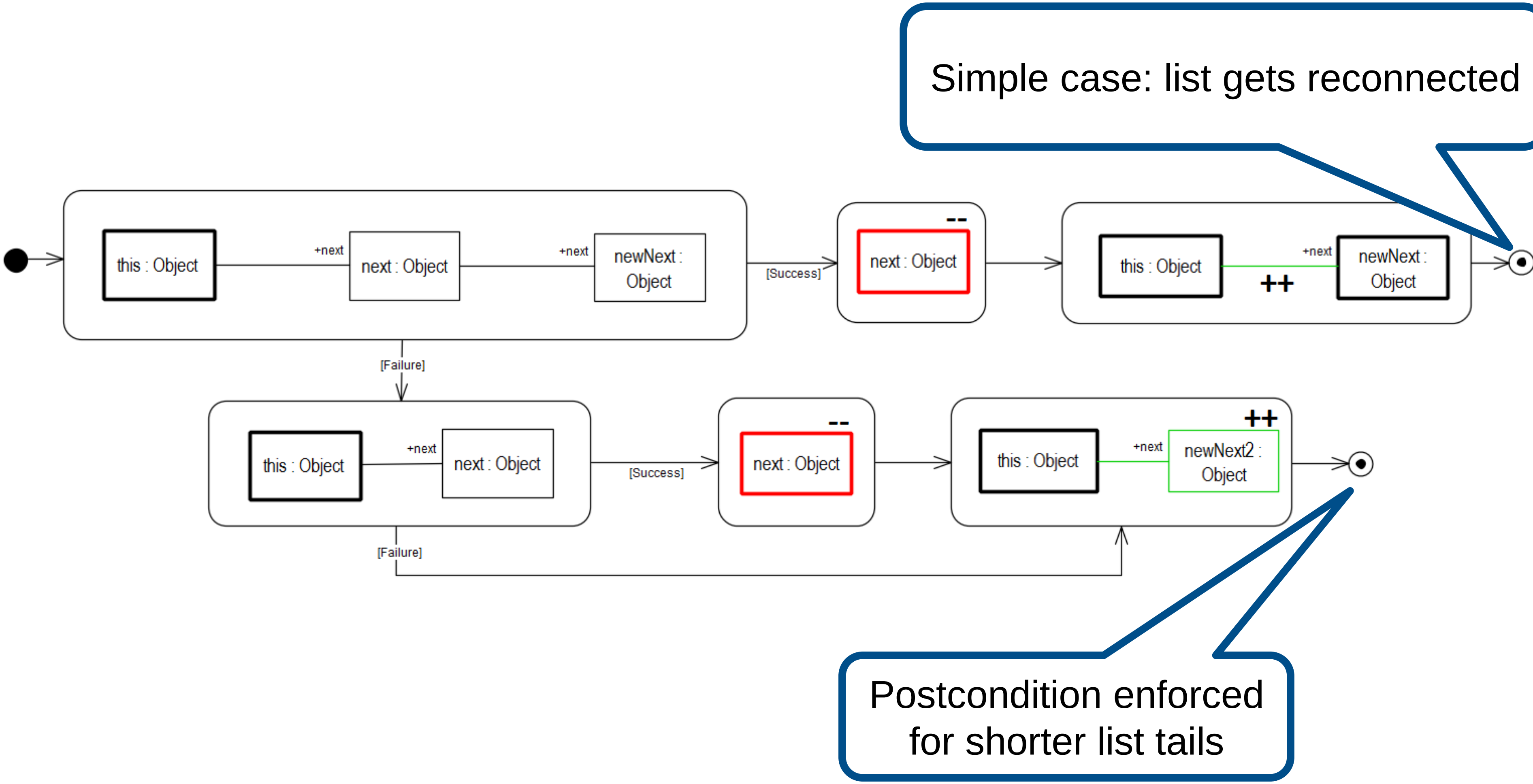


Start node

Bound variables represent partial matches (binding via identifiers)

Conditional branching

What is Story-Driven Modelling (SDM)?



Why bother with a formal semantics?

