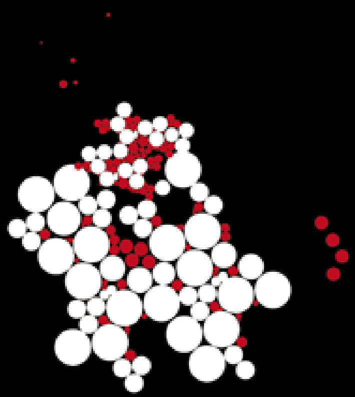
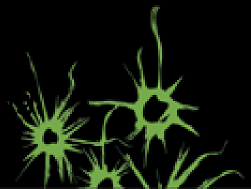
UNIVERSITY OF TWENTE.



GRAPH TRANSFORMATION USING GROOVE

Arend Rensink, University of Twente

November 2021



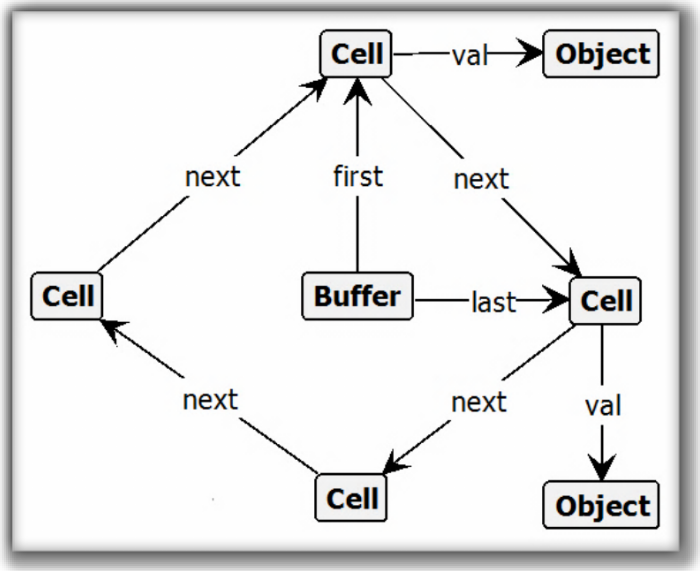
GRAPH TRANSFORMATION

- Formal language to capture dynamic system behaviour
  - Graphs will capture state snapshots
  - Transformation rules will capture program statements
- Aim (here): Behavioural analysis
  - Qualitative behaviour captured by graph production system
  - Requirements captured by logic properties expressed as graphs
- Why graph transformation?
  - Very powerful, widely applicable paradigm
  - Graphs are natural for many domains
  - Makes for (very) rapid prototyping

Note: There is GT life beyond behavioural analysis

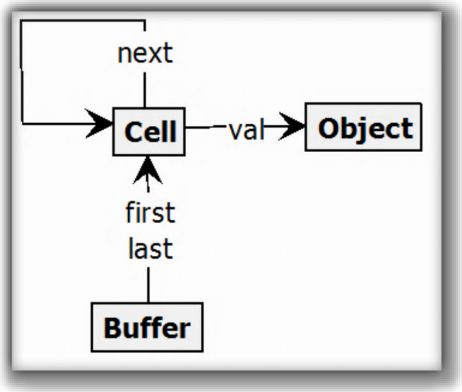
- Graph Grammars for reasoning about (non-textual) languages
- Graph Transformation for Model Transformation

GRAPHS AS STATE MODELS



- Example state graph
  - Nodes represents objects
  - Edges represent fields or relations between objects
- Here: Circular buffer
  - Objects inserted at the tail (last element)
  - Objects removed from the head (first element)

TYPE GRAPHS AS METAMODELS



- Example type graph
  - Compare with (UML) class diagrams
- Nodes stand for object *types*
  - Also supported: Node inheritance
- Edges stand for field/relation types
  - Not shown here: multiplicities

