Synthesising Provably Correct Programs for Autonomous Systems

Brian Logan
Utrecht University

Motivation

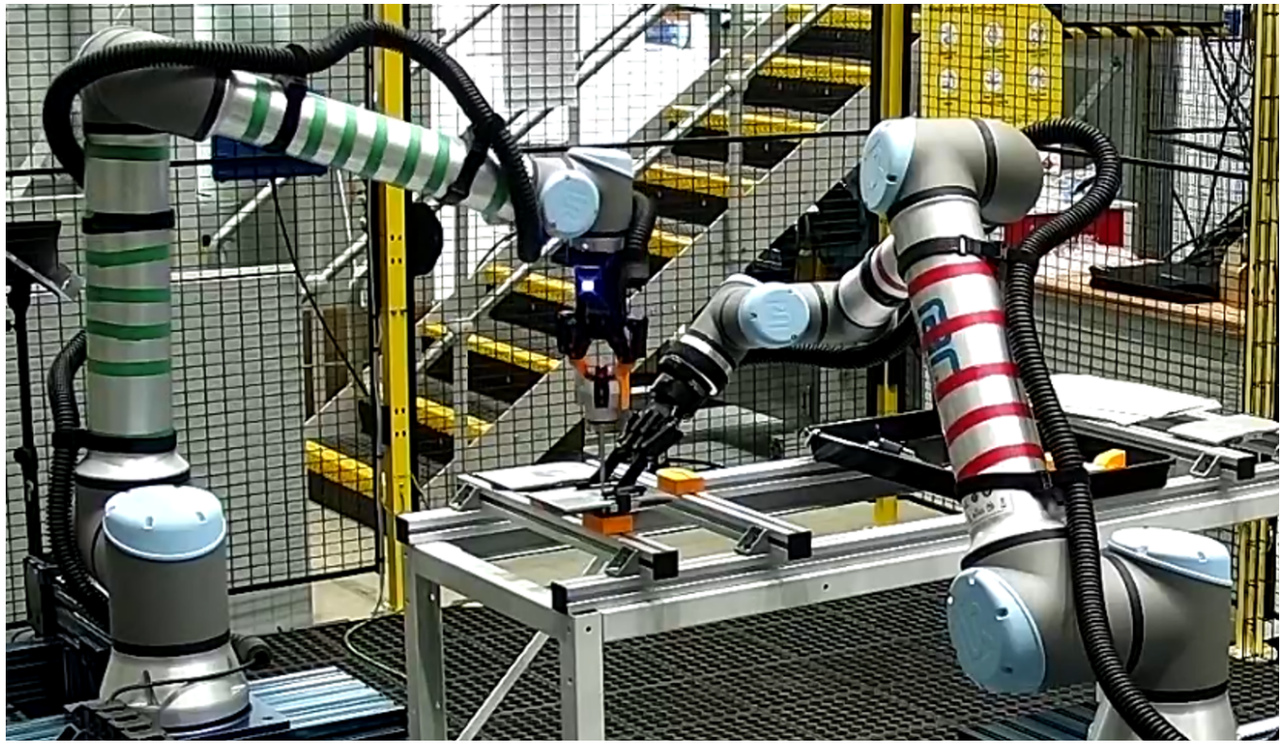
- writing software for autonomous systems is hard
- for many applications, the code must be verified, and verification may reveal bugs
- wouldn’t it be great if we could somehow generate the software for autonomous systems directly from their specification?

Overview

- given a description of an autonomous system and a high-level specification of the tasks to be performed
- automatically synthesise a controller for the system to perform the task
- controller is (provably) ‘correct by construction’

Applications

autonomous manufacturing



autonomous inspection & maintenance

Approach

- **resources:** (non-deterministic) robots, AGVs, etc.
- **topology:** the relationships between the resources
- **process:** (high-level) specification of the tasks to be performed by the system
- synthesise a **controller** that delegates each task in the process to a resource in the topology

