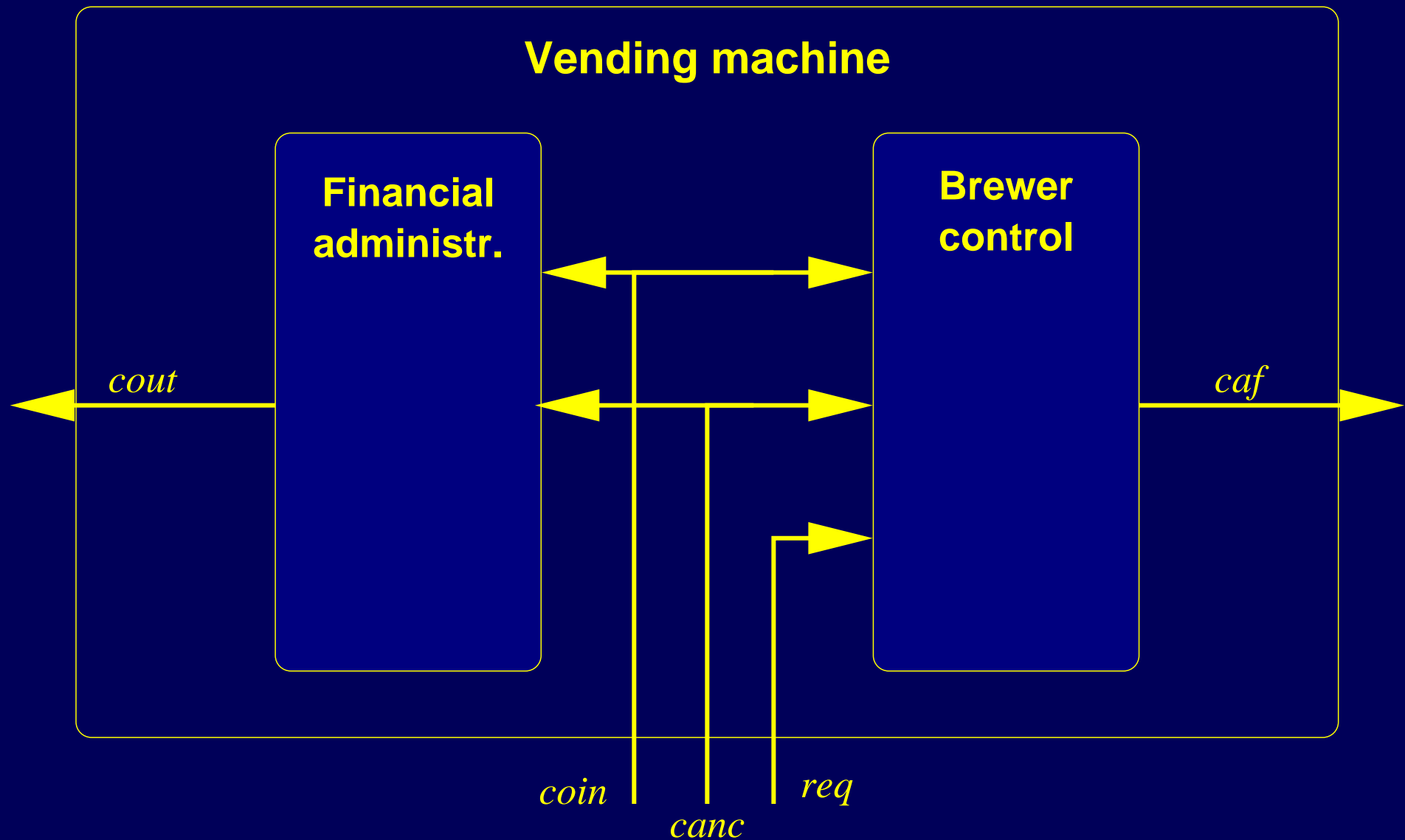

High-Level Modelling with Statecharts

Martin Fränzle

Informatics and Mathematical Modelling
The Technical University of Denmark

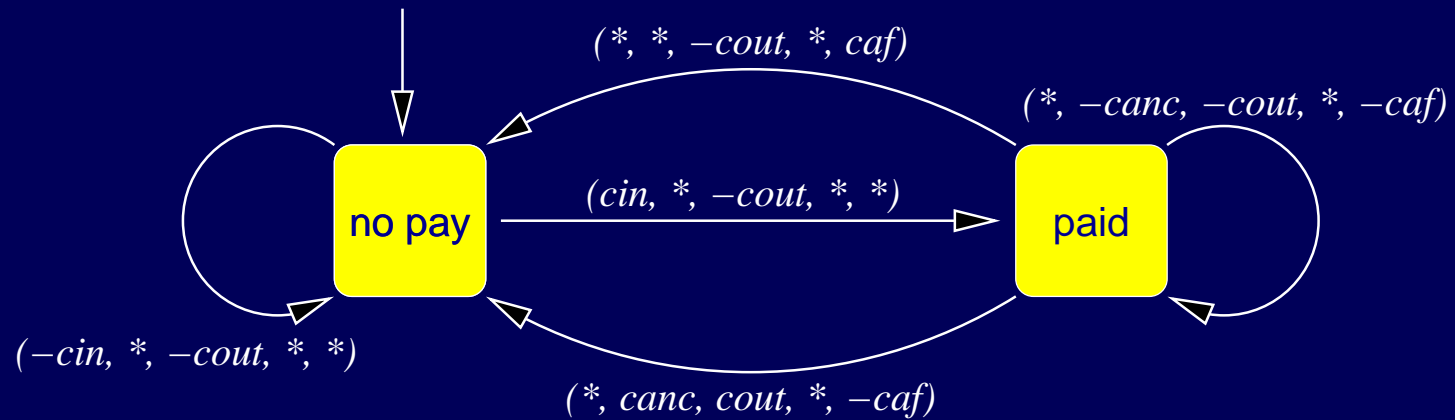
Simple automata models

The coffee vending machine — architecture

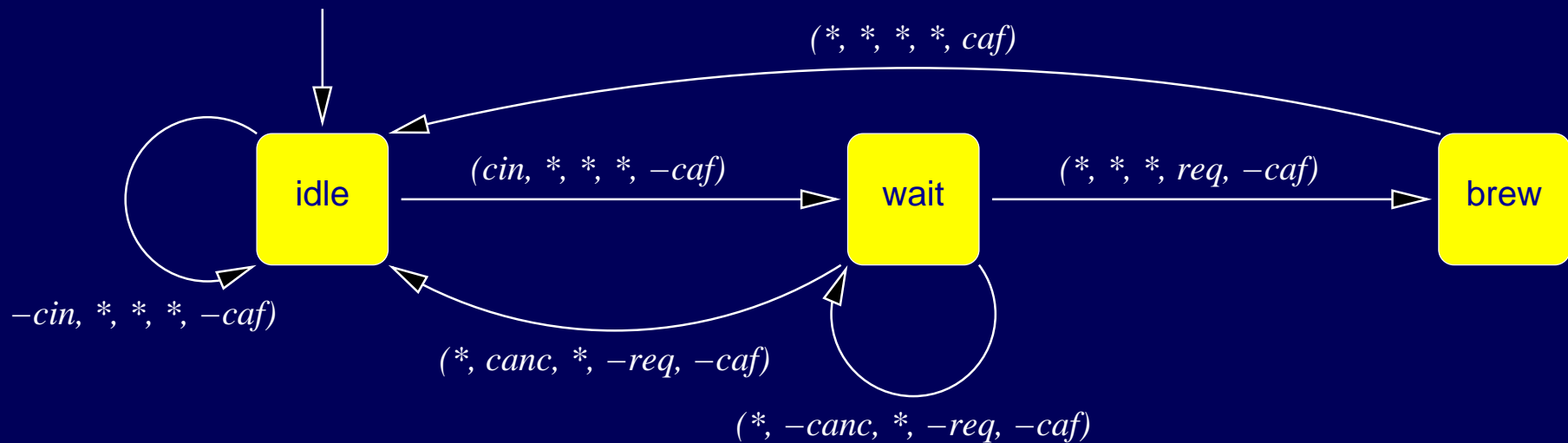


The coffee vending machine — dynamics

Financial administration

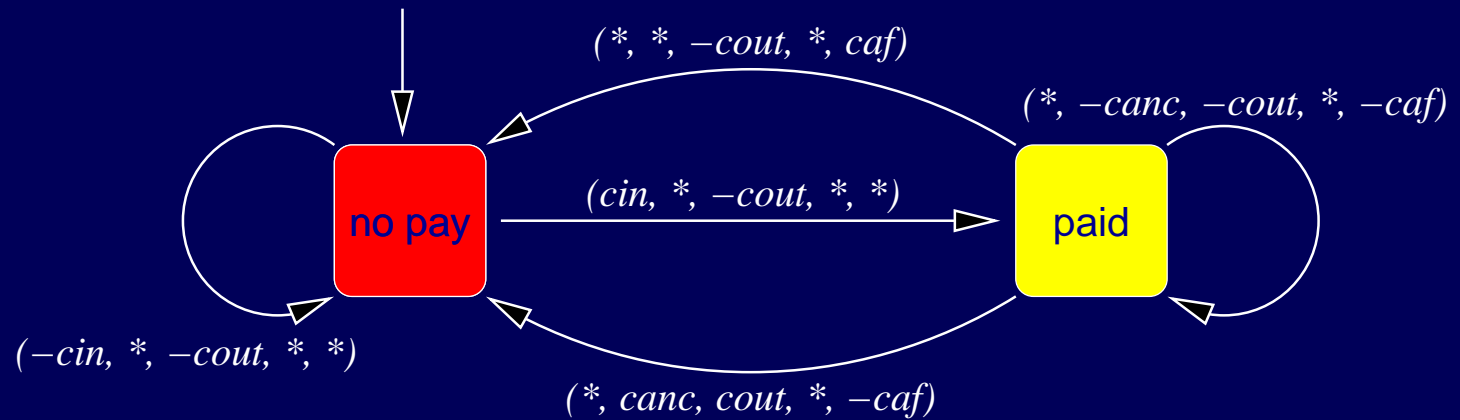


Brewer control

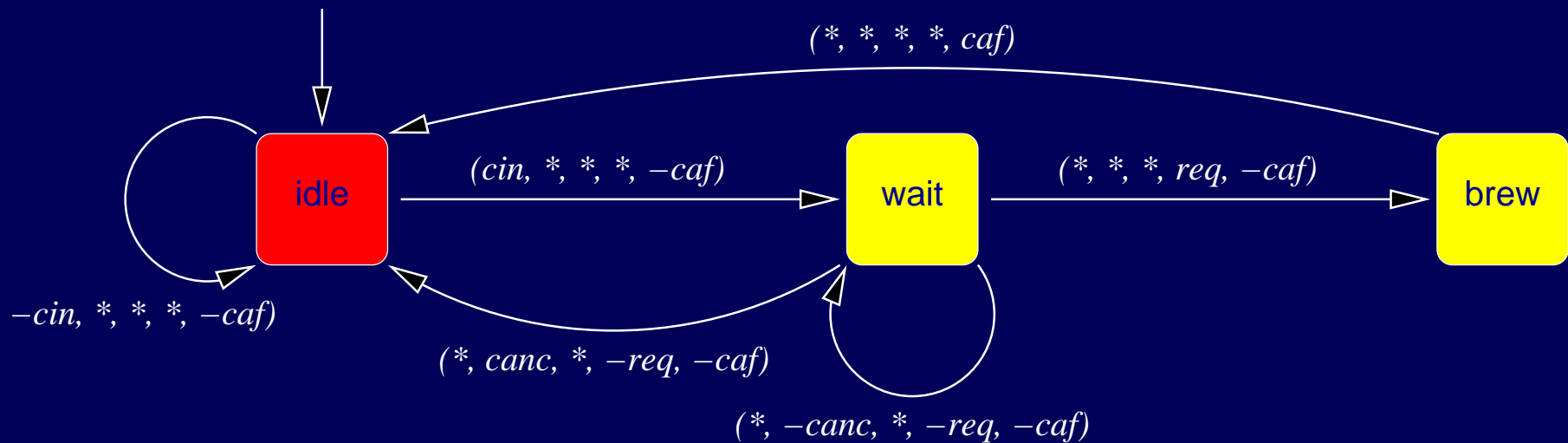


An example run

Financial administration

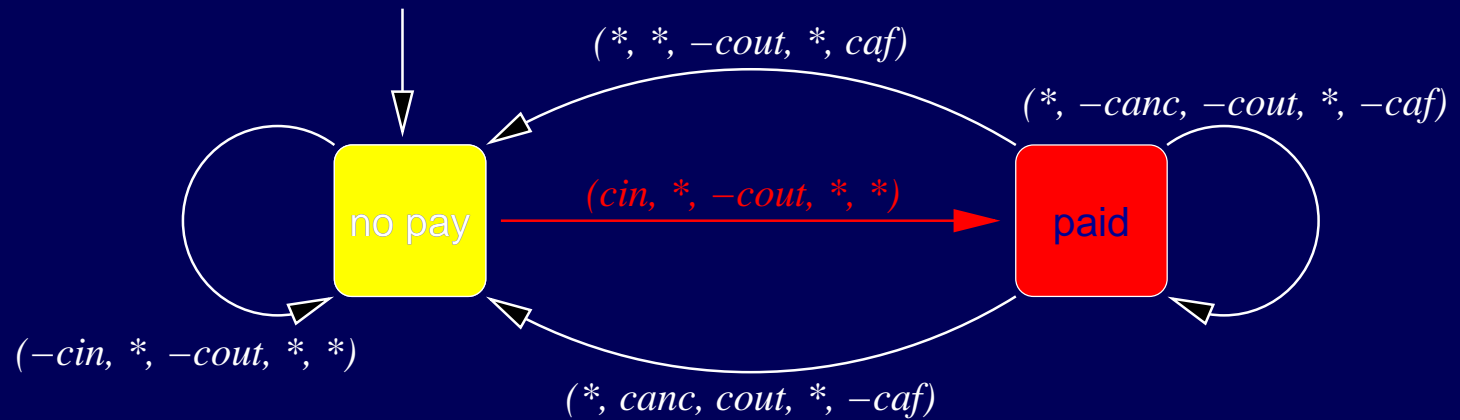


Brewer control

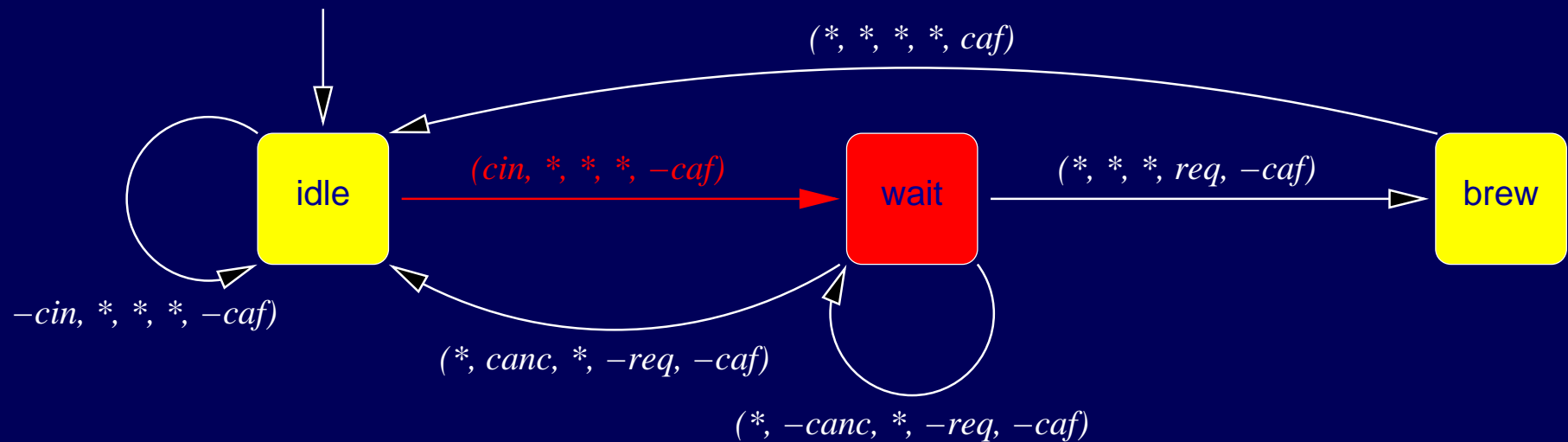


An example run

Financial administration

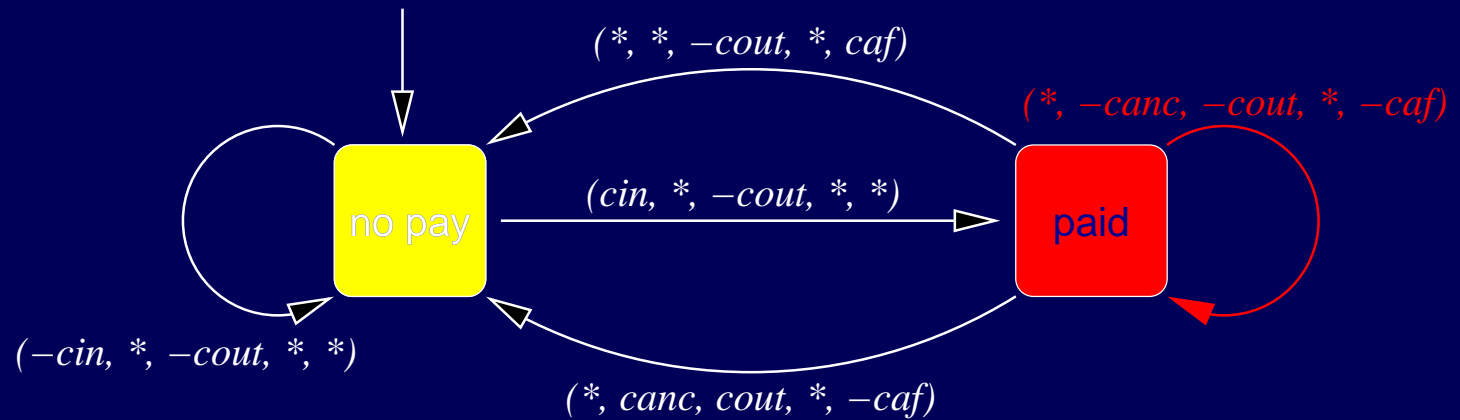


Brewer control

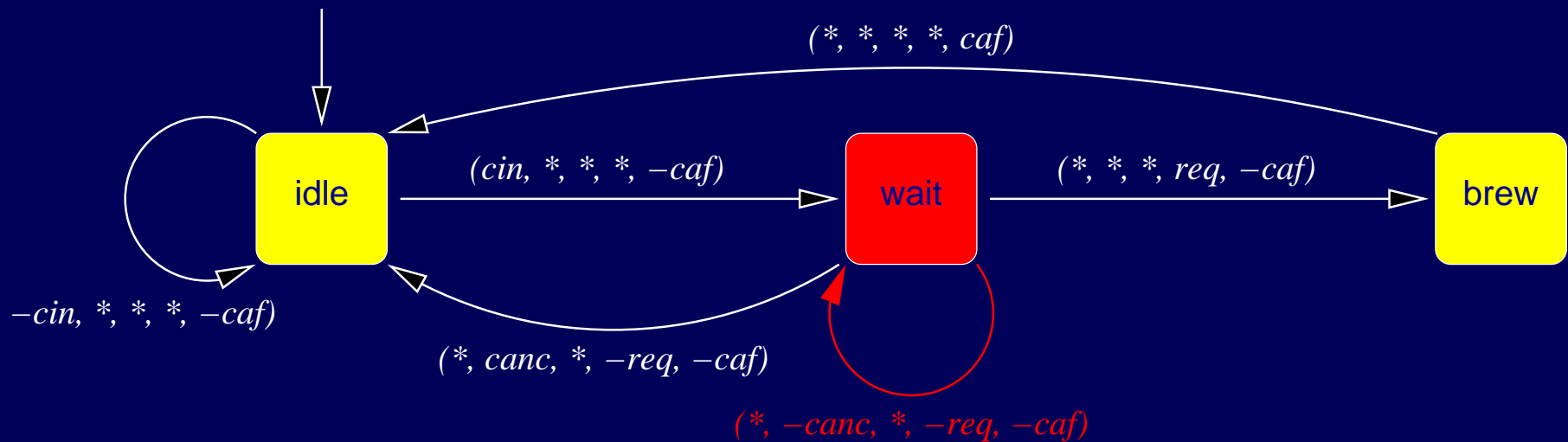


An example run

Financial administration

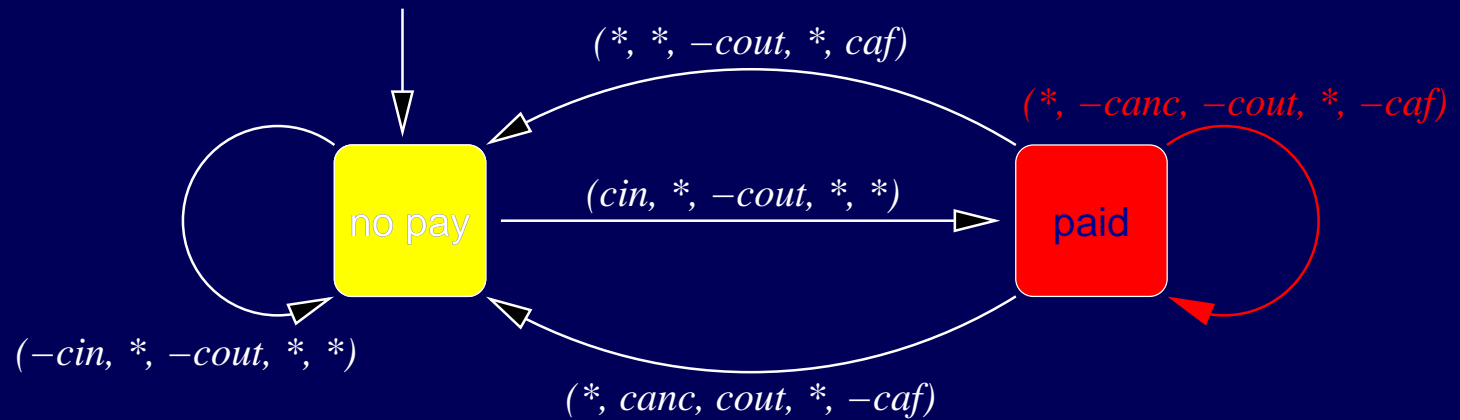