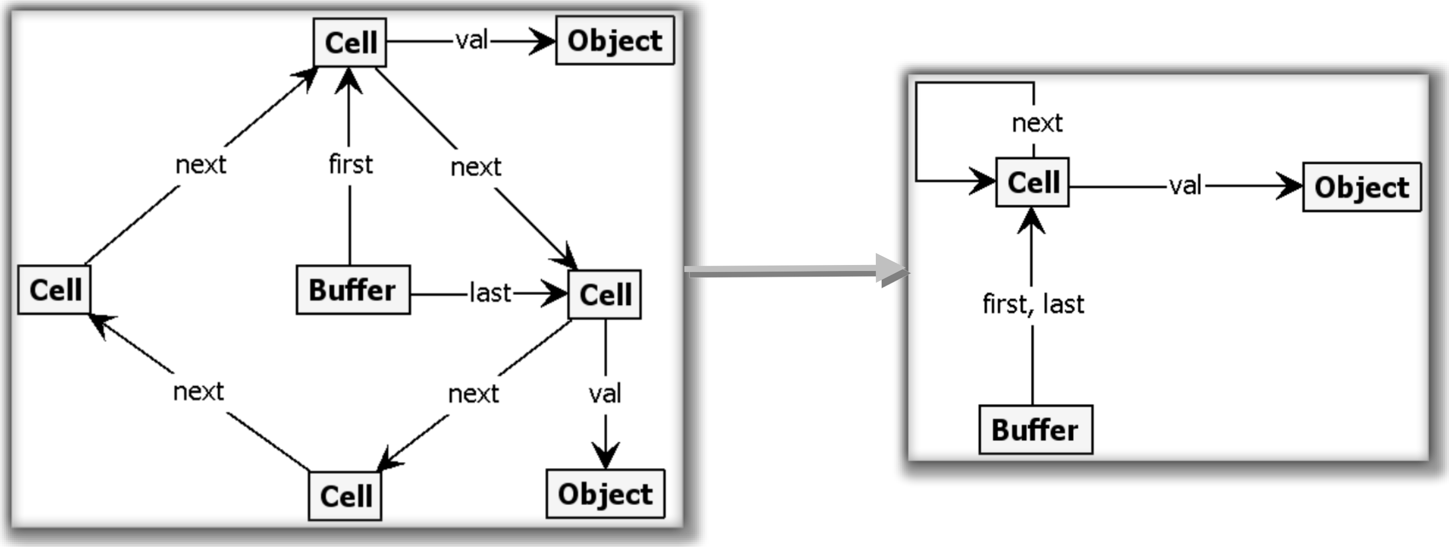
GRAPH FORMALISM

- Graphs in this presentation (*simple graphs*):
  - Flat (i.e., not hierarchical)
  - Directed, edge-labelled, no parallel edges
  - Self-edges depicted as node labels
- Formally: with
  - Global set of labels
    - Fixed subsets of type labels and flags (node labels)
  - finite set of nodes
  - finite set of labelled edges
- Partial morphisms
  - Structure-preserving node mappings (need not be injective)
  - Isomorphism: bijective (total) morphism
    - Used to abstract from node identities

EXAMPLE MORPHISM

State graph

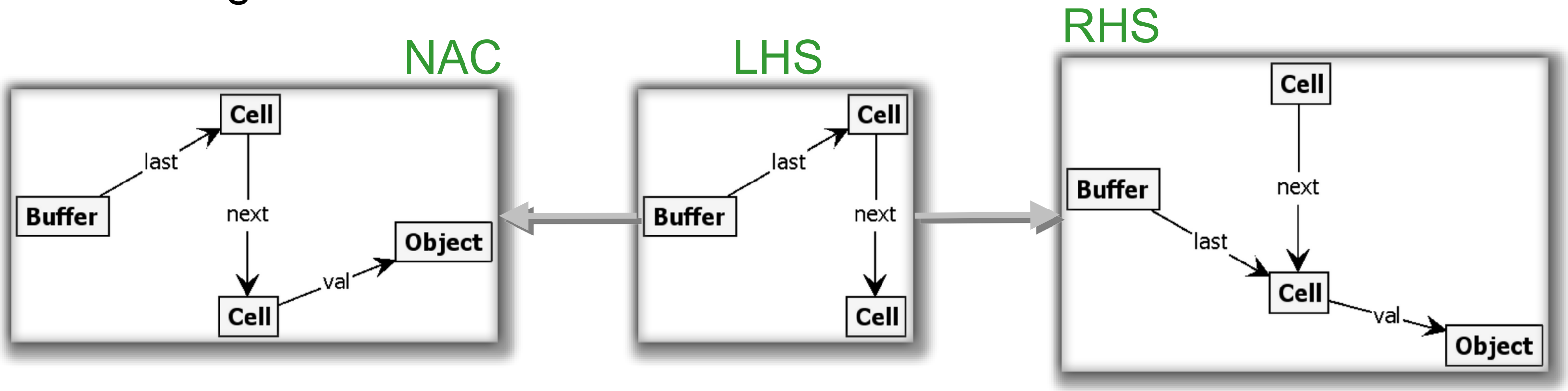
Type graph



- Typing is a (weak) structuring mechanism
  - Limits node and edge labels and their interconnection
  - Does not enforce presence or absence of edges

