Independence and Concurrent Separation Logic

Jonathan Hayman
joint work with Glynn Winskel

Computer Laboratory
University of Cambridge

13th August 2006

Introduction

This talk:

- Give a net semantics to a programming language
- Apply it to study the semantics of a program logic: Concurrent separation logic

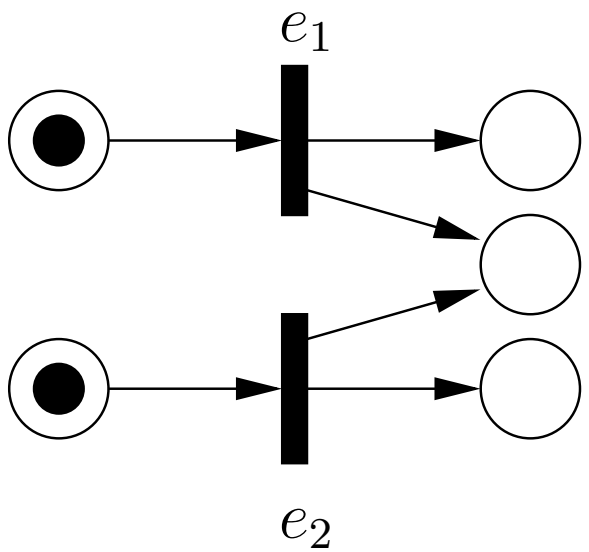
Introduction

- Independence models help alleviate problems with giving a standard, 'interleaved' semantics to parallel processes:
  - Atomicity: $\alpha \parallel \beta$ or $(\alpha_1; \alpha_2) \parallel \beta$
  - Genuine concurrent execution
  - Unnecessary interleavings
- Concurrent separation logic:
  - How is spatial separation related to independence?
  - The logic discriminates between parallel composition and interleaved expansion:

$$\begin{array}{lccc} \not\vdash & \{\ell \mapsto 0\} & \ell := 1 \parallel \ell := 2 & \{\ell \mapsto 1 \vee \ell \mapsto 2\} \\ \vdash & \{\ell \mapsto 0\} & (\ell := 1; \ell := 2) + (\ell := 2; \ell := 1) & \{\ell \mapsto 1 \vee \ell \mapsto 2\} \end{array}$$

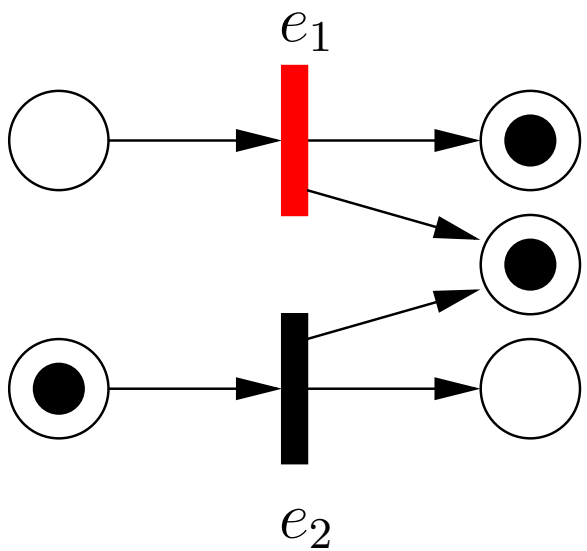
Petri nets

- A Petri net consists of:
  - *Conditions* (places / circles)
  - *Events* (transitions / rectangles)
  - An initial *marking* (subset of conditions)
- Events have *preconditions* and *postconditions*, determining whether they are able to fire in a marking
- Two events are said to be *independent* if they operate on disjoint sets of conditions



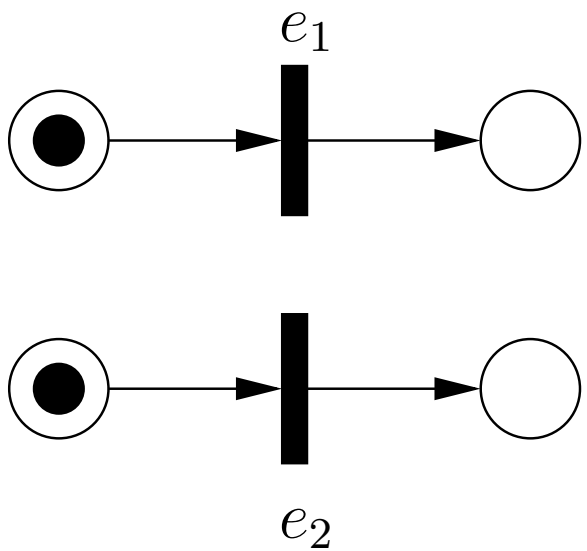
Petri nets

- A Petri net consists of:
  - *Conditions* (places / circles)
  - *Events* (transitions / rectangles)
  - An initial *marking* (subset of conditions)
- Events have *preconditions* and *postconditions*, determining whether they are able to fire in a marking
- Two events are said to be *independent* if they operate on disjoint sets of conditions



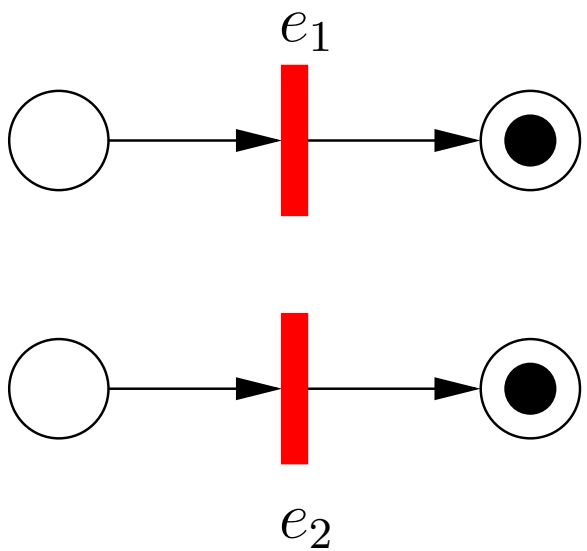
Petri nets

- A Petri net consists of:
  - *Conditions* (places / circles)
  - *Events* (transitions / rectangles)
  - An initial *marking* (subset of conditions)
- Events have *preconditions* and *postconditions*, determining whether they are able to fire in a marking
- Two events are said to be *independent* if they operate on disjoint sets of conditions



Petri nets

- A Petri net consists of:
  - *Conditions* (places / circles)
  - *Events* (transitions / rectangles)
  - An initial *marking* (subset of conditions)
- Events have *preconditions* and *postconditions*, determining whether they are able to fire in a marking
- Two events are said to be *independent* if they operate on disjoint sets of conditions



Terms

- Concurrent separation logic uses separating conjunction to develop Owicki-Gries proof rules to deal with pointer-manipulating programs
- Want a net semantics for terms of the following programming language

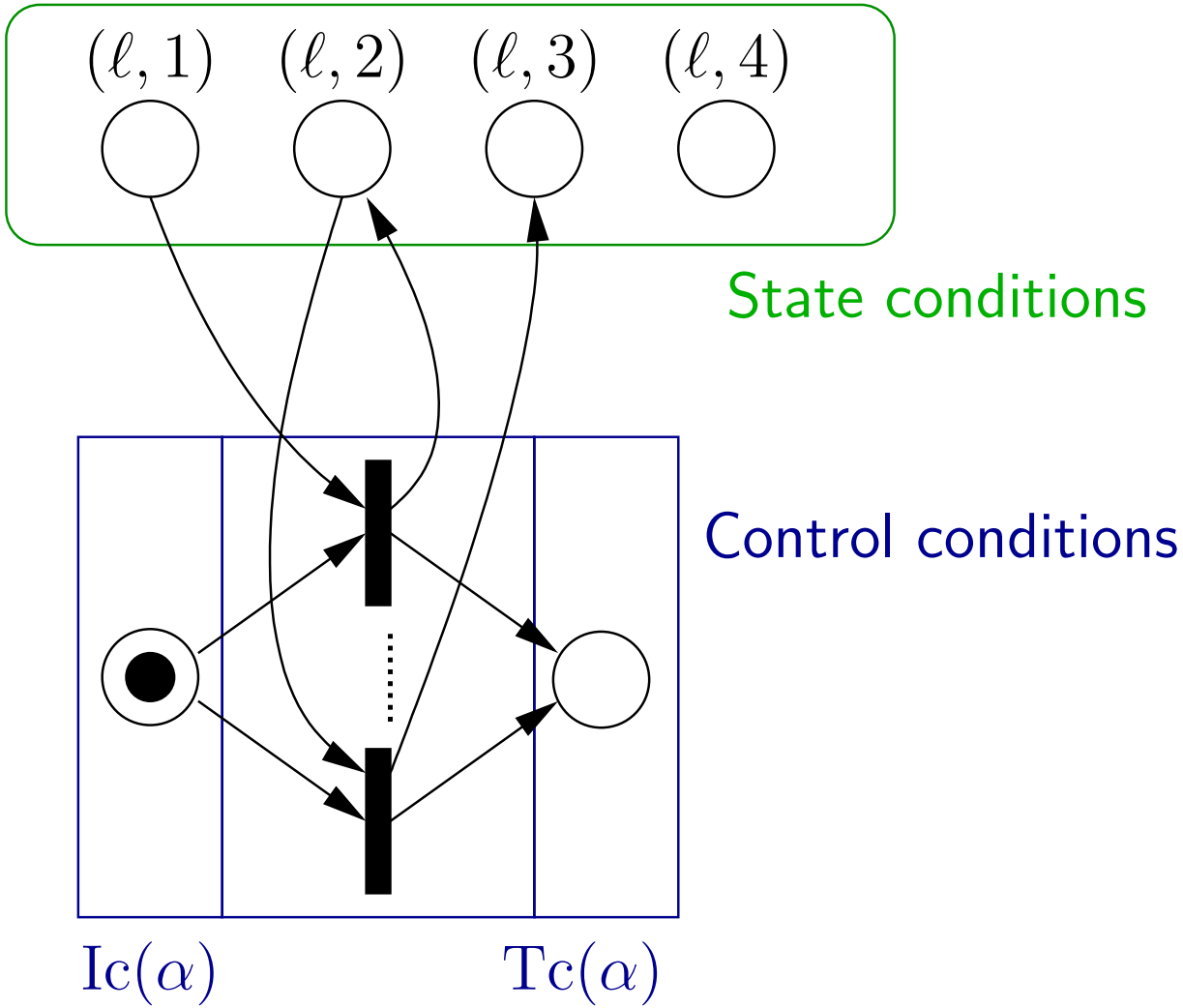
$$\begin{array}{rcl} t & ::= & \alpha \mid t;t \mid t \parallel t \mid \texttt{while}\ b\ \texttt{do}\ t\ \texttt{od} \\ & \mid & \texttt{alloc}(\ell) \mid \texttt{dealloc}(\ell) \mid \texttt{with}\ r\ \texttt{do}\ t\ \texttt{od} \\ & \mid & \ldots \end{array}$$