Brewer control

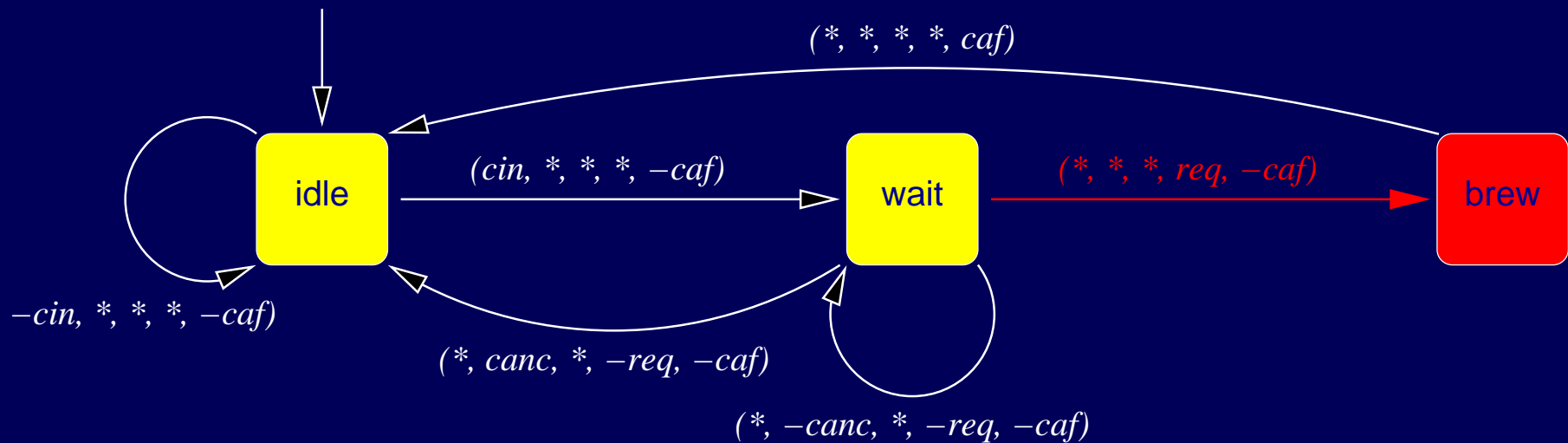


An example run

Financial administration

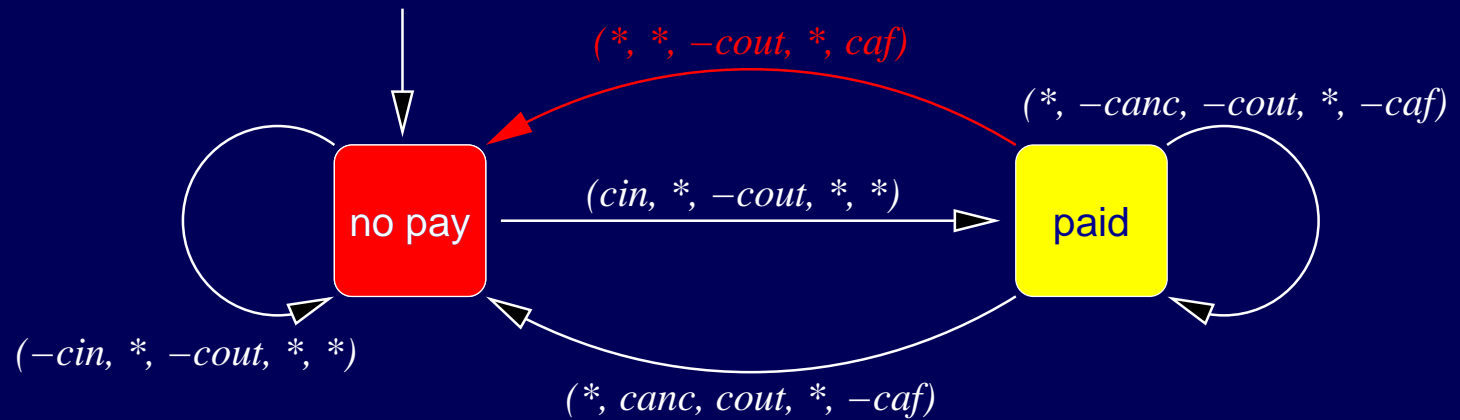


Brewer control

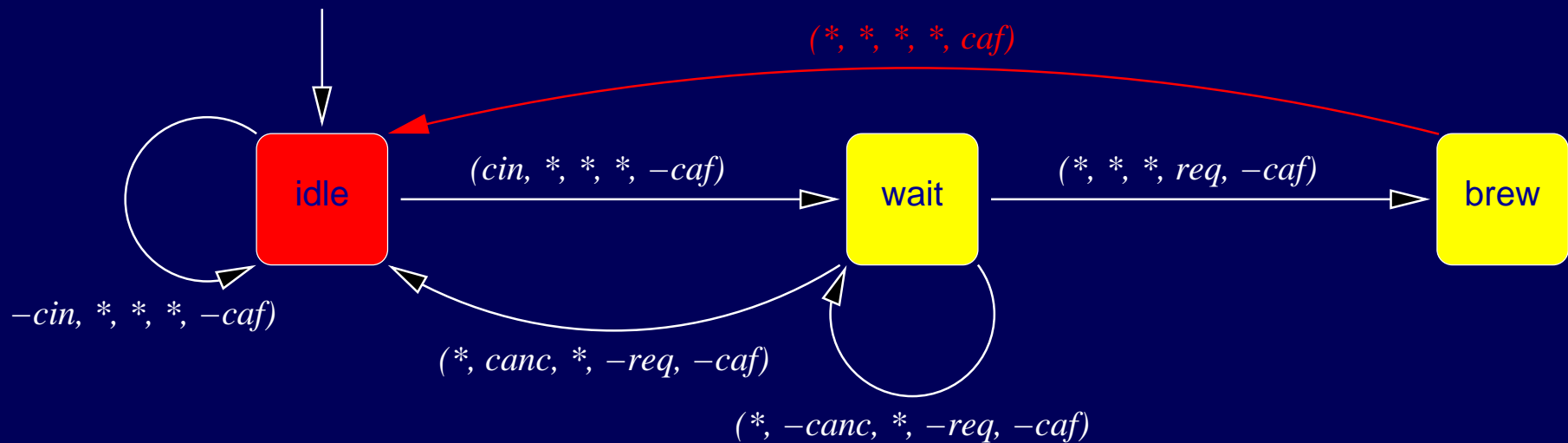


An example run

Financial administration

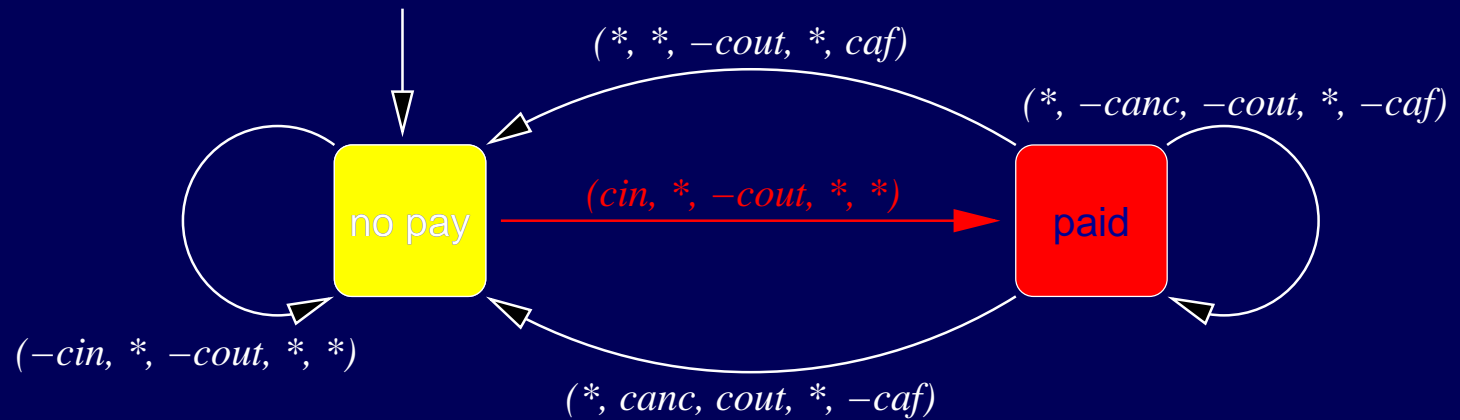


Brewer control

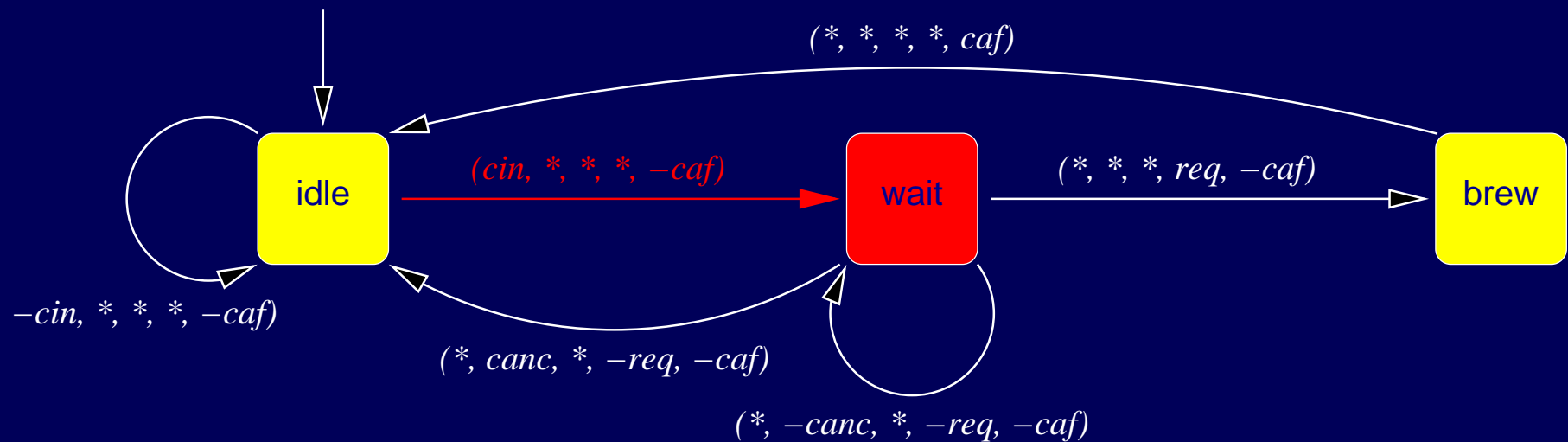


An example run

Financial administration

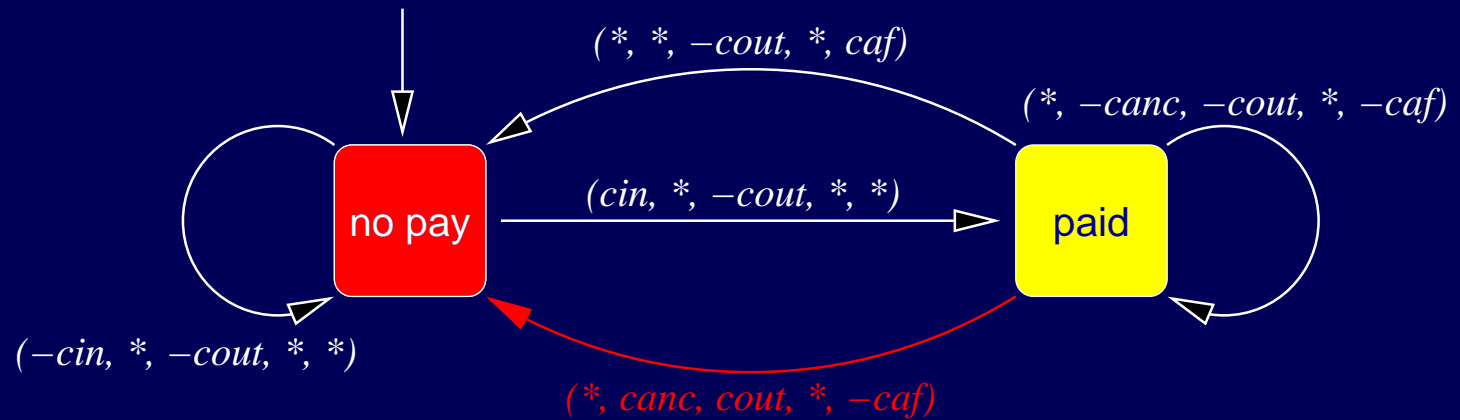


Brewer control

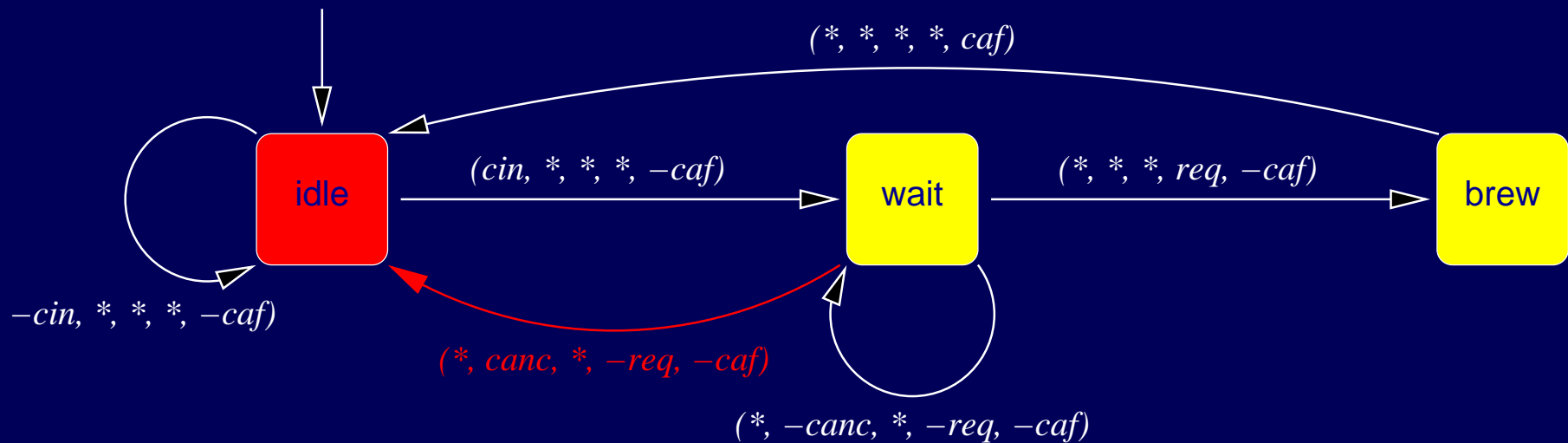


An example run

Financial administration

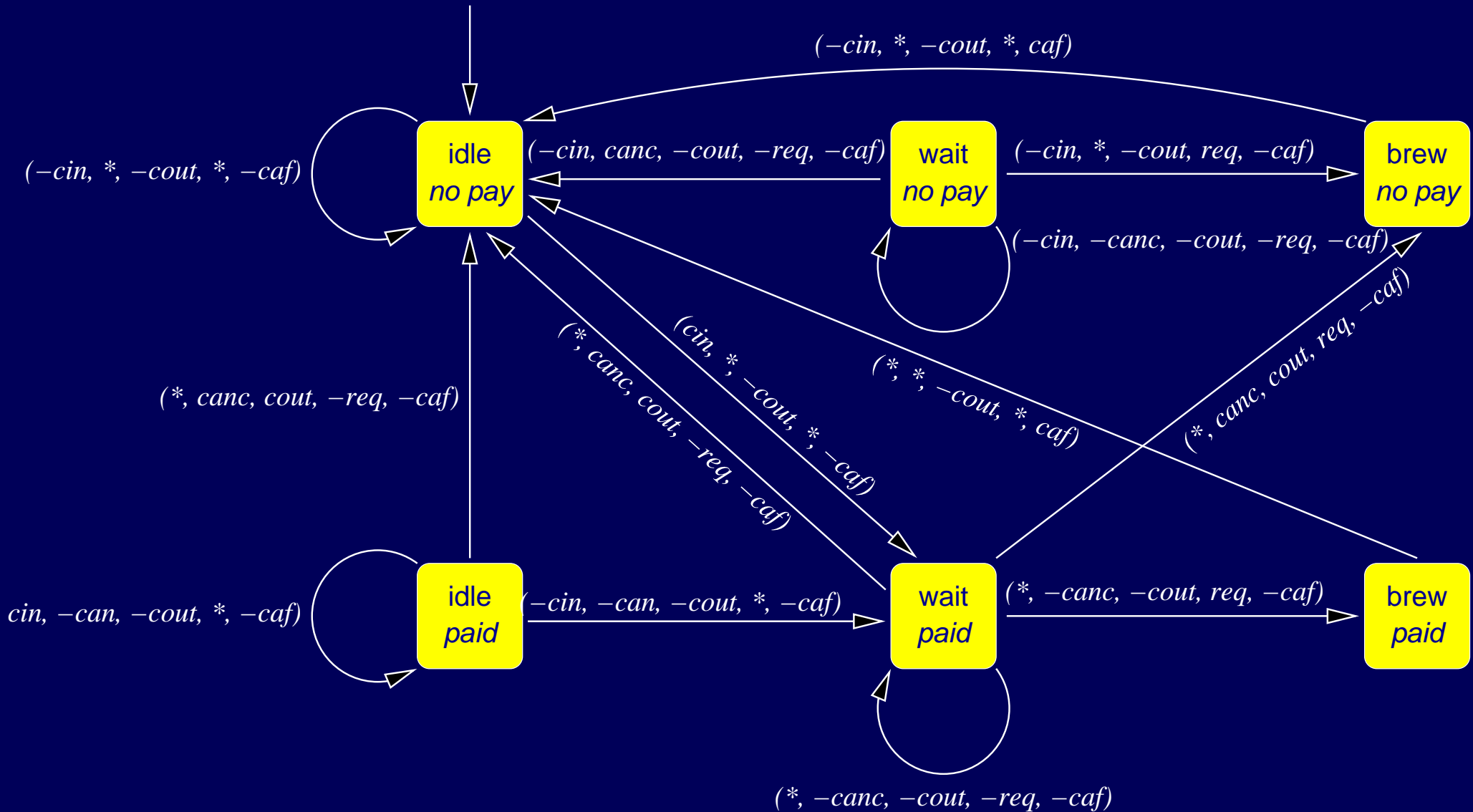


Brewer control



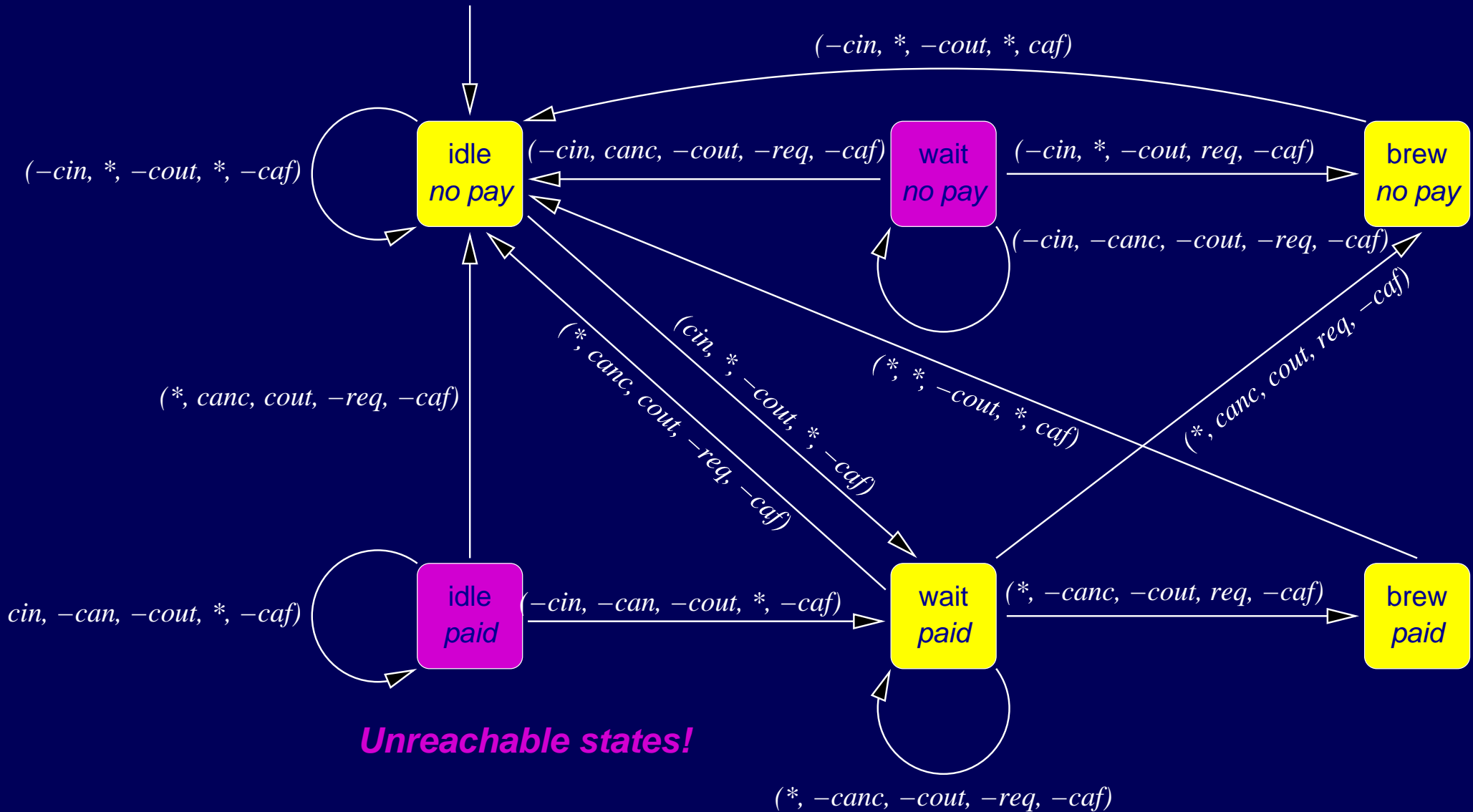
The coffee vending machine

Product automaton: Financial admin. * Brewer ctrl.



The coffee vending machine

Product automaton: Financial admin. * Brewer ctrl.



Pros and cons

- 😊 Standard notation of electrical engineering and computer science