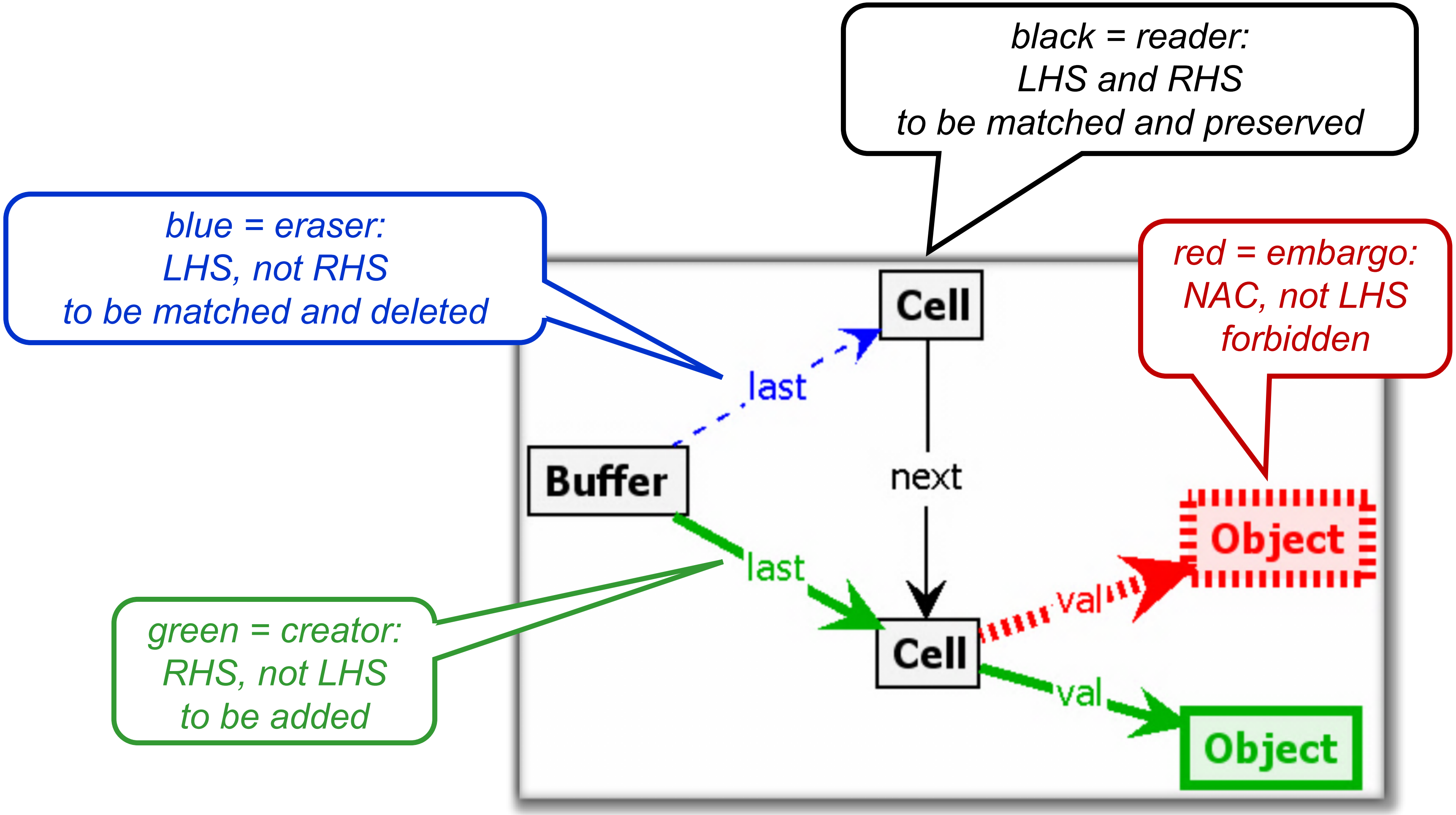
GRAPH REWRITE RULES

Putting an element into a circular buffer:



- Graph Production System: Set of rewrite rules
- A rewrite rule embodies a particular *change* to a graph
  - Left Hand Side (LHS): to be matched in the *host* (source) graph
  - Difference of Right Hand Side (RHS) and LHS defines change
  - Negative Application Condition (NAC): should not occur in host graph (there can be any number of these)
- Compare to string rewriting/hyperedge replacement
  - Graph rewrite rules are context sensitive