TECHNISCHE UNIVERSITÄT DARMSTADT

Denotational semantics (Zündorf, 2002):

- Defines the semantics in terms of valid input-output pairs of graphs
- Useful to, e.g., test an implementation for correctness
- It leaves crucial implementation details open ➡ insufficient to develop consistent tool support

Complementary step semantics:

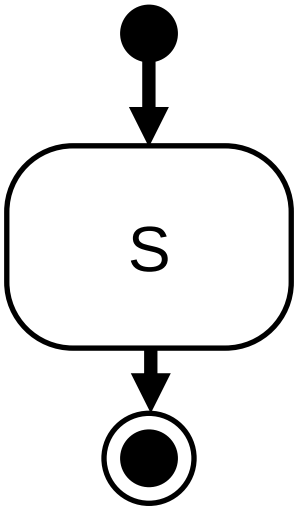
- Models directly the execution of an SDM specification
- Clarifies details of the execution behavior
- Supports SDM tool development and enriching SDM with new language constructs

Implementation based on denotational semantics: **CodeGen2** (Fujaba tool suite)

Implementation towards a unified semantics: **Democles** (eMoflon)

Denotational Semantics for SDM (Zündorf)

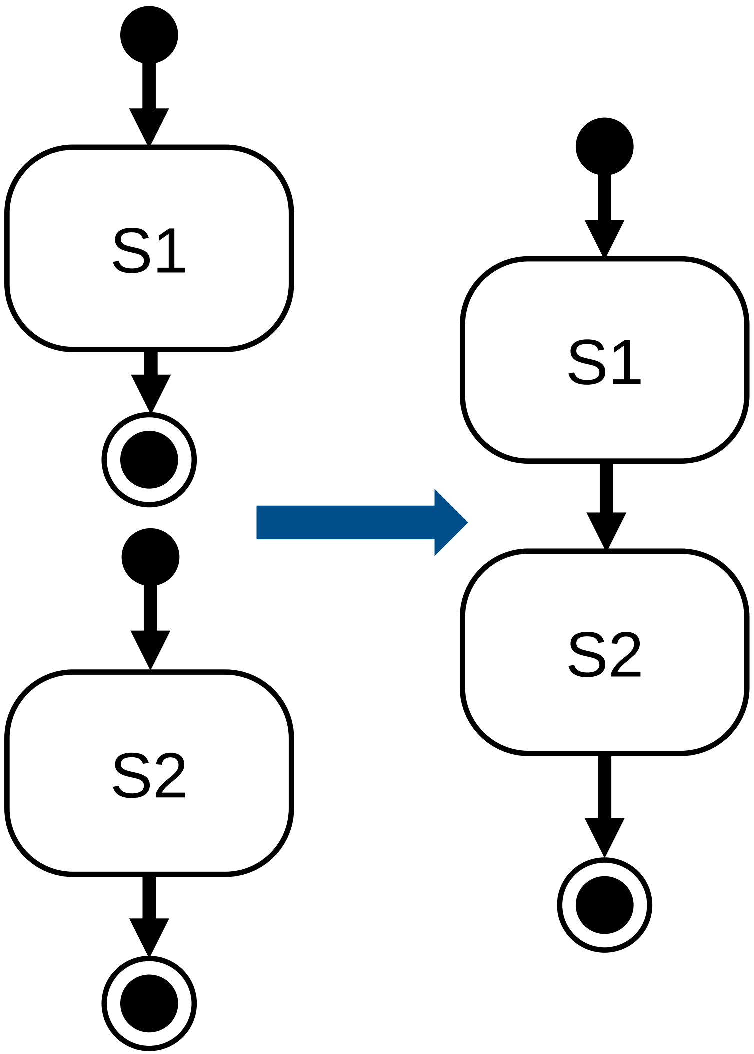
Defined for Fujaba (CodeGen2)



Semantics of a single story node is the set of all pairs of input and output graphs

$$Sem(S) \coloneqq \{(G_i, G_o) \mid G_i \overset{rule(S)}{\Longrightarrow} G_o\}$$

Denotational Semantics for SDM (Zündorf)