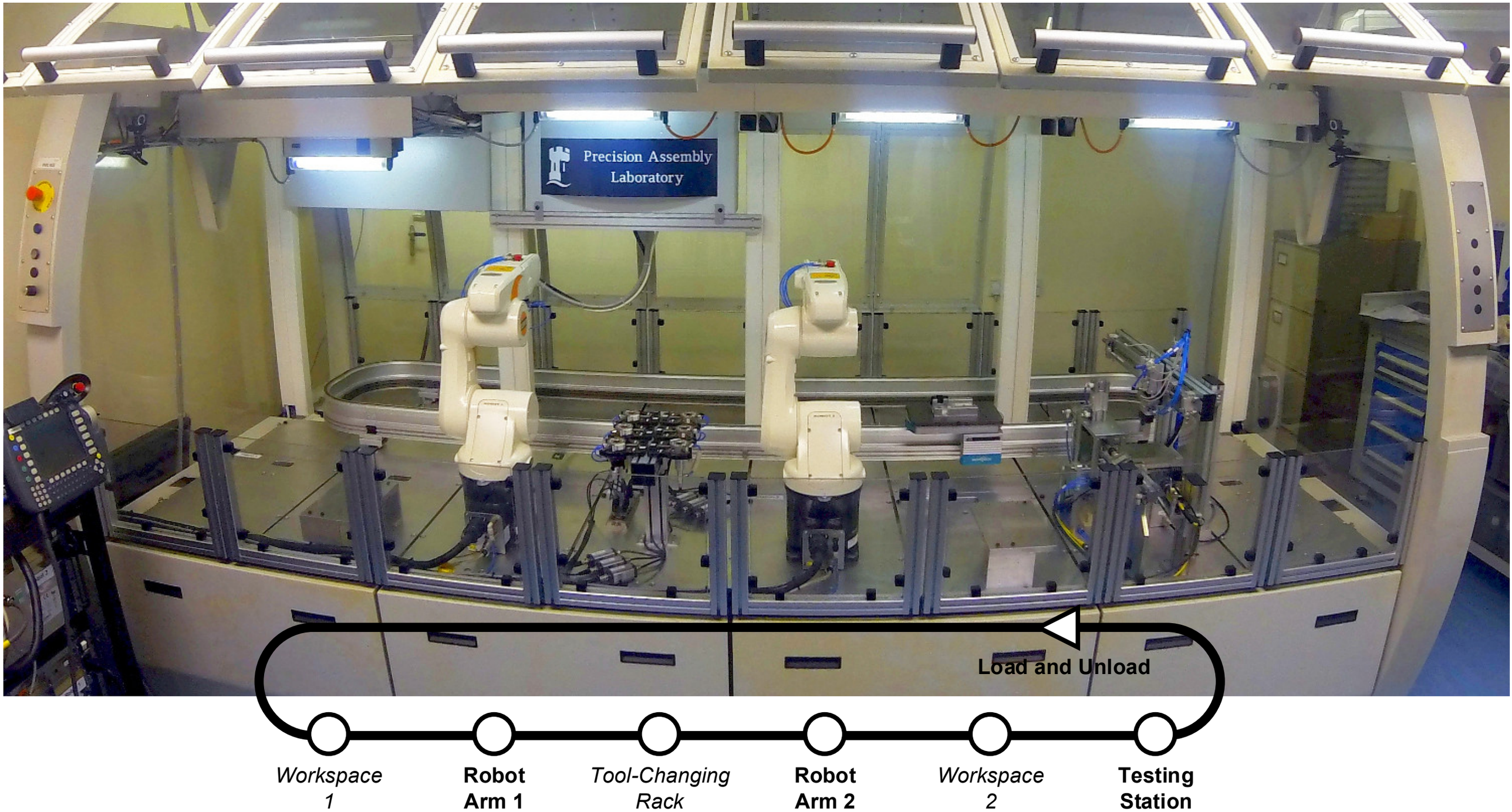
Manufacturing resources

- **production resources** perform operations on parts, e.g., machine tools, robots
  - several resources may perform an operation in parallel on the same parts
  - resources may not do the right thing first time (i.e., non-deterministic)
- **transport resources** route parts between production resources, e.g., conveyer belts, AGVs