- α ranges over commands that affect the memory in which the programs execute (e.g. assignments)

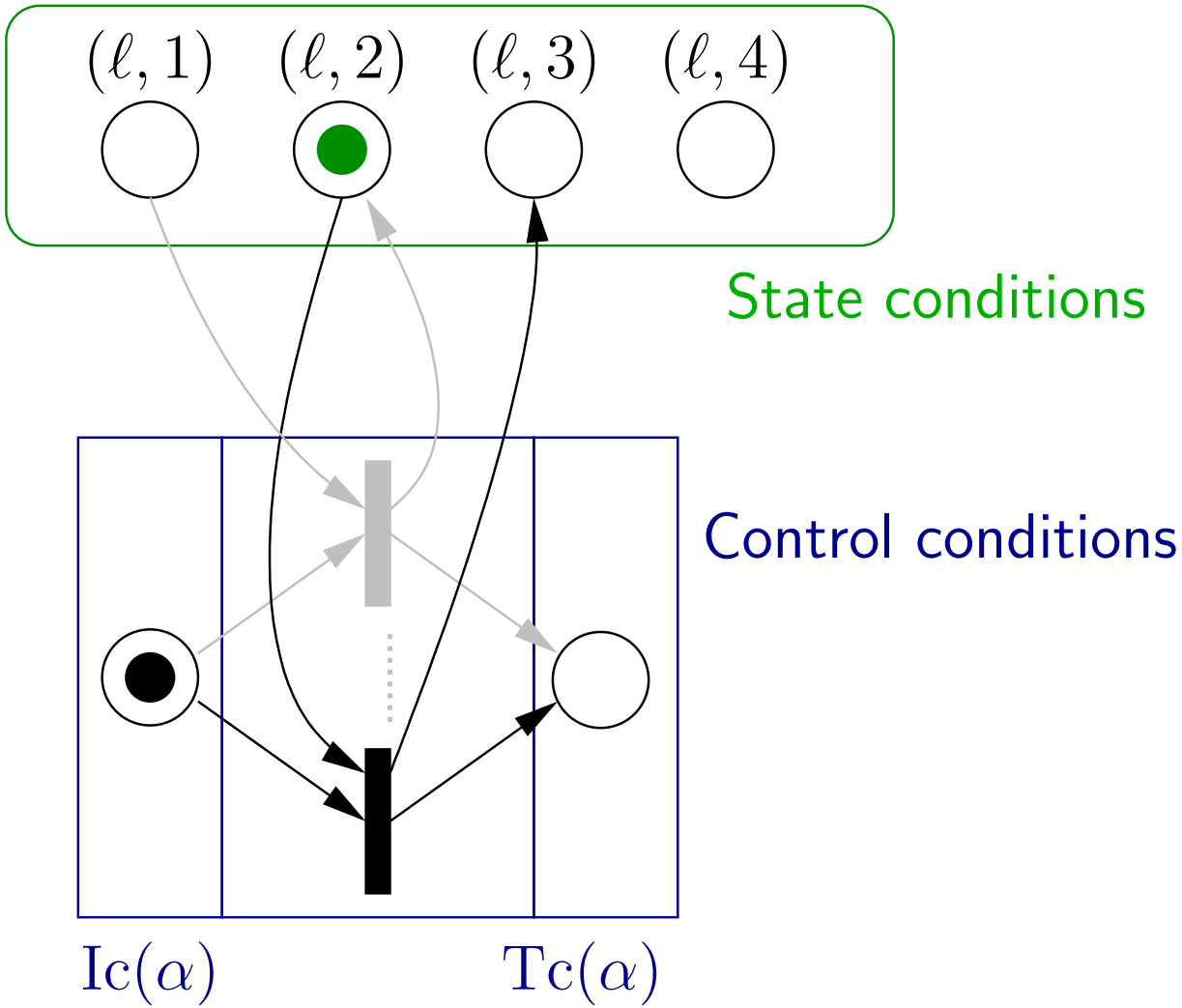
Net semantics

- Within nets formed, shall distinguish two forms of condition: *control* conditions and *state* conditions.
- The semantics $\mathcal{N}[\![t]\!]$ of a process consists of:
  - Events
  - Initial marking of control conditions
  - Terminal marking of control conditions
- State conditions are:
  - (ℓ, v) for each $\ell \in \mathrm{Loc}$ and $v \in \mathrm{Val}$. Marking H
  - $\mathsf{curr}(\ell)$ for each $\ell \in \mathrm{Loc}$. Marking L
  - r for each $r \in \mathrm{Res}$. Marking R
- The marking of state conditions must be *consistent*:
  H is a partial function defined at the locations current in L

