

Table 1. Structural constraints for UML activity diagrams with their inclusion status in LADEX and justification

	Constraint	Inclusion	Reason
1	An activity diagram can have zero or more initial nodes	SC1	Activity diagrams may use more than one initial node when they are hierarchical; otherwise, they have one or none. Our formalization, as detailed in Section 2, does not support hierarchies. In addition, state-of-the-art software modelling practices require every behavioural model to have a starting point so that the model has a clear semantics. Therefore, we require exactly one initial node.
2	An activity diagram can have zero or more end nodes	SC2	State-of-the-art software modelling practices require every behavioural model to include a termination point. Thus, we require at least one end node to terminate the process.
3	An initial node must have no incoming edges	SC3	Included without modification.
4	An end node must have no outgoing edges	SC4	Included without modification.
5	A decision node must have at least two outgoing edges, each labelled by a guard condition	SC5	Included without modification.
6	There should be at least one path from the initial node to every other node in the activity diagram	SC6	Included without modification.
7	An action node can have multiple incoming edges	✗	We implicitly support this constraint, as we do not constrain action nodes' incoming edges.
8	An action node can have multiple outgoing edges	✗	We implicitly support this constraint, as we do not constrain action nodes' outgoing edges.
9	A merge or join node must have multiple incoming edges	✗	We abstract merge and join nodes as action nodes. This simplification eliminates the need for enforcing specific merge and join node constraints.
10	A merge or join node must have exactly one outgoing edge	✗	We abstract merge and join nodes as action nodes with a single outgoing flow. This simplification eliminates the need for enforcing specific fork and join constraints.
11	A fork node must have exactly one incoming edge	✗	We abstract fork nodes as action nodes. Thus, we do not explicitly enforce this constraint.
12	A fork node must have multiple outgoing edges	✗	We implicitly support this constraint by abstracting fork nodes as action nodes, which allows any number of outgoing edges.
13	An object node can have multiple incoming and outgoing edges	✗	Since we represent object nodes as action nodes, and action nodes can have multiple incoming and outgoing edges, this constraint is already satisfied.
14	Every fork node that creates parallel flows should have a corresponding join node	✗	As we abstract fork and join nodes to action nodes, this constraint does not need explicit enforcement.
15	Object flows must connect compatible object nodes and actions	✗	We assume data object compatibility but do not formally constrain it, as we do not explicitly model object nodes.
16	Swimlanes (partitions) must not alter semantics	✗	Our formalization, as detailed in Section 2, excludes swimlanes.
17	A node should not have an outgoing transition unless it connects to a target node	✗	Our model generation and encoding process inherently enforces this constraint, as it disallows transitions without both a source and a target node.