- 😊 Automatically analyzable

Pros and cons

- 😊 Standard notation of electrical engineering and computer science
- 😊 Automatically analyzable
- 😞 No intrinsic distinction between in- and output; well-formedness can only be enforced through conventions of use
- 😞 Not scalable:
 - 10 parallel components of 8 states each yield a diagram with $8^{10} \approx 10^9$ nodes,
 - 4 integer variables of 32 bit each yield $(2^{32})^4 \approx 3 \cdot 10^{38}$ nodes.
- 😞 No inherent notion of time:
 - representing time requires fixing a firing rate of transitions — interferes with the idea that design models should be architecture-independent,
 - representing large (relative to the firing rate) time constants requires long transition chains, yielding unwieldy diagrams.

State charts

Idea

Add syntactic sugar to transition diagrams such that suitable abbreviations for unwieldy diagrams become available.

Idea

Add syntactic sugar to transition diagrams such that suitable abbreviations for unwieldy diagrams become available.

1. Symbolic variables, as in programming languages

Idea

Add syntactic sugar to transition diagrams such that suitable abbreviations for unwieldy diagrams become available.

1. Symbolic variables, as in programming languages
2. Explicit parallelism

Idea

Add syntactic sugar to transition diagrams such that suitable abbreviations for unwieldy diagrams become available.

1. Symbolic variables, as in programming languages
2. Explicit parallelism
3. Hierarchical nesting of subcharts,
 - priorities between competing actions/transitions
 - certain forms of behaviour inheritance

Idea

Add syntactic sugar to transition diagrams such that suitable abbreviations for unwieldy diagrams become available.