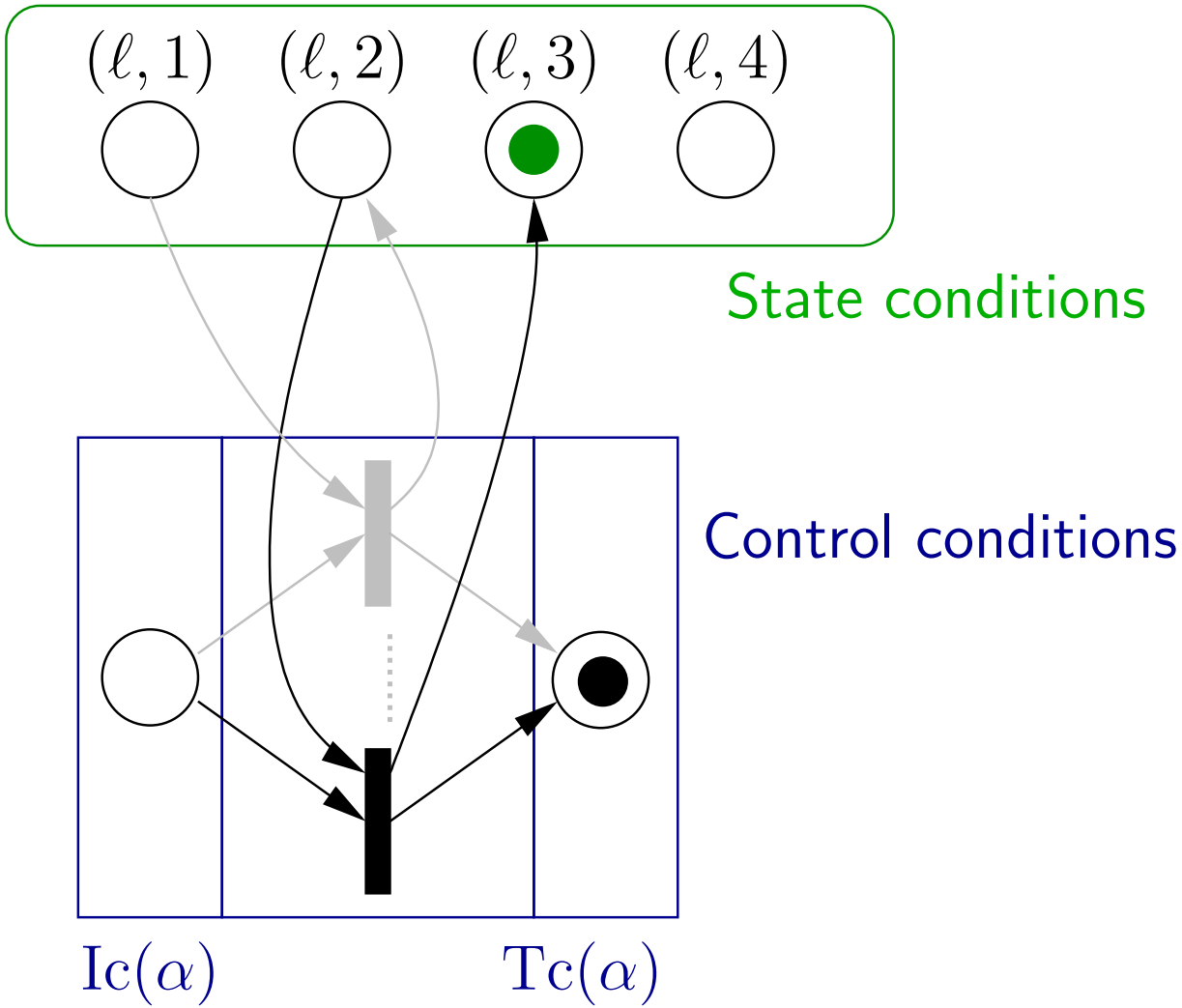
Action


$(\ell, 1)$
$(\ell, 2)$
$(\ell, 3)$
$(\ell, 4)$
State conditions
Control conditions
$\mathrm{Ic}(\alpha)$
$\mathrm{Tc}(\alpha)$

Action


$(\ell, 1)$
$(\ell, 2)$
$(\ell, 3)$
$(\ell, 4)$
State conditions
Control conditions
$\mathrm{Ic}(\alpha)$
$\mathrm{Tc}(\alpha)$

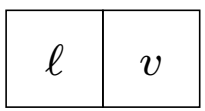
Action


$(\ell, 1)$
$(\ell, 2)$
$(\ell, 3)$
$(\ell, 4)$
State conditions
Control conditions
$\mathrm{Ic}(\alpha)$
$\mathrm{Tc}(\alpha)$

Allocation

`alloc`(ℓ):

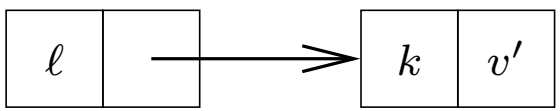
- Make an arbitrary non-current location current, assign an (arbitrary) initial value to it, and make ℓ point to the new location.
- If ℓ initially holds v and run `alloc`(ℓ):



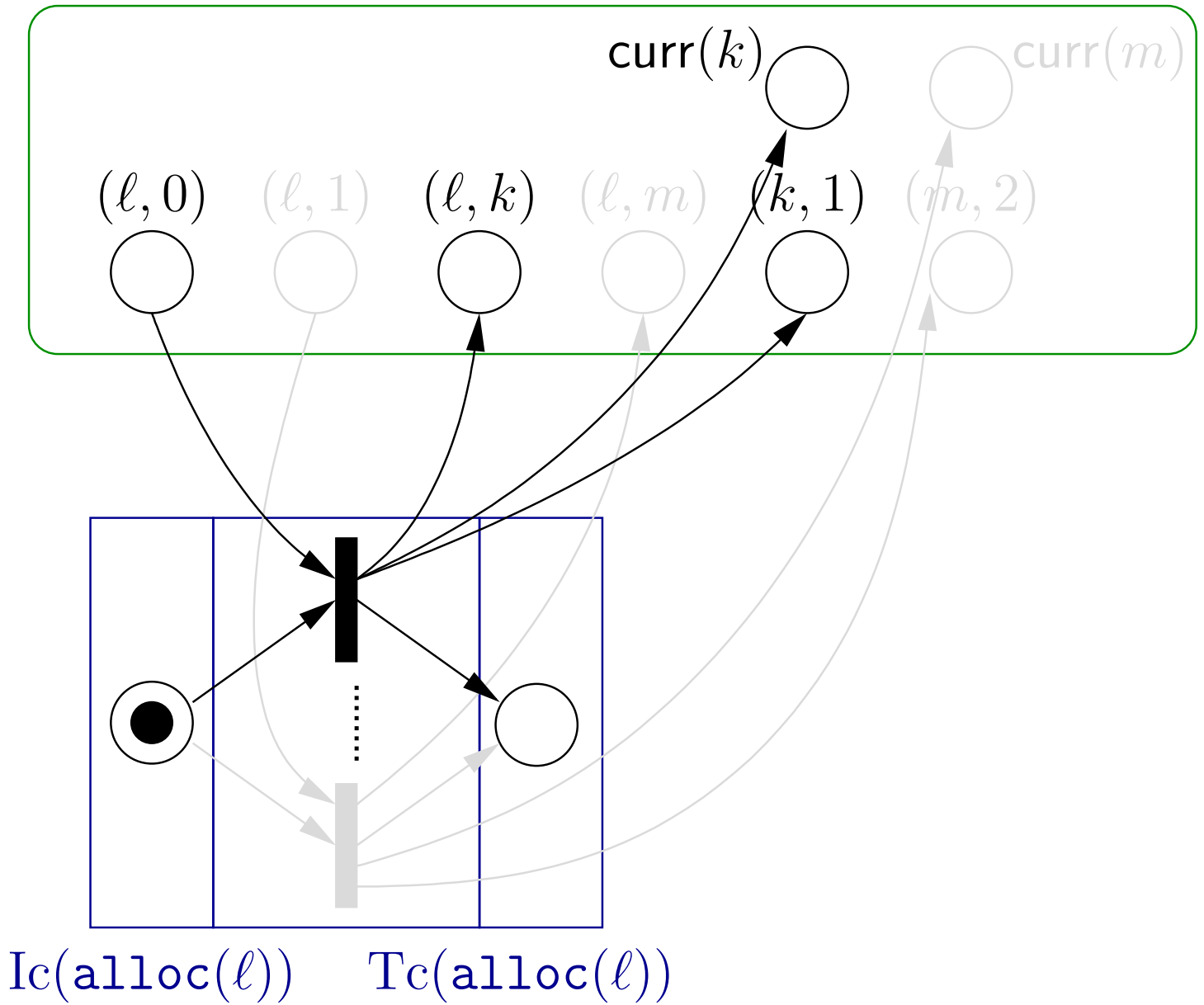
Allocation

`alloc`(ℓ):

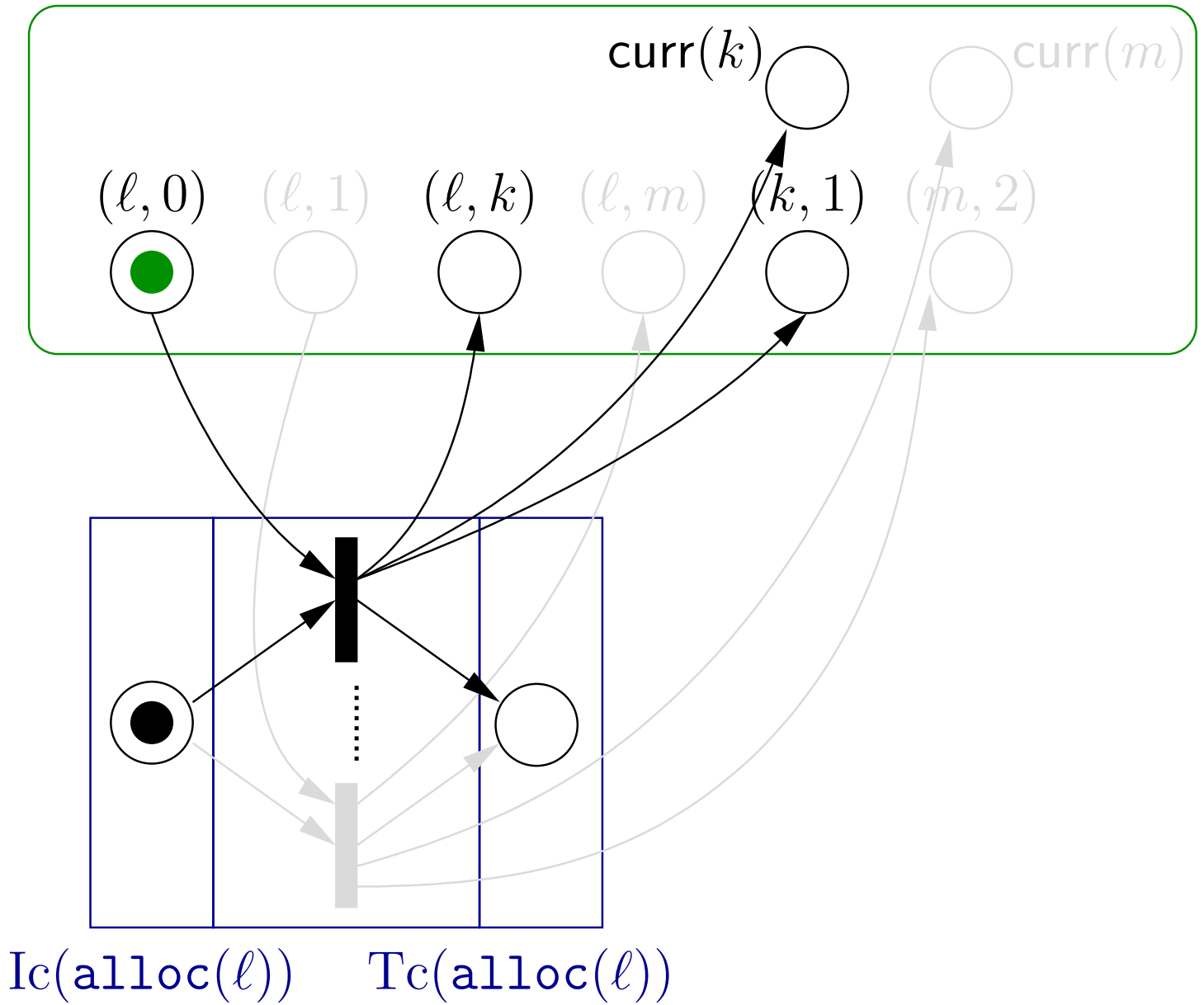
- Make an arbitrary non-current location current, assign an (arbitrary) initial value to it, and make ℓ point to the new location.
- If ℓ initially holds v and run `alloc`(ℓ):