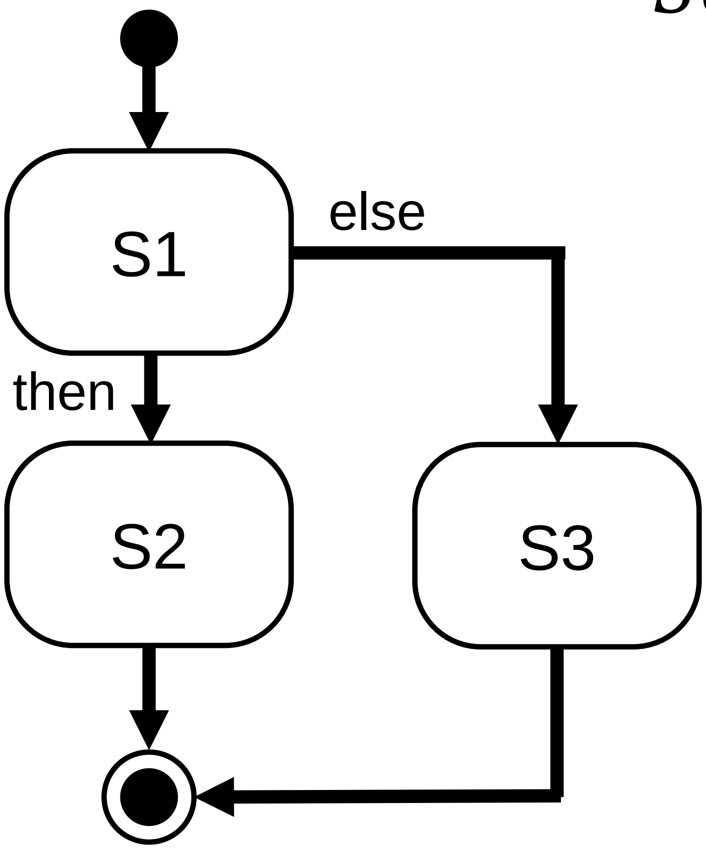


The semantics of a **sequence** of story diagrams is the set of all pairs consisting of the input graph of the first story diagram, and the corresponding output graph of the last story diagram

$$Sem(S1;S2) :=$$

$$\{(G_i, G_o) | G_i \overset{rule(S1)}{\Longrightarrow} G' \overset{rule(S2)}{\Longrightarrow} G_o\}$$

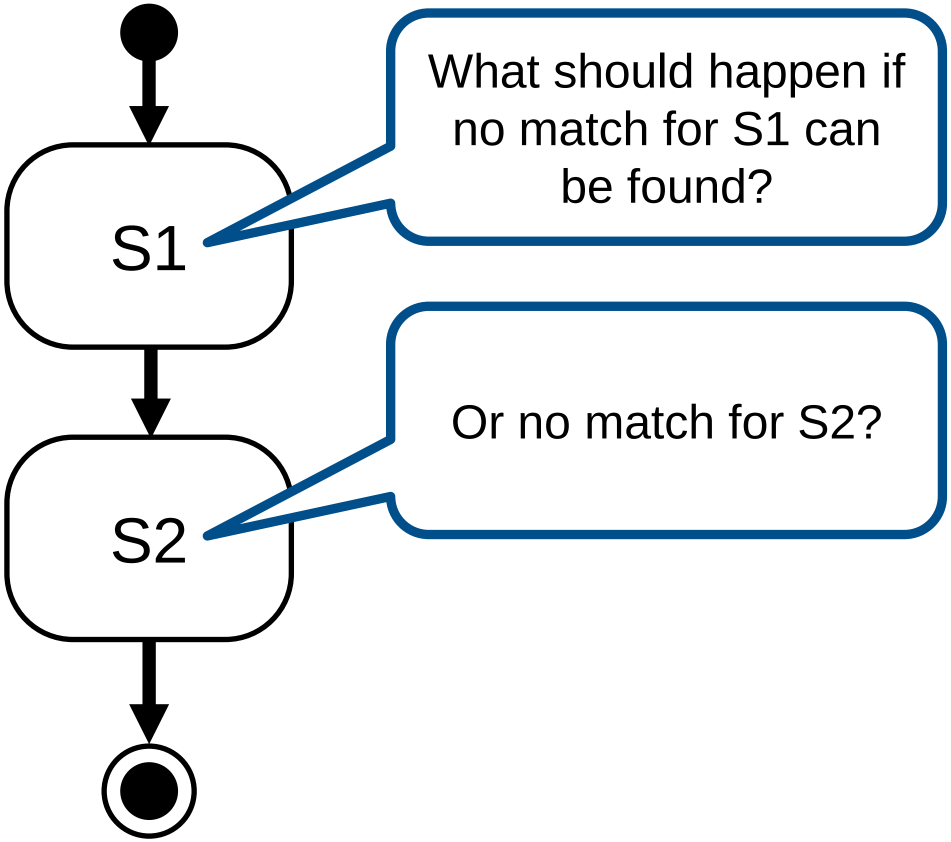
Denotational Semantics for SDM (Zündorf)