1. Symbolic variables, as in programming languages
2. Explicit parallelism
3. Hierarchical nesting of subcharts,
 - priorities between competing actions/transitions
 - certain forms of behaviour inheritance
4. Explicit in- and output, plus scoping rules:
 - A statechart will never constrain its input: any input is accepted at any time (doesn't imply that the chart will always react to that input!)
 - A statechart has complete control of the values of its outputs
 - Shared variables are controlled by all the partners: they are updated in an “interleave operations and overwrite previous values” fashion

Idea

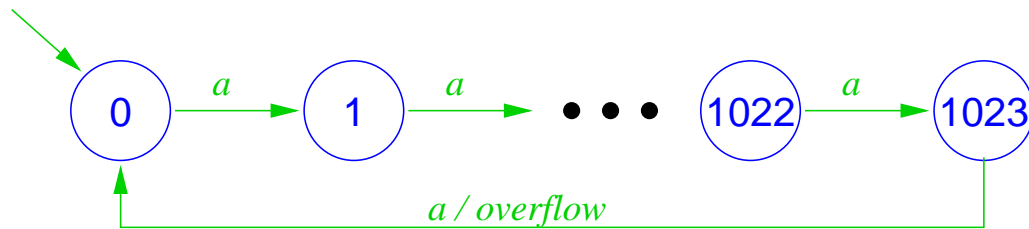
Add syntactic sugar to transition diagrams such that suitable abbreviations for unwieldy diagrams become available.

1. Symbolic variables, as in programming languages