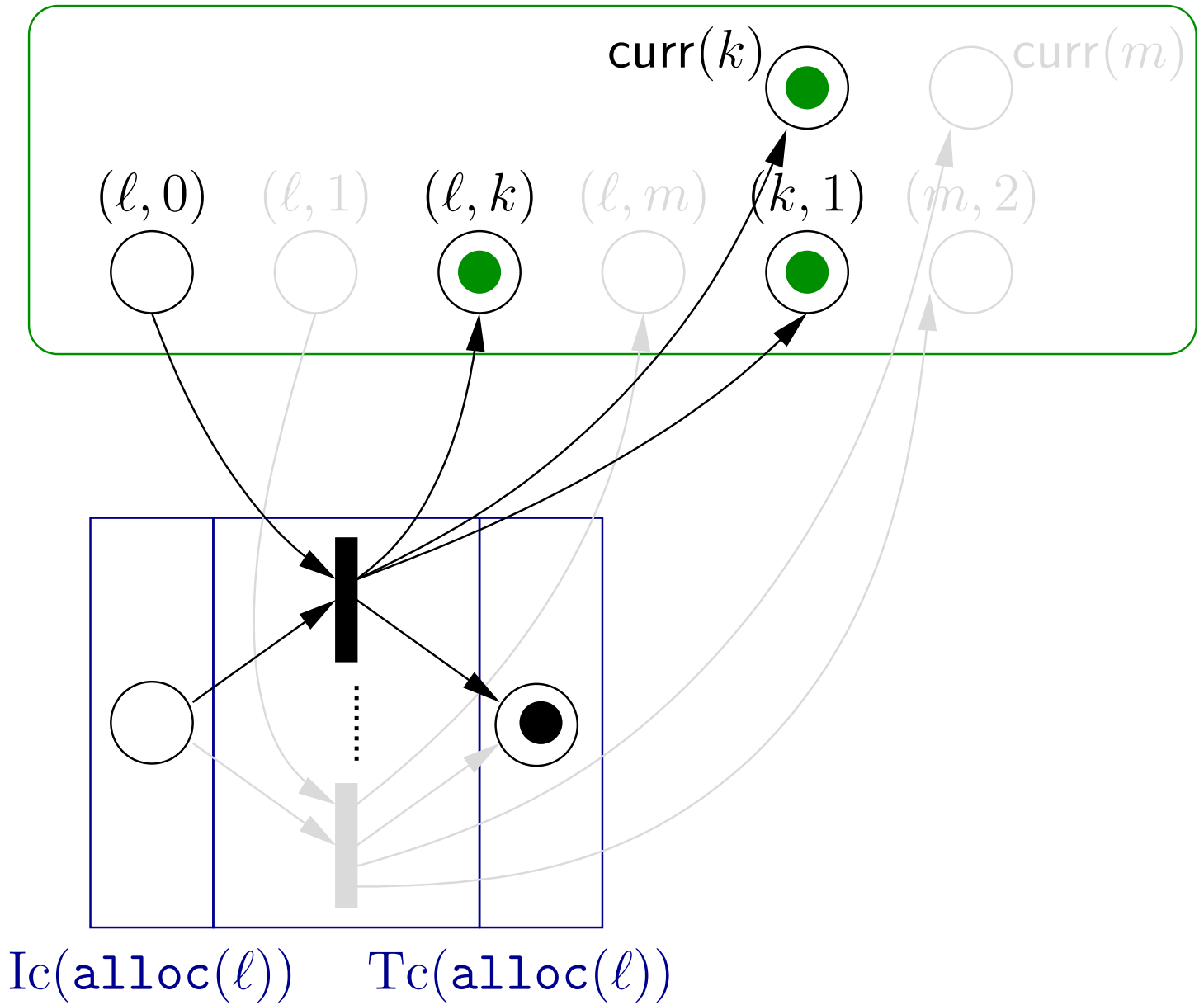
Allocation



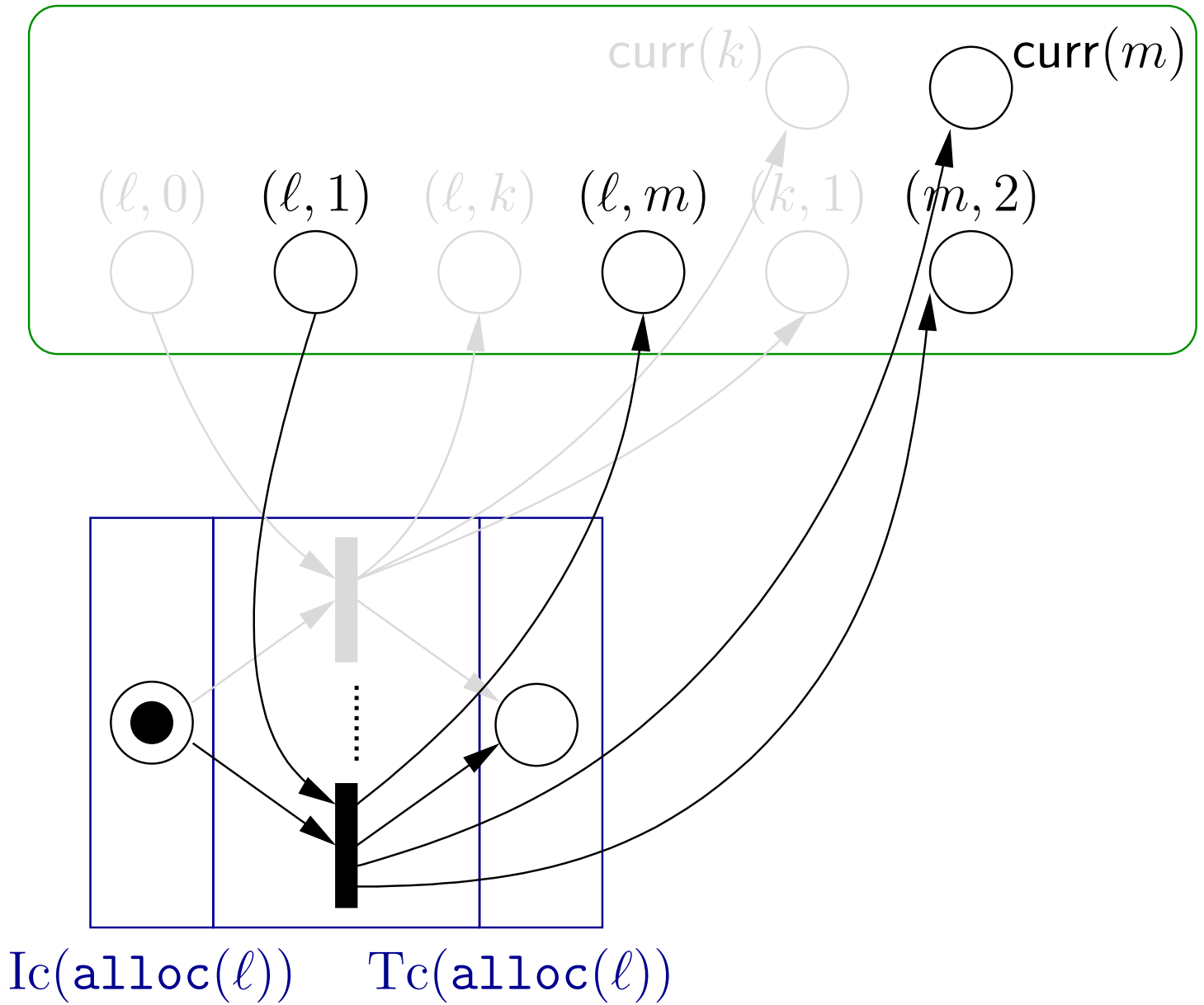
Allocation


curr(k)
curr(m)
$(\ell, 0)$
$(\ell, 1)$
(ℓ, k)
(ℓ, m)
$(k, 1)$
$(m, 2)$
$\mathrm{Ic}(\texttt{alloc}(\ell))$
$\mathrm{Tc}(\texttt{alloc}(\ell))$