SINGLE-GRAPH REPRESENTATION



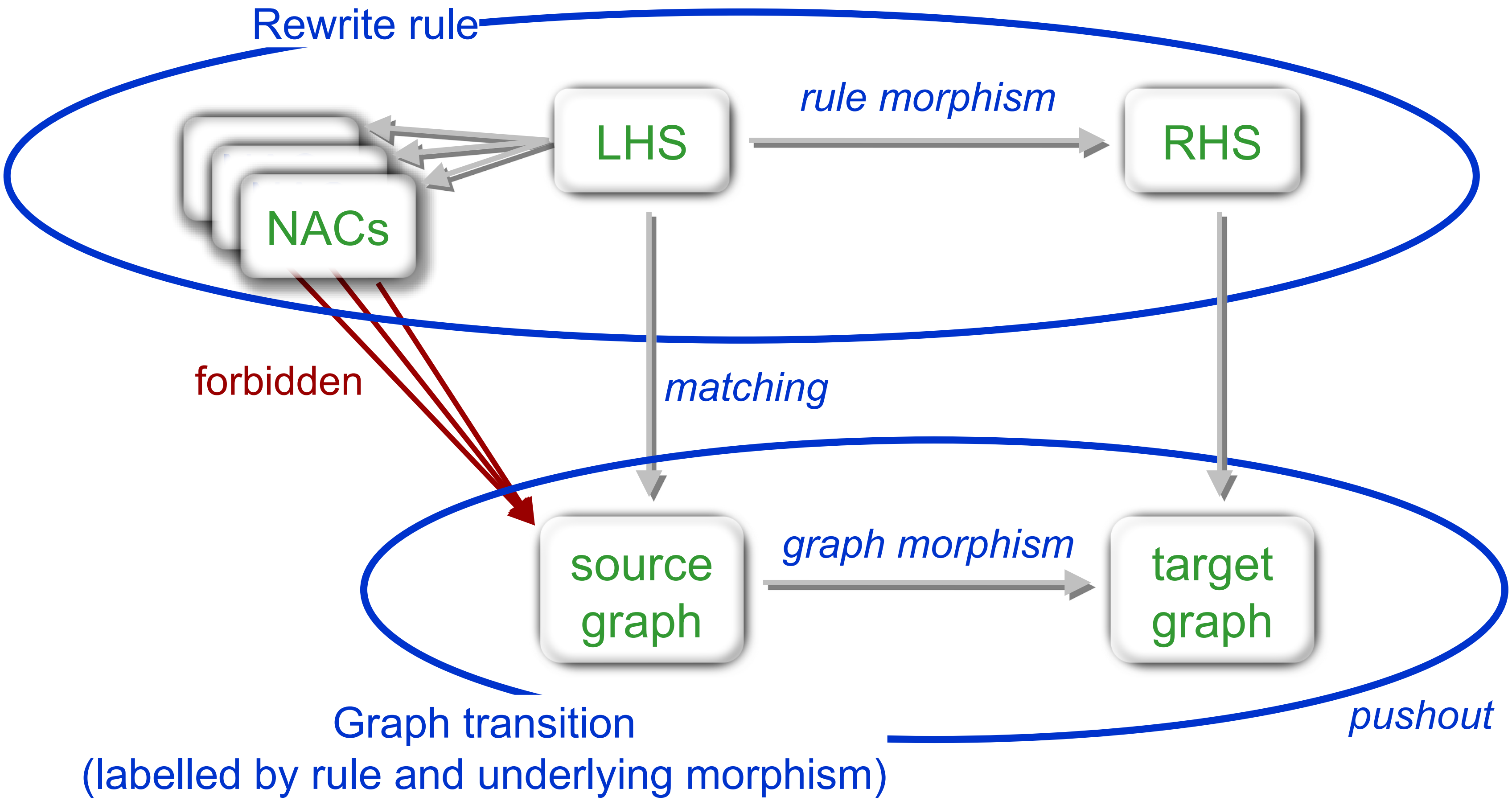
black = reader:
LHS and RHS
to be matched and preserved