2. Explicit parallelism
3. Hierarchical nesting of subcharts,
 - priorities between competing actions/transitions
 - certain forms of behaviour inheritance
4. Explicit in- and output, plus scoping rules:
 - A statechart will never constrain its input: any input is accepted at any time (doesn't imply that the chart will always react to that input!)
 - A statechart has complete control of the values of its outputs
 - Shared variables are controlled by all the partners: they are updated in an "interleave operations and overwrite previous values" fashion
5. Transition guards referring to physical (a.k.a. "real") time

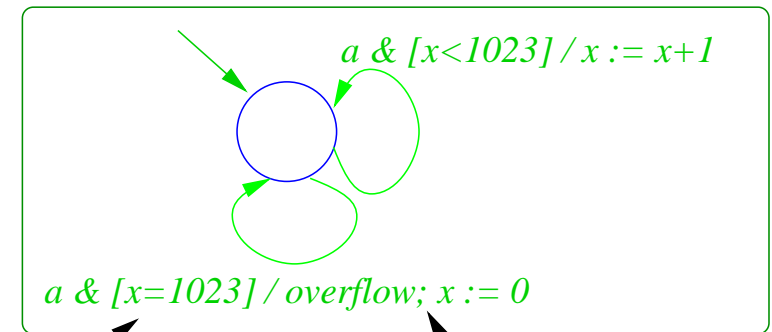
Statecharts: Distinguishing control from data

A 10-Bit counter, counting on event a and issuing overflow after 1024 occurrences:

As FSM:



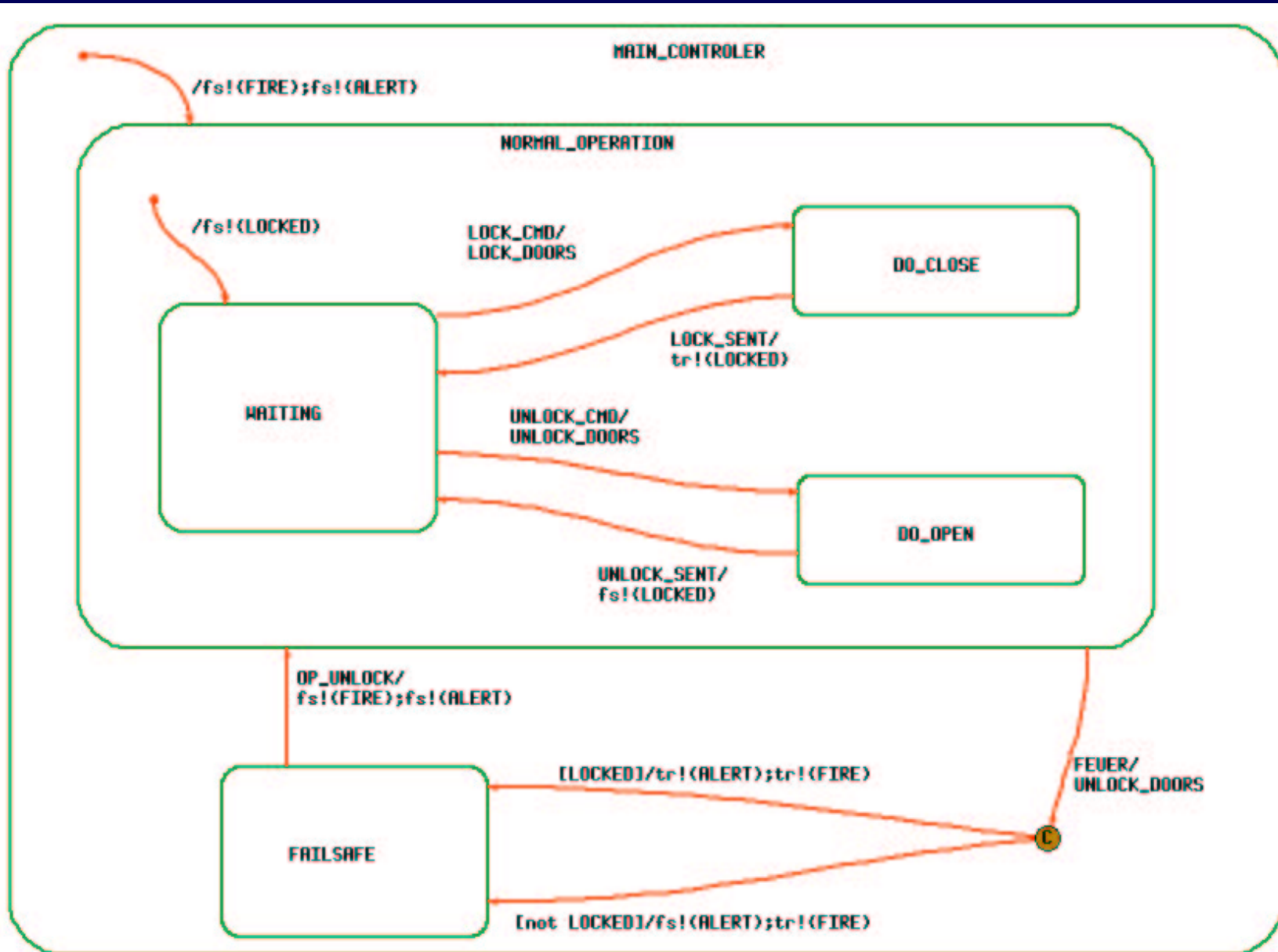
As Statechart:



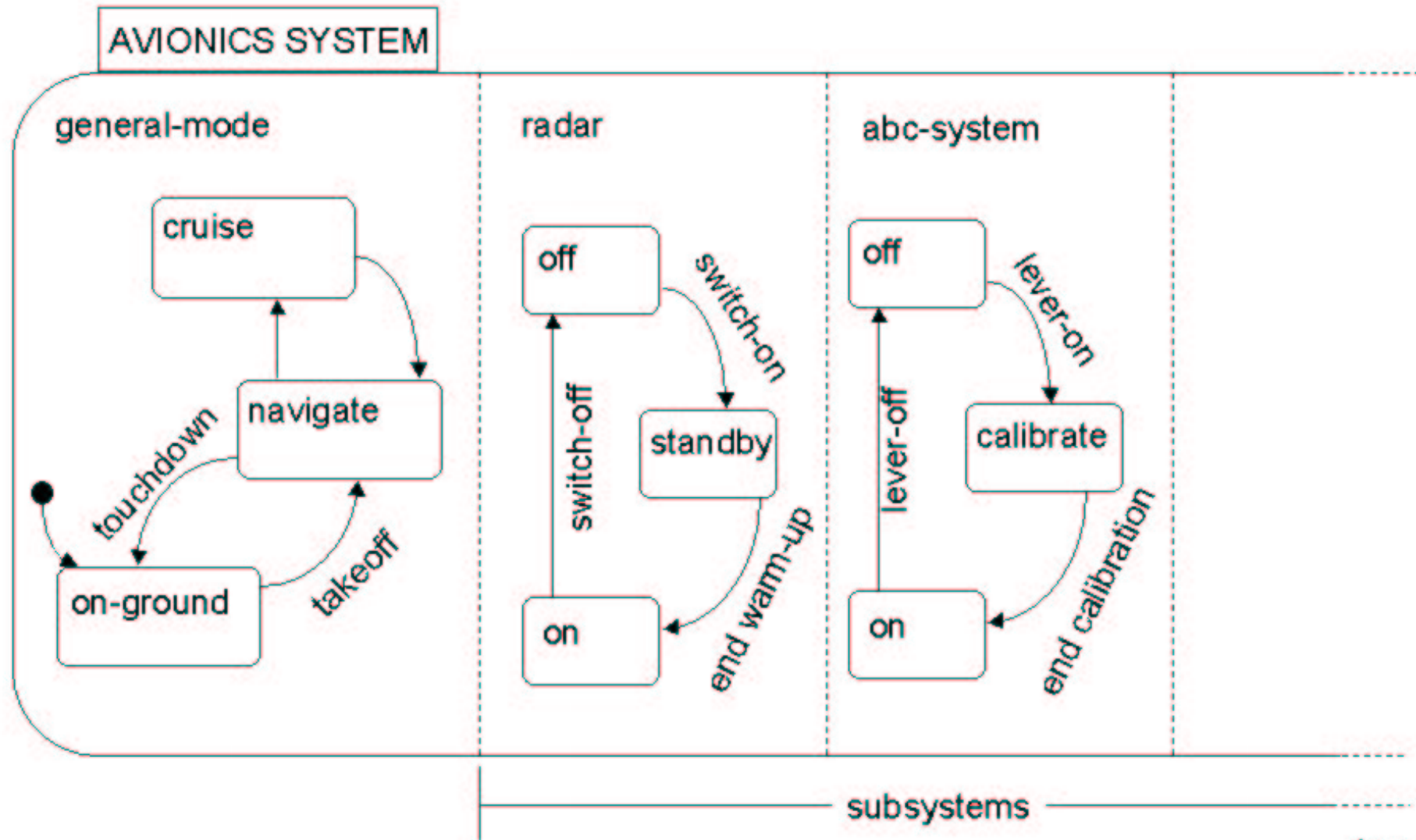
trigger condition:
events and/or state
predicate

action: event generation and/or
state assignment

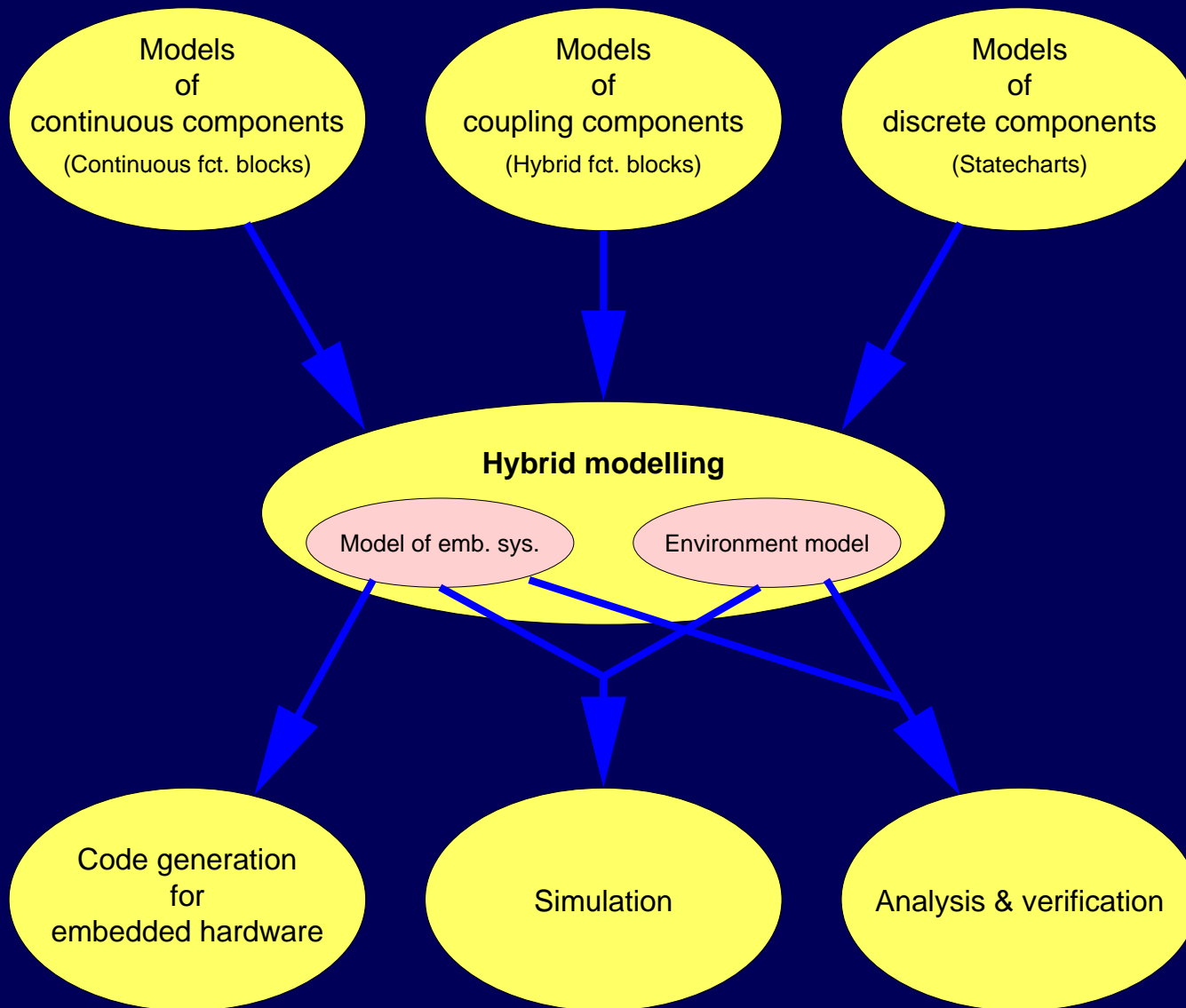
Statecharts: Hierarchy



Statecharts: Concurrency / AND-states



from: [Ha87]



Hybrid modelling

Rationale: Used to model

1. advanced control techniques (e.g., mode-switching control),
2. embedded system & environment in combination (“Virtual prototyping”).