Allocation

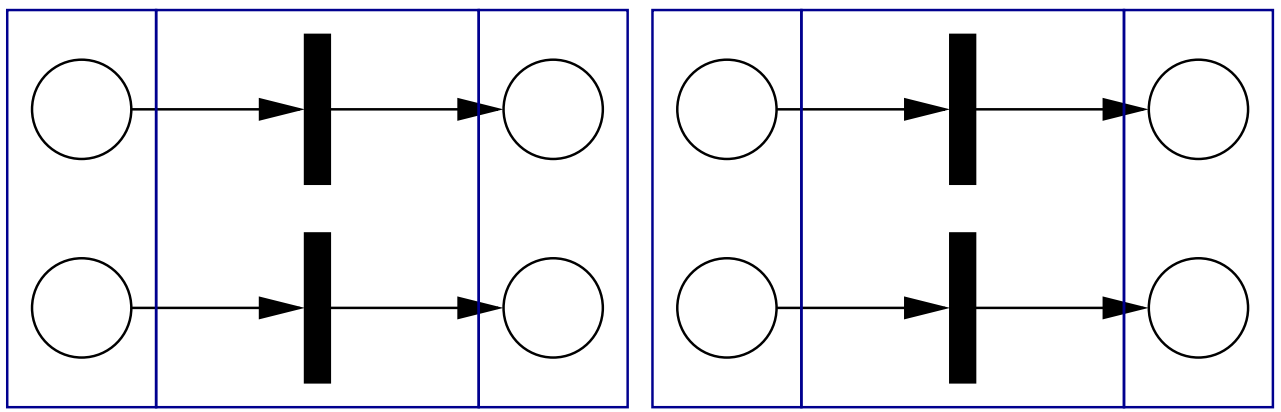


Allocation



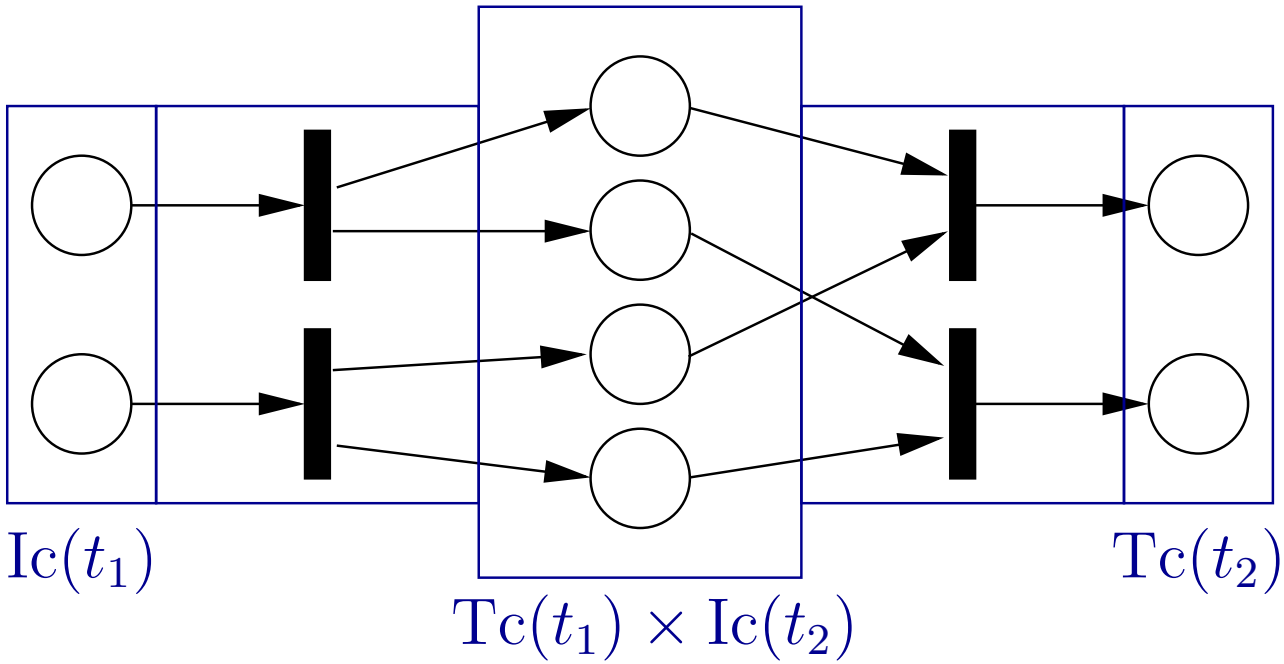
Structural connectives

- Operations on control part of net allow us to form the usual operations such as:
- Parallel composition:



Structural connectives

- Operations on control part of net allow us to form the usual operations such as:
- Sequential composition



Operational semantics

Theorem

The net semantics corresponds to the operational semantics.

Concurrent separation logic

- Key judgement $\Gamma \vdash \{\varphi\}\ t\ \{\psi\}$
- φ is an assertion about the heap (assignment of values to locations):

  $H \models \ell \mapsto v$ iff $H = \{(\ell, v)\}$

  $H \models \mathrm{emp}$ iff $H = \emptyset$

  $H \models \varphi_1 * \varphi_2$ iff there exist H_1, H_2 s.t. H_1 disjoint H_2 and $H = H_1 \cup H_2$ and $H_1 \models \varphi_1$ and $H_2 \models \varphi_2$

  with usual $\neg$, $\wedge$, $\vee$, etc.
- Meaning (approximate):

  Whenever t runs in a heap that has a part satisfying φ, the resulting heap has a part satisfying ψ. Moreover, t never accesses locations outside the identified part of the heap.
- Example: $\Gamma \vdash \{\ell \mapsto 0\}\ \ell := 1\ \{\ell \mapsto 1\}$

  Non-example: $\Gamma \nvdash \{k \mapsto 0\}\ \ell := 1\ \{k \mapsto 1\}$

Concurrent separation logic

- Permits a simple rule for parallel composition:

$$\frac{\Gamma \vdash \{\varphi_1\}\ t_1\ \{\psi_1\} \quad \Gamma \vdash \{\varphi_2\}\ t_2\ \{\psi_2\}}{\Gamma \vdash \{\varphi_1 * \varphi_2\}\ t_1 \parallel t_2\ \{\psi_1 * \psi_2\}}$$

- Example: $\Gamma \vdash \{\ell \mapsto 0 * k \mapsto 0\}\ \ell := 1 \parallel k := 2\ \{\ell \mapsto 1 * k \mapsto 2\}$
  Non-example: $\Gamma \nvdash \{\ell \mapsto 0 * k \mapsto 0\}\ \ell := 1 \parallel \ell := 2\ \{\top\}$
- Critical regions: (χ precise)

$$\frac{\Gamma \vdash \{\varphi * \chi\}\ t\ \{\psi * \chi\}}{\Gamma, r : \chi \vdash \{\varphi\}\ \texttt{with}\ r\ \texttt{do}\ t\ \texttt{od}\ \{\psi\}}$$

- Allows "transfer" of heap:
  Let the invariant $\chi \equiv k \mapsto 0 \vee k \mapsto 1 * \ell \mapsto 0$.

$$r : \chi \vdash \{\mathrm{emp}\}\ \texttt{with}\ r\ \texttt{do}\ \texttt{if}\ k = 1\ \texttt{then}\ k := 0\ \texttt{od}\ \{\ell \mapsto 0\}$$

- Initially proved sound by Brookes (with 'local' semantics)

Formal model

- Want a definition of validity $\Gamma \models \{\varphi\}\ t\ \{\psi\}$ that allows us to prove rule for parallel composition sound