$$Sem(if\ S1\ then\ S2\ else\ S3) :=$$

$$\begin{array}{ll} Sem(S1;S2) & if\ \exists G_i \overset{rule(S1)}{\Longrightarrow} G' \\ Sem(S1;S3) & otherwise \end{array}$$

Unclear Situations: Termination



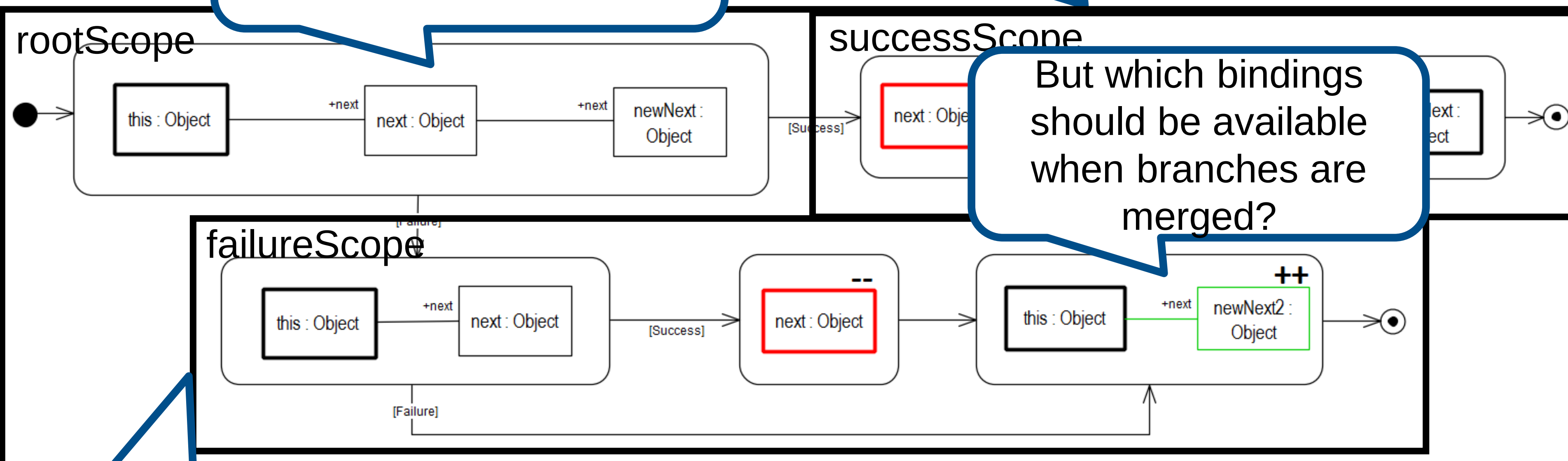
- The denotational semantics says nothing about how to “terminate”
- Practically, it requires backtracking or breadth-first search to discover every possible rule application path

- This is mostly too much effort and in practice, we expect that the execution terminates if a rule is not applicable
- The step semantics allows for formally describing this behavior

Unclear Situations: Bindings and Scopes

If a match is found, **next** and **newNext** are bound to a model element

All such bindings can be used in story nodes in the success branch



But which bindings should be available when branches are merged?