⇒ Need a seamless semantic integration of e.g.

- continuous signal transducers,
- A/D & D/A functional blocks,
- FSMs / Statecharts.

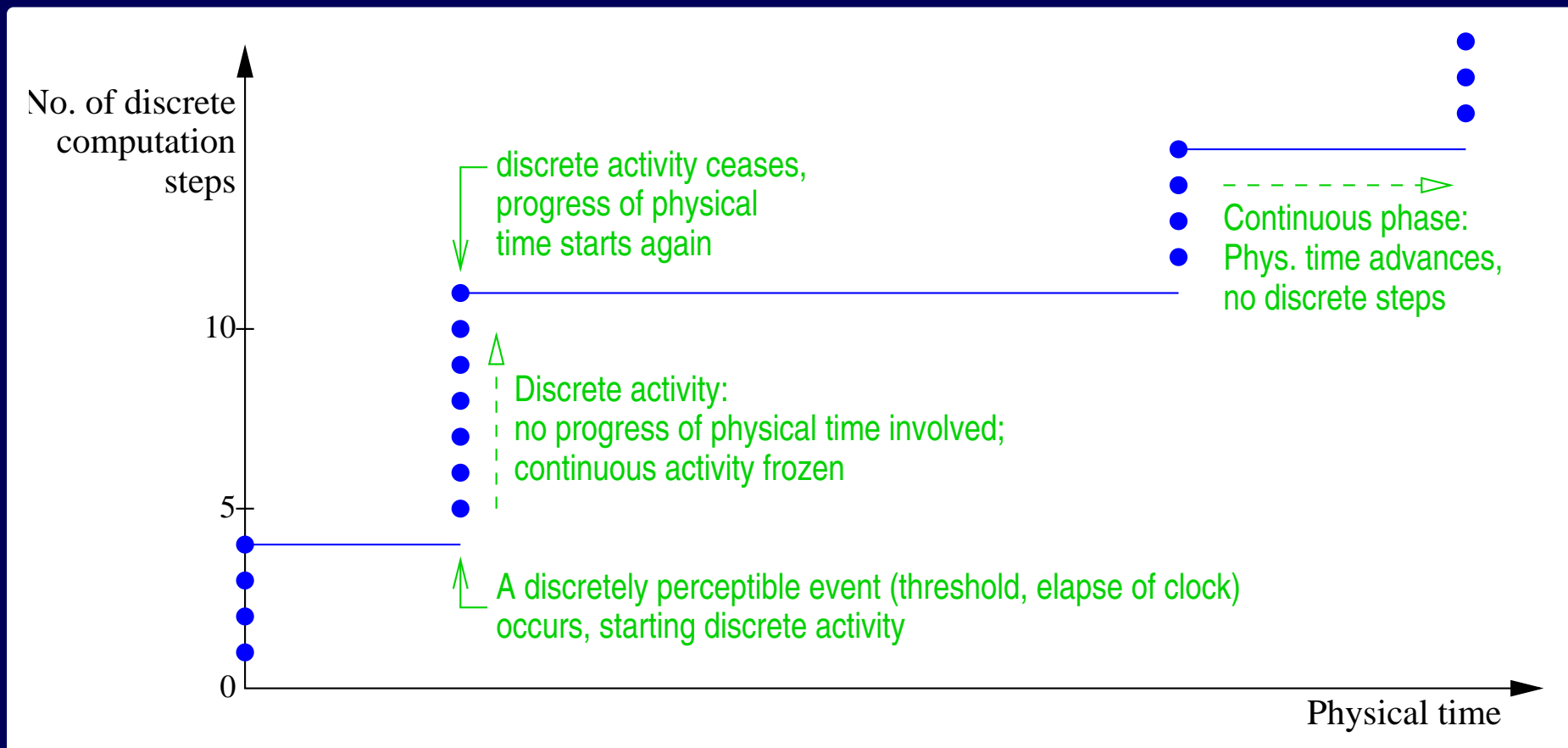
Integration of discrete actions

Design decisions concerning the semantic model:

1. How much time shall discrete actions take?
2. Is input sampling/output delivery instantaneous or durational?
3. What happens to input sampling/output delivery upon fast dynamics?

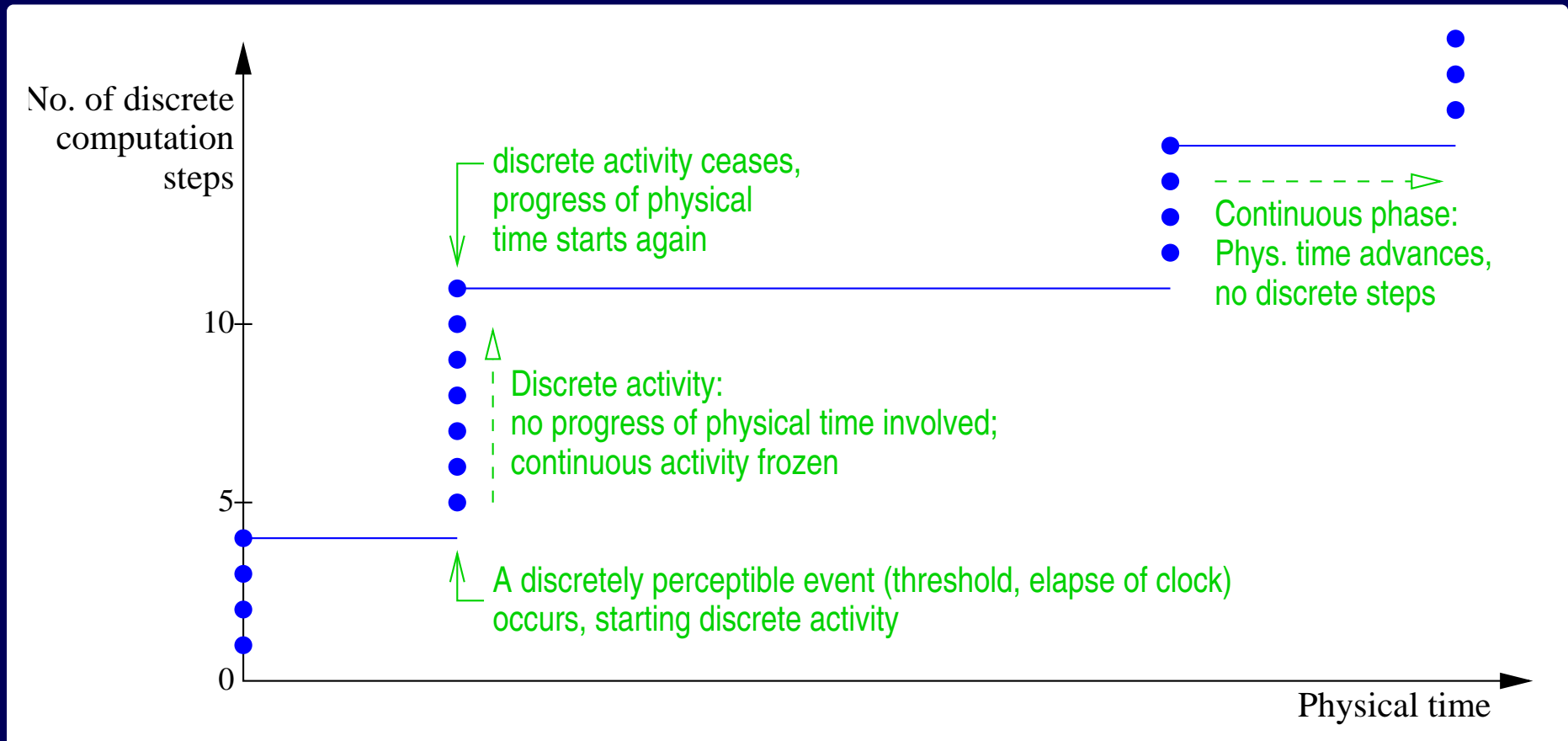
Two-dimensional time

through adoption of the so-called *synchrony hypothesis*: Computation time is deemed negligible compared to the pace of environment dynamics.



Two-dimensional time

through adoption of the so-called *synchrony hypothesis*: Computation time is deemed negligible compared to the pace of environment dynamics.



- Easy on simulation, but impossible to implement.
- Partially justified in case of slow environment dynamics.