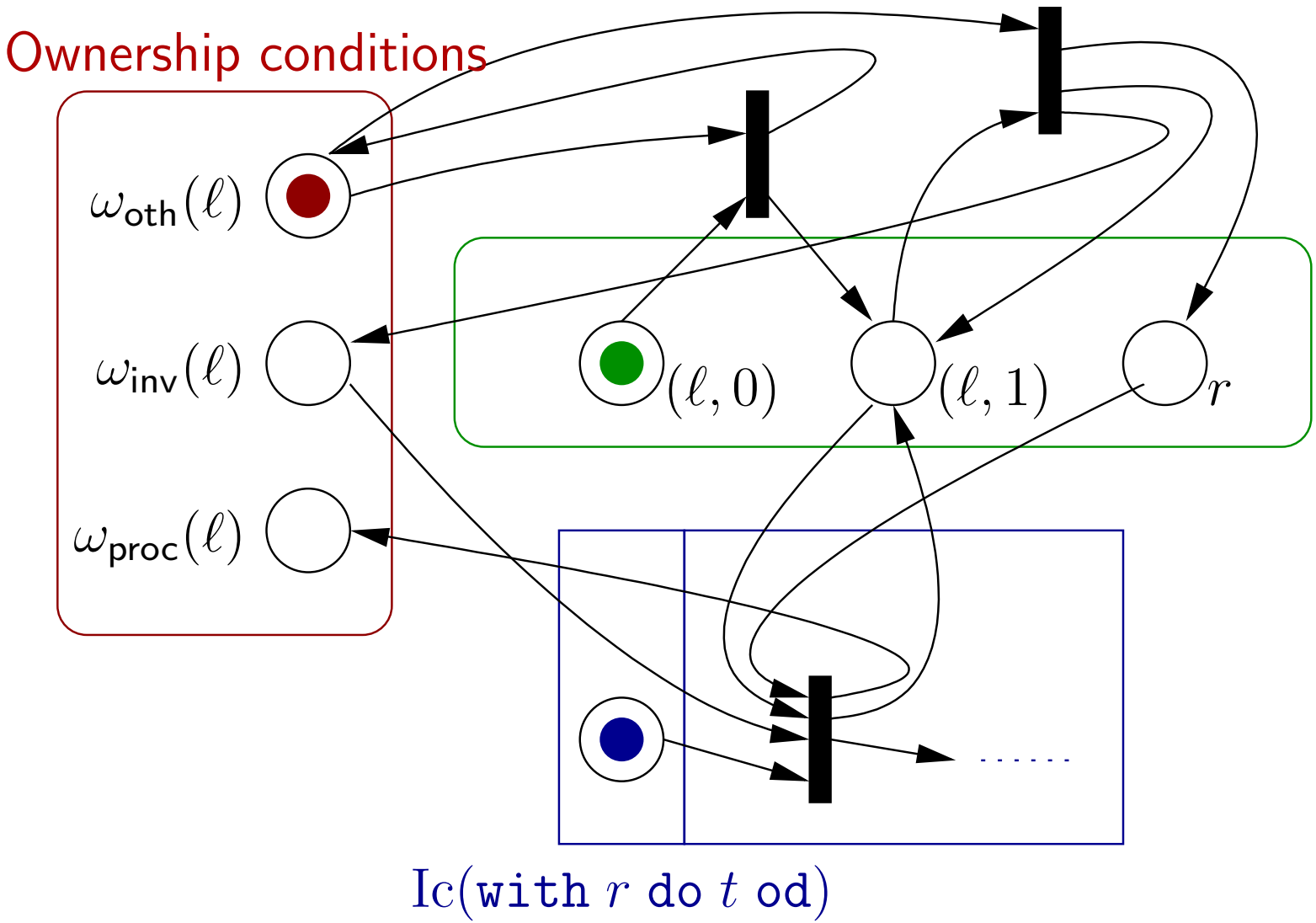
  > *Whenever t runs to completion in an environment where other well-behaved processes act from a heap where it owns locations satisfying φ, the resulting heap that the process owns satisfies ψ. Moreover, t is well-behaved.*

- Place in parallel with an *interference net* containing events to simulate the permitted forms of interference
- Intuition presented by O'Hearn is *ownership*, which constrains interference

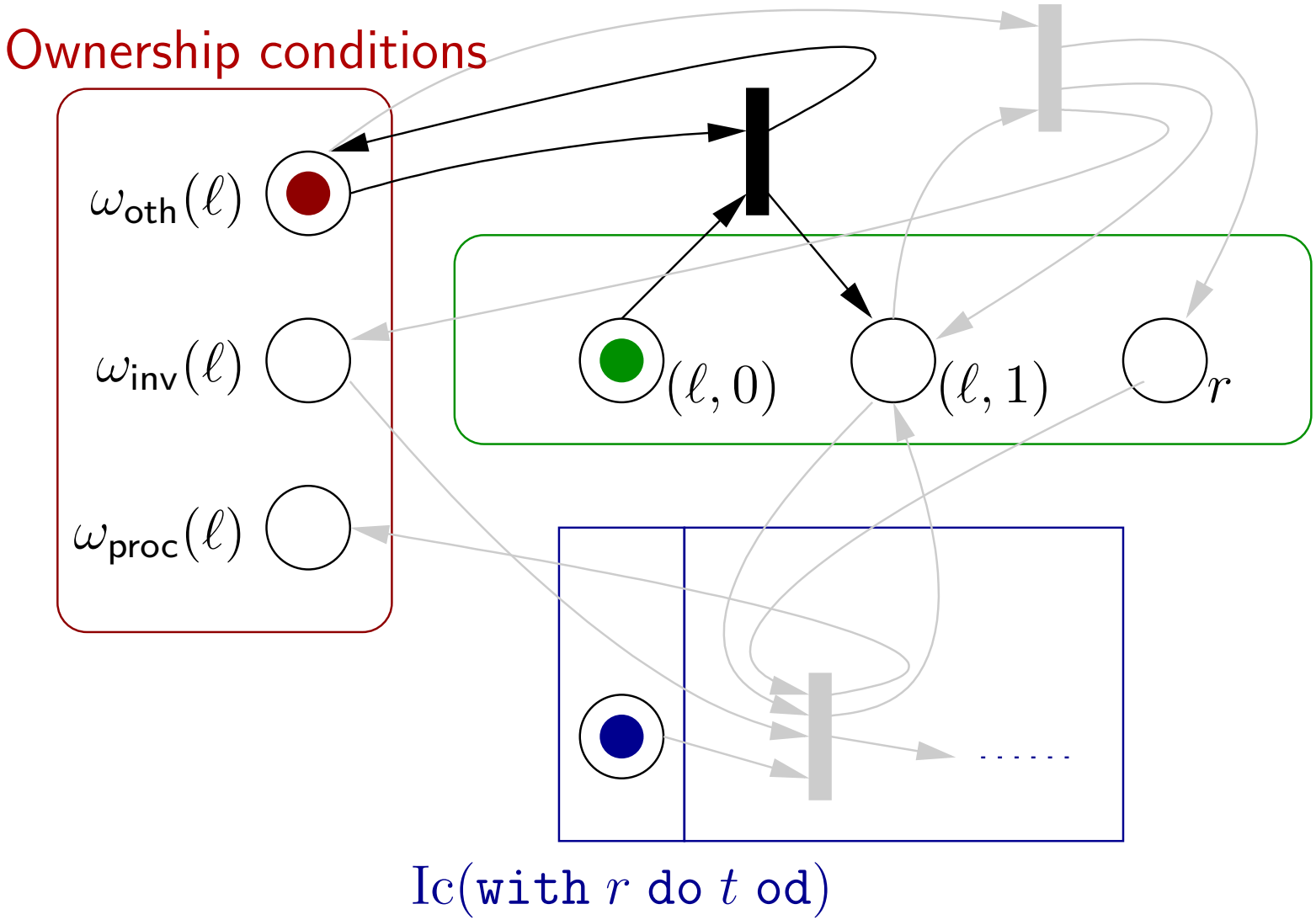
Interference and synchronization

- Add events to the net to represent the execution of other processes
- Possible behaviour of other processes depends on 'ownership' of the process, so add conditions $\omega_{\mathsf{proc}}(\ell)$, $\omega_{\mathsf{oth}}(\ell)$ and $\omega_{\mathsf{inv}}(\ell)$ to the net