blue = eraser:
LHS, not RHS
to be matched and deleted

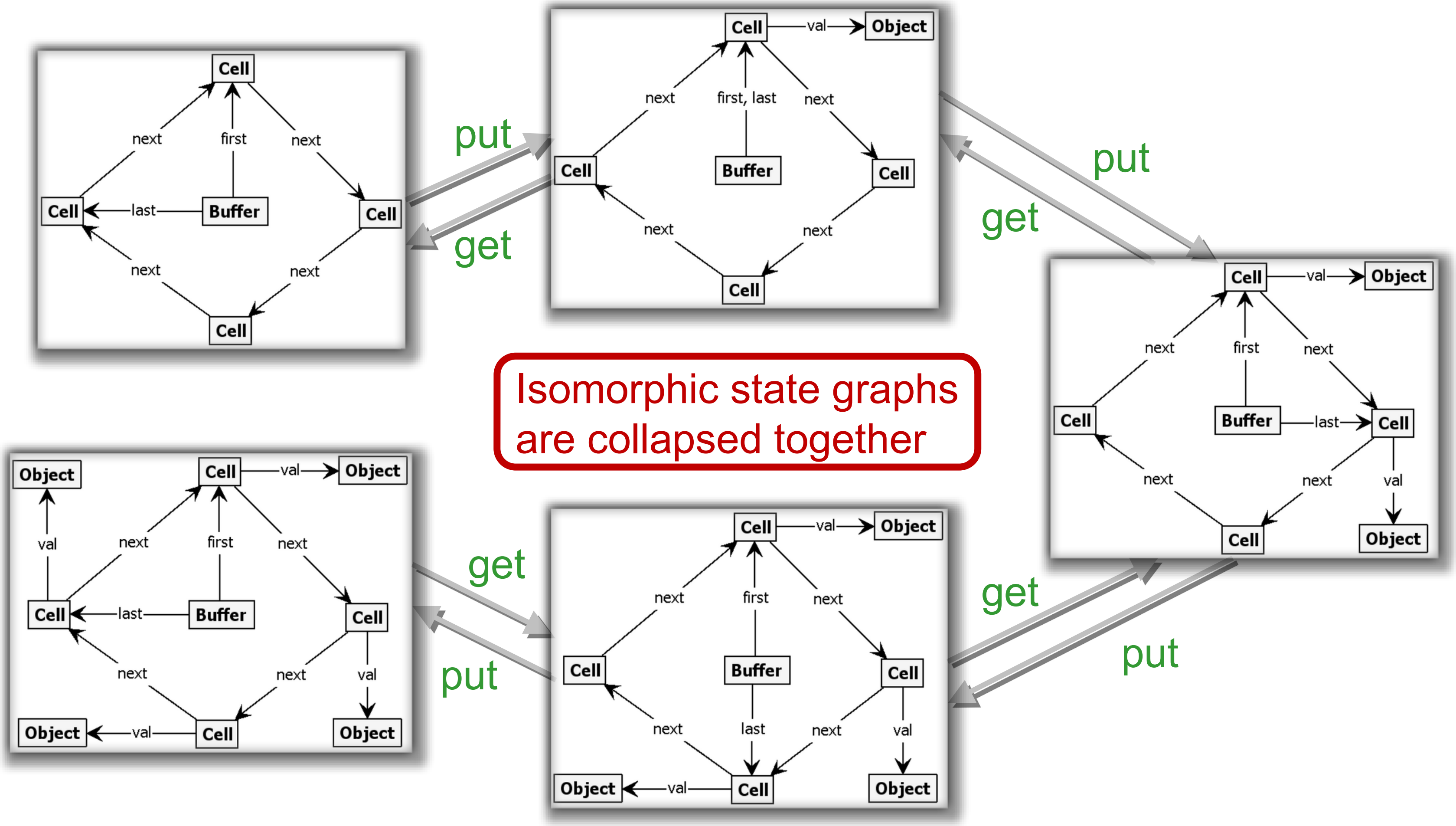
red = embargo:
NAC, not LHS
forbidden

green = creator:
RHS, not LHS
to be added

GRAPH PRODUCTIONS



GRAPH TRANSITION SYSTEMS



Isomorphic state graphs are collapsed together

AIM: MODEL CHECKING

- Construct graph production system
  - Directly from problem description, *or*
  - From UML diagrams / other specification language, *or*
  - From programs to be checked
- Generate state space
  - States = graphs
  - Transitions = transformations
- Formulate properties
  - State invariants and forbidden patterns (safety)
  - Liveness (absence of deadlock)
  - Full temporal logic (LTL/CTL)
- Check properties on the model

WOLF, GOAT & CABBAGE

Propositiones ad Acuendos Juvenes (n.C.)
("Problems to sharpen the young")