No bindings from the conditional node can be used in the failure branch

Conservative: Remove bindings deleted in **any** branch, no new bindings **(paper)**

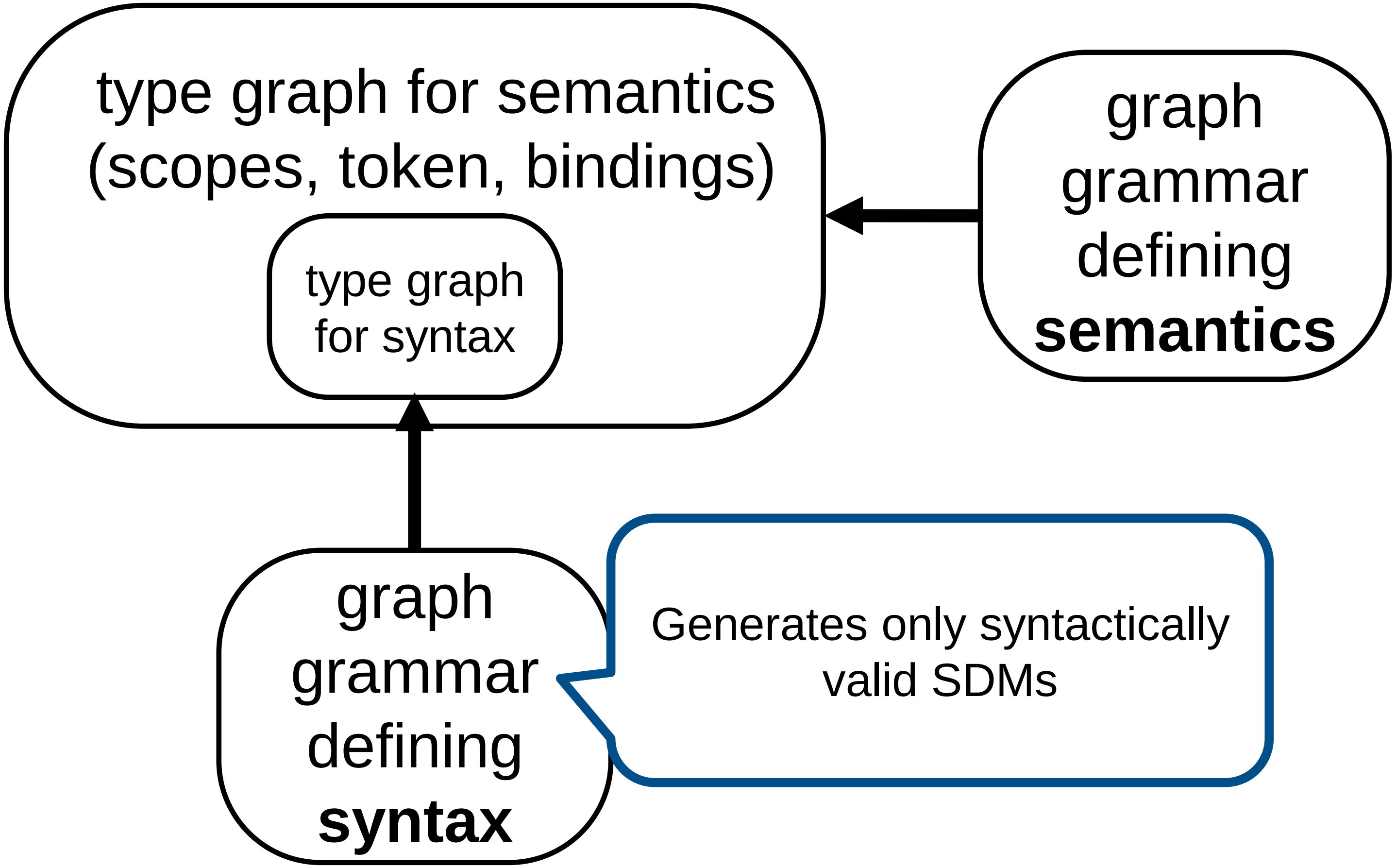
Optimistic: Remove bindings deleted in any branch, allow new bindings created in **both** branches