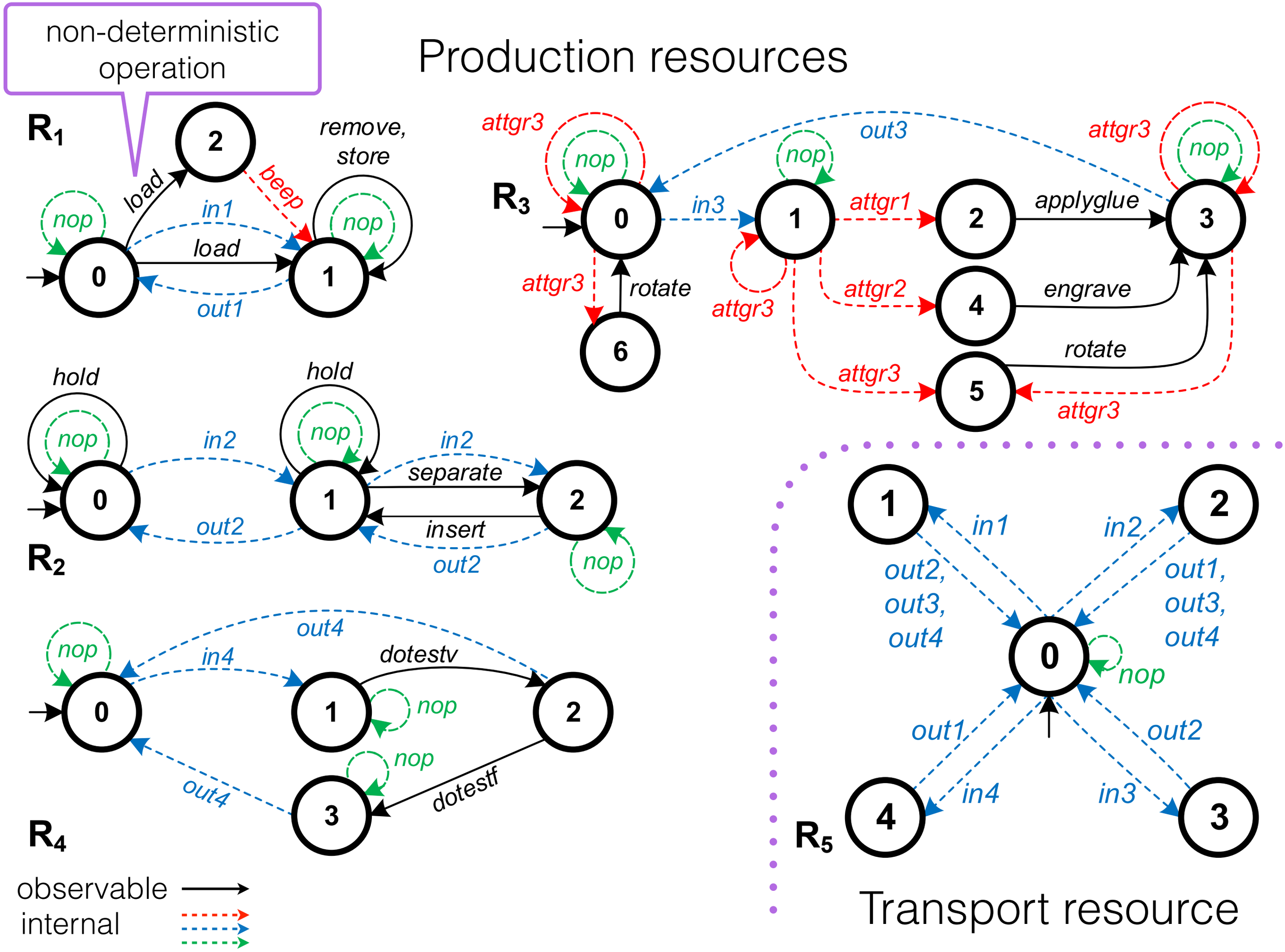
Resources modelled as LTSs

- **nodes** represent the state of a resource
- **edges** represent (possibly non-deterministic) resource operations:
  - *observable* manufacturing operations
  - *internal* operations necessary for the resource to function, e.g., attaching a gripper to a robot arm
  - *in* and *out* internal operations indicate the transfer of parts; *nop* indicates an ‘idle’ operation