Contribution

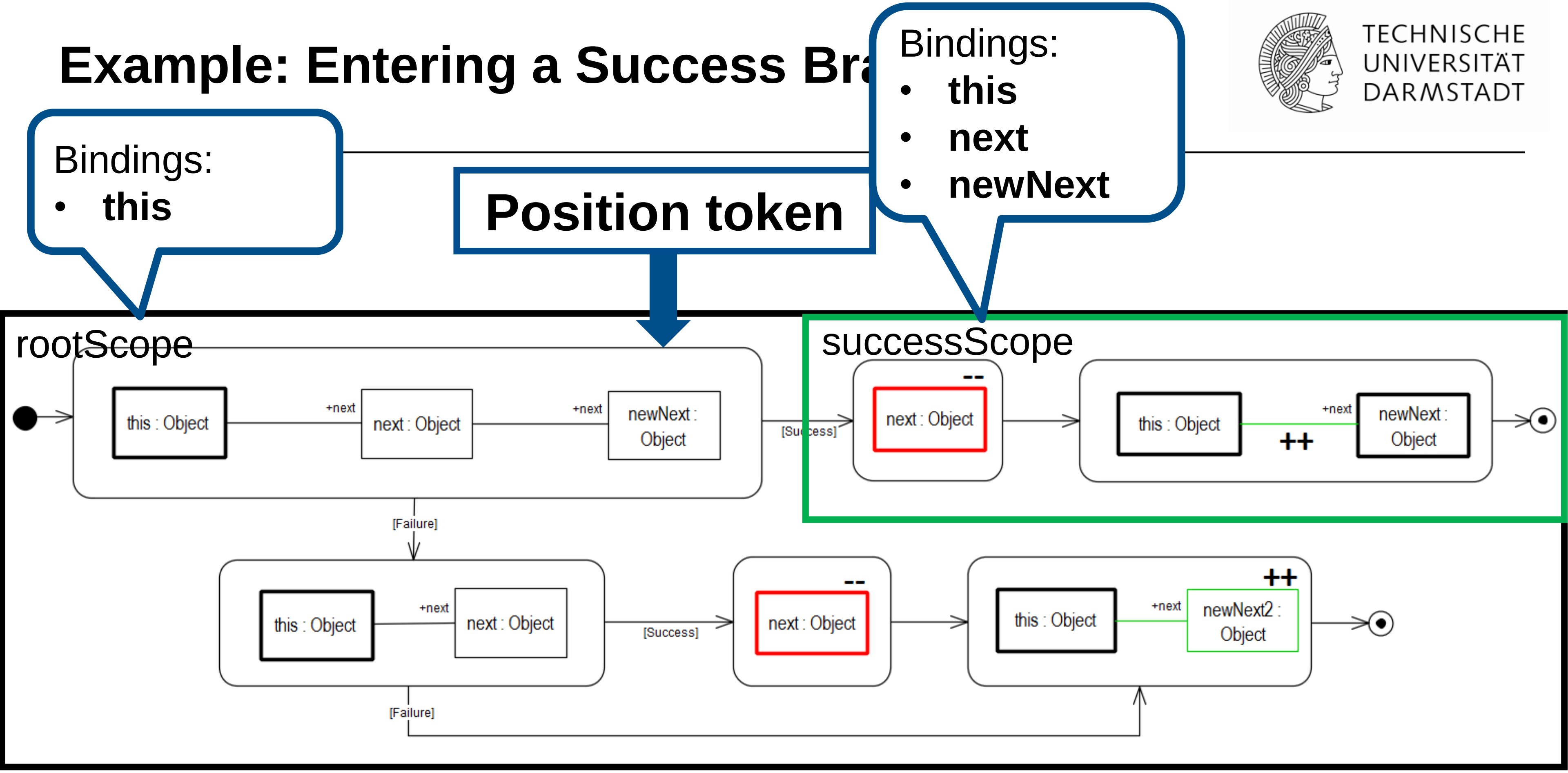
TECHNISCHE UNIVERSITÄT DARMSTADT

- In our paper, we suggest a complementary, operational semantics for SDM to fix such practical "low-level" design decisions
- These details might not be crucial for proving correctness, but greatly influence tool compatibility in practice
- Could be used to define compatibility levels for SDM tools

Structure of the Semantics

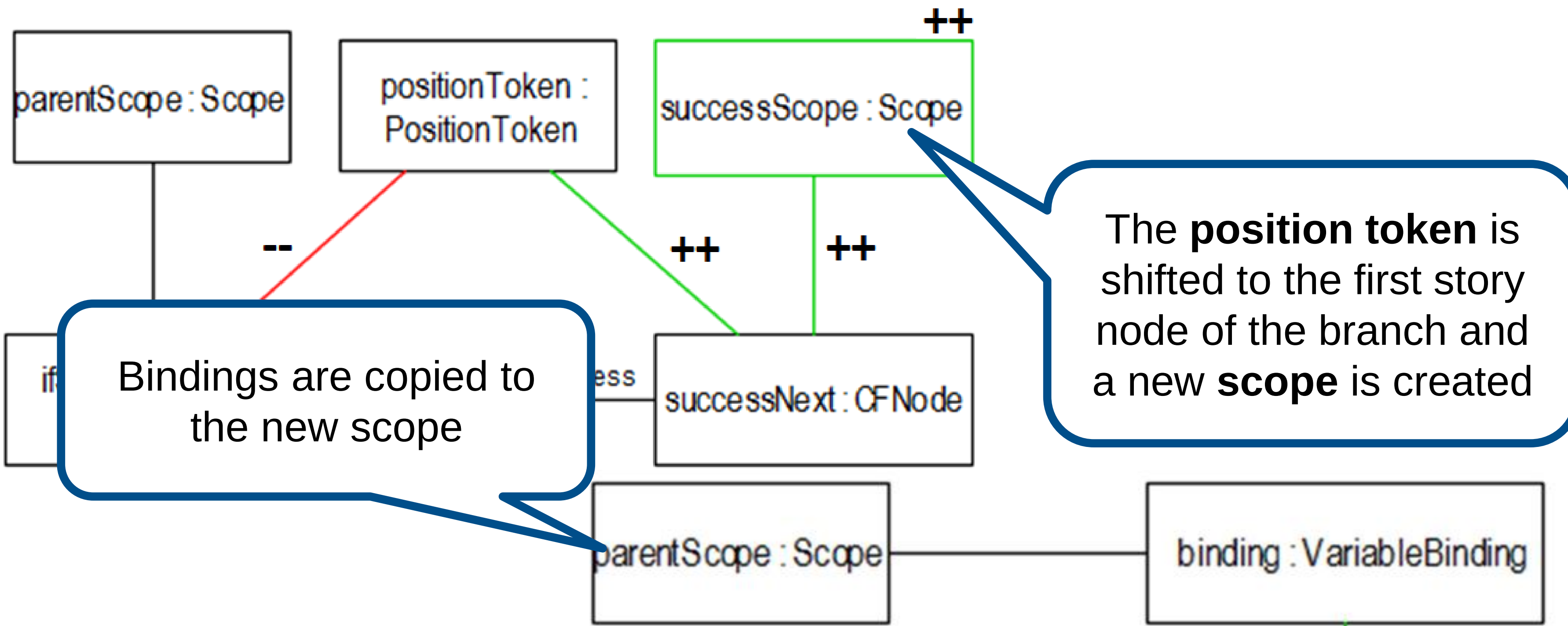


Example: Entering a Success Bra

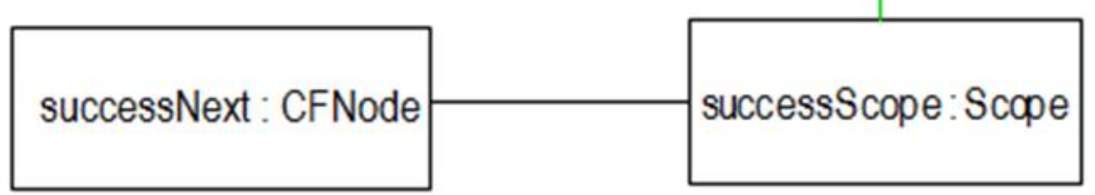


- Semantics is given in terms of **graph transformation rules for the semantic elements** (another abstraction level)
  - The semantic specification relies only on standard rule applications

Example: Entering a Success Branch



- Afterwards, bindings are updated according to the conditional story pattern



Conclusion and Future Work

- We proposed a step semantics to have a uniform definition of SDM executional behavior
- The semantics allows for detailed decisions left open by the previous denotational approach
- The semantics is based on a type graph which also allows for defining a syntax grammar which generates valid SDMs

- Future extensions to SDM in the works: we propose to extend *both* the denotational and operational semantics appropriately!
- Example: apply rule **for each** match
  - Recompute matches in each iteration? (CodeGen2)
  - Compute each match once and apply in „parallel"? (Democles)
  - Demand parallel independence?