Precision assembly demonstrator



Production resources



Transport resource

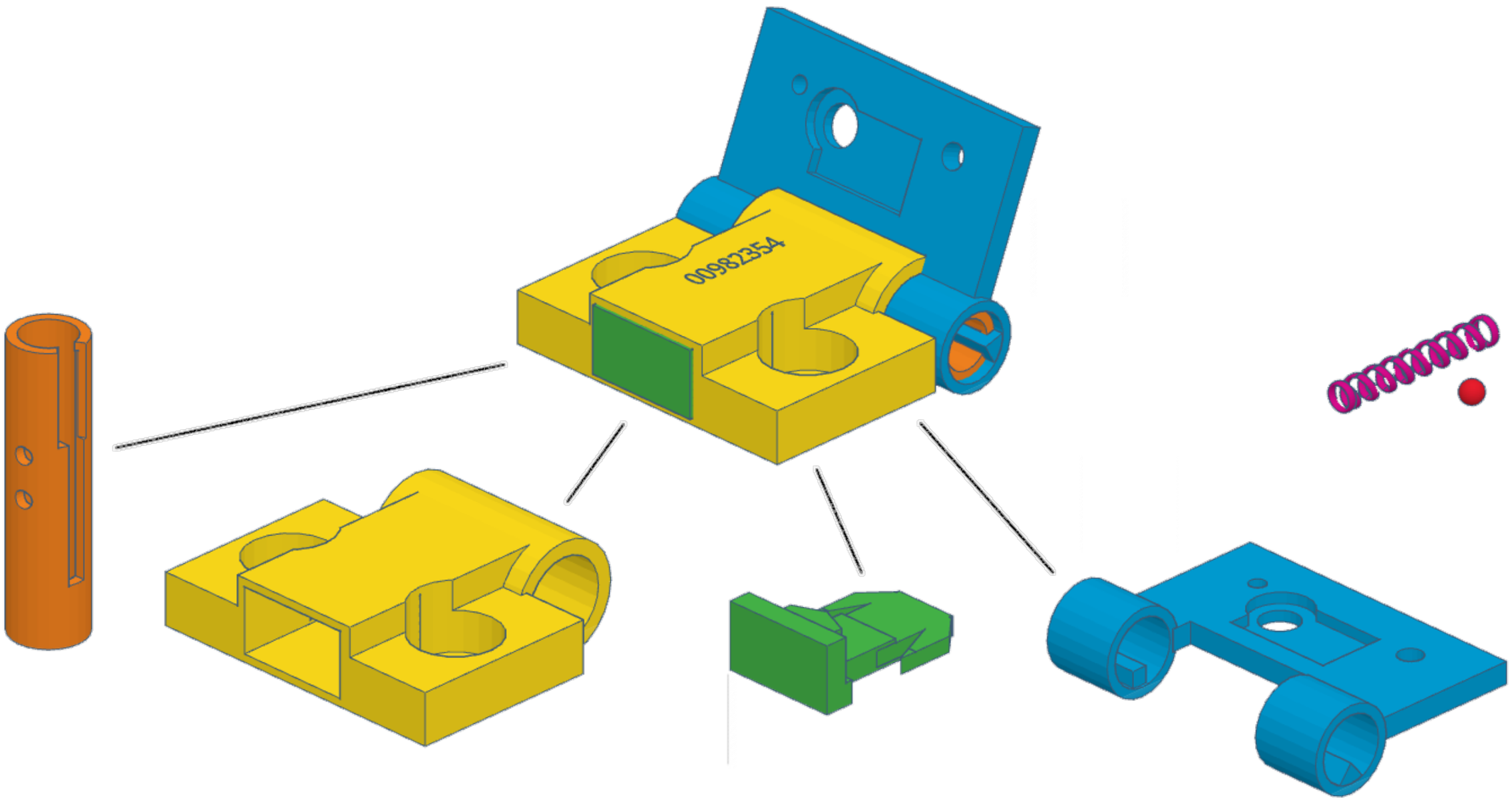
Production topology LTS

- represents the layout of the manufacturing system, i.e., how the resources are interconnected
  - **nodes** represent the state of the system, i.e., state of each resource and which parts are allocated to it
  - **edges** represent a set of concurrent operations by each resource (joint actions on parts, transfer of parts)
- intuitively, the cross product of the resource LTSs with transitions where the *in* and *out* edges don't match removed

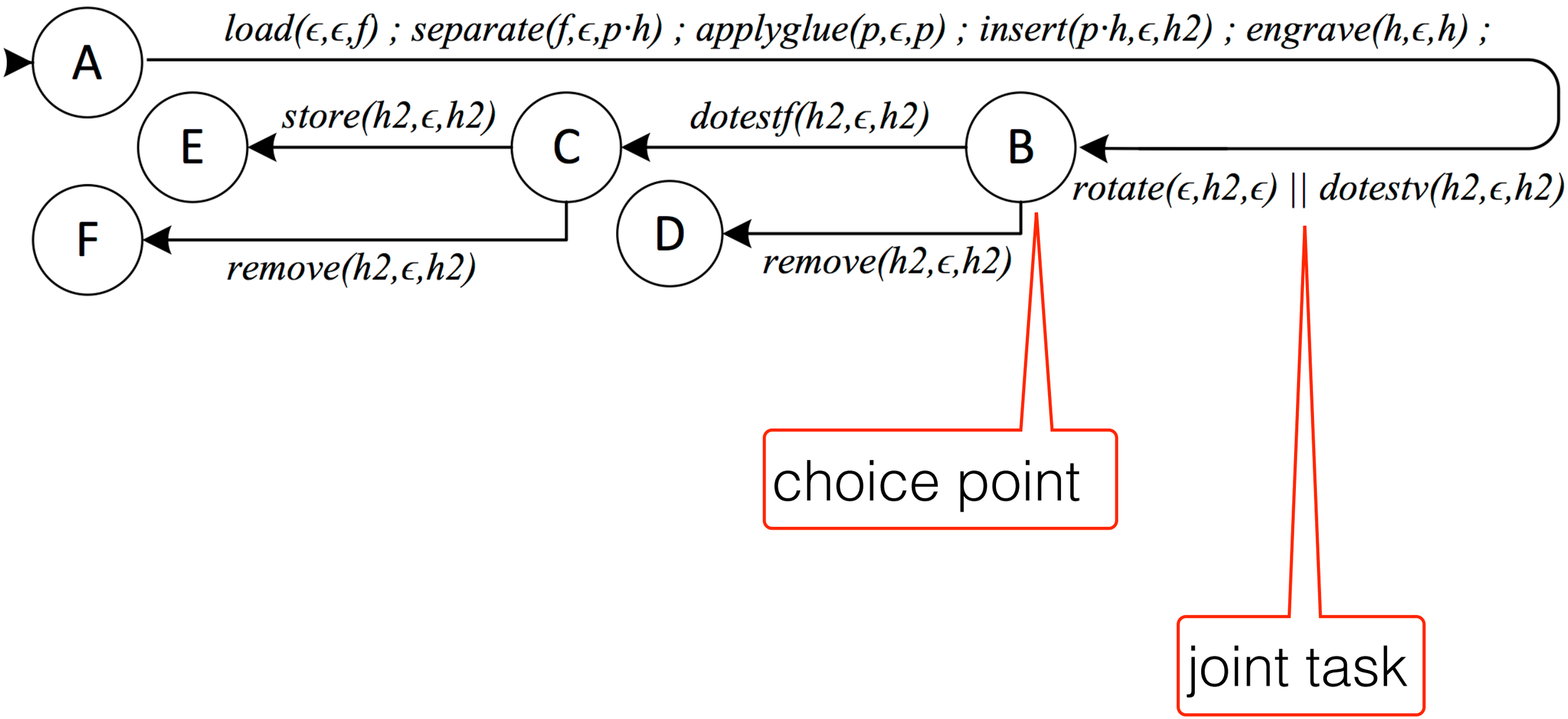
Process recipe LTS

- specifies the *tasks* necessary to manufacture a product type
- **nodes** are *choice points* – which tasks to perform is determined by the specification of the current product instance or a test
- **edges** are *task expressions* consisting of *primitive tasks* executed in sequence ';', in parallel, '||', or interleaved, '|'
- **primitive (joint) tasks** $t(x,y,z)$ specify the operation to be performed, t, the input parts, x, the parts assigned to another resource, y, and the output parts, z

Example: detent hinge



Process recipe for a hinge



Process plan controller

- maps *primitive tasks* in the recipe to *observable* operations by a resource
- a resource R_i in state s_i can execute $t(x,y,z)$ if
  - t is possible in s_i,
  - the input parts x are currently assigned to R_i, and
  - there exists resources R_j to which the parts in y are currently assigned

Non-determinism

- primitive tasks in the recipe are assumed to be *deterministic*, but resources are *non-deterministic*
- standard notions of *fairness* used in behaviour synthesis are *not applicable* in assembly:
  - repeating an *observable* operation violates the recipe
  - only *internal* operations ‘implementing’ an observable operation may be retried
- controller may have to implement a ‘recovery’ internal plan fragment to remedy undesired states

Manufacturability & control

- a recipe can be manufactured by a topology if transitions in the recipe can be matched (step-by-step) by transitions in the topology
- uses a novel simulation relation that involves allocating tasks and parts in the recipe to resources in the topology
- controller is computed as a by-product of the manufacturability check

Questions?