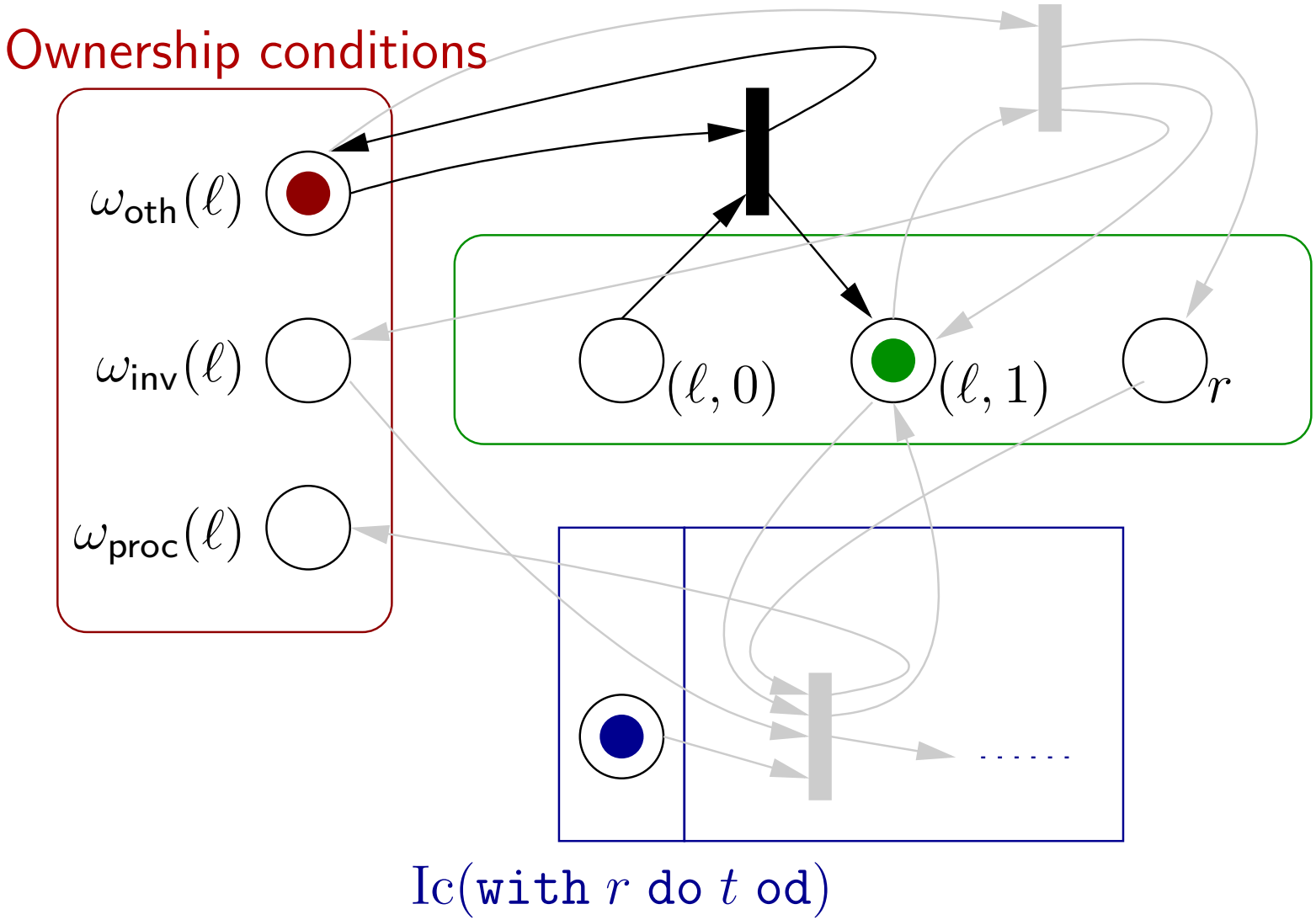
Interference and synchronization



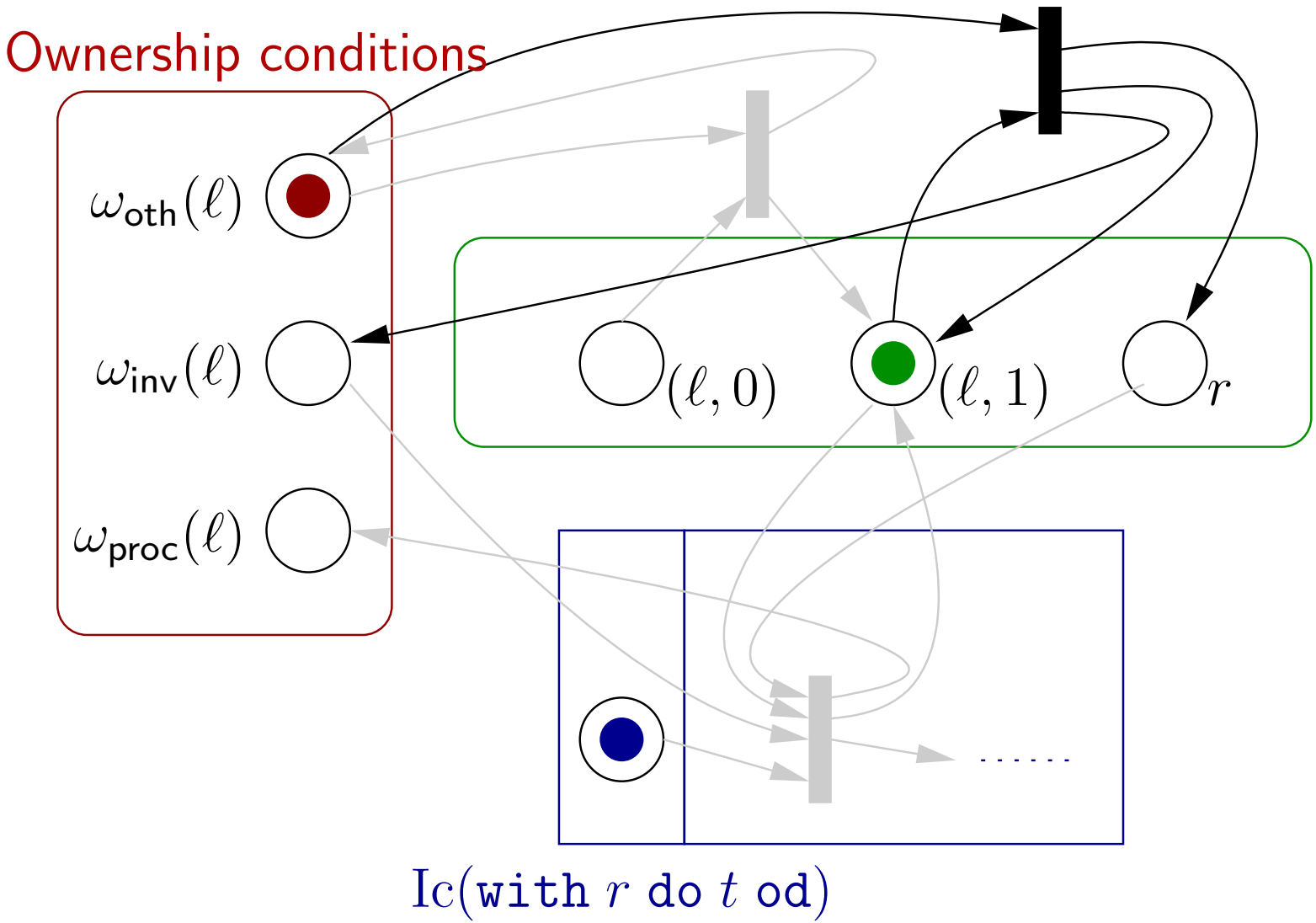
Interference and synchronization



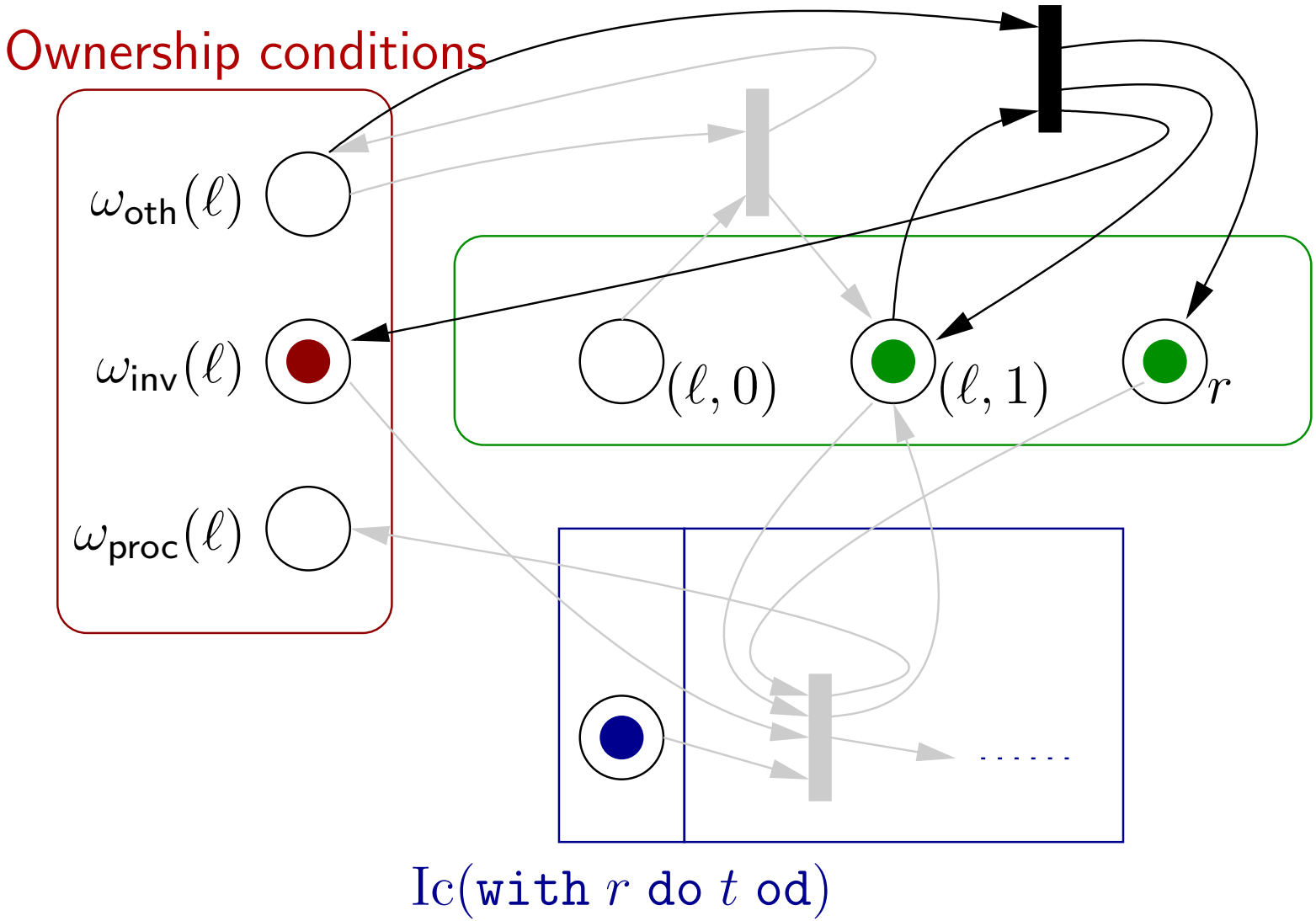
Interference and synchronization



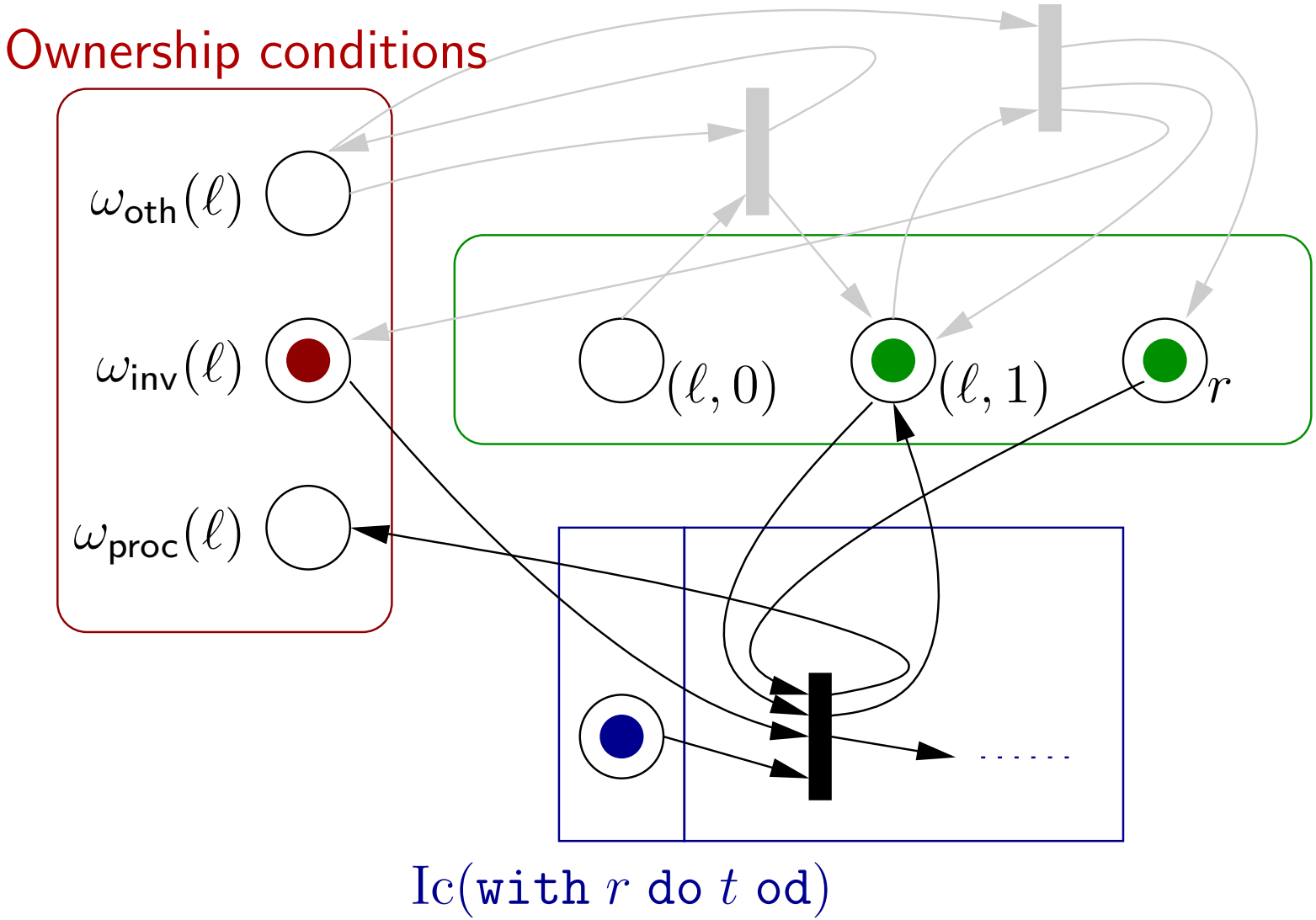
Interference and synchronization



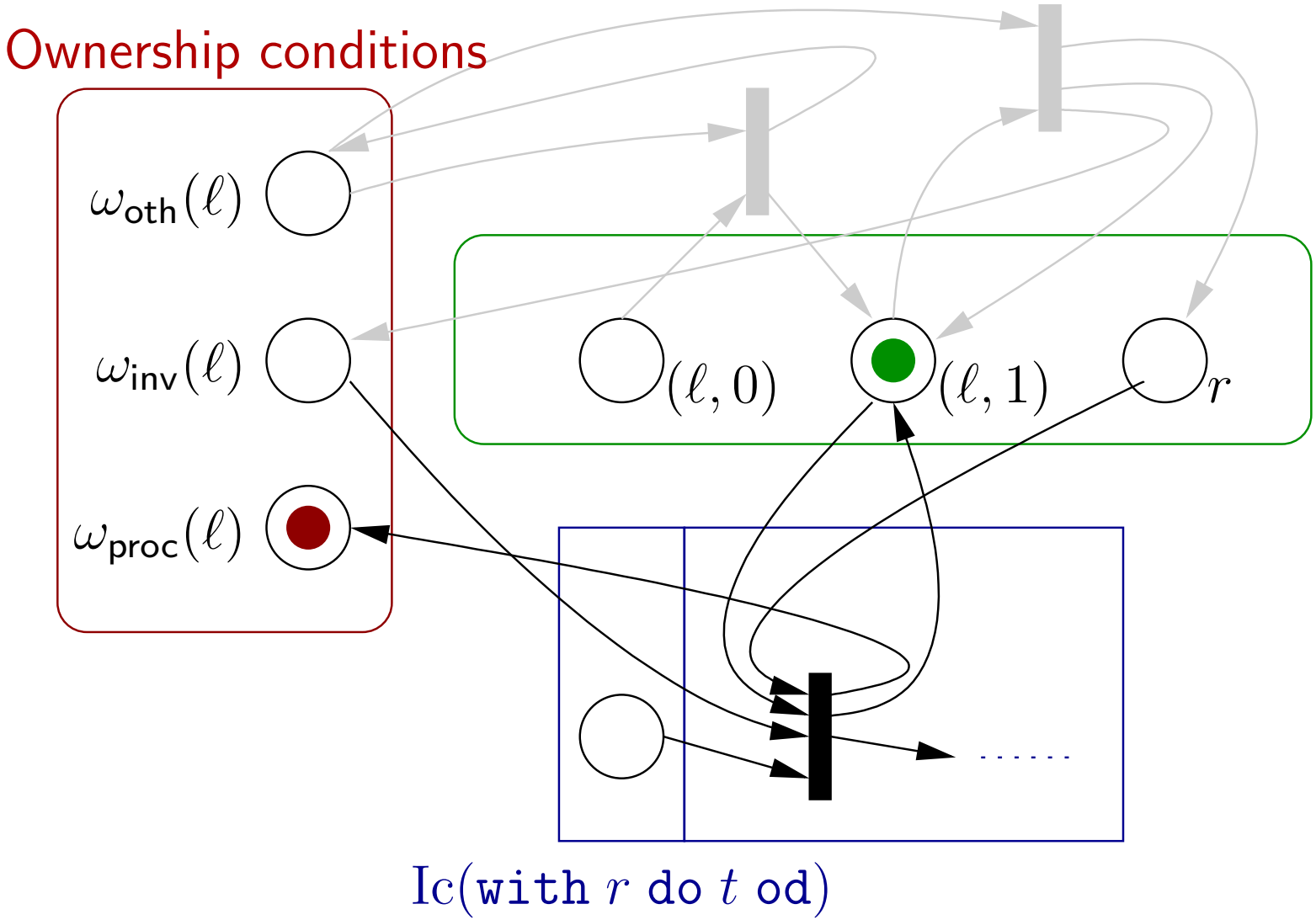
Interference and synchronization



Interference and synchronization



Interference and synchronization



Interference and synchronization

- Add events to the net to represent the execution of other processes
- Possible behaviour of other processes depends on 'ownership' of the process, so add conditions $\omega_{\mathsf{proc}}(\ell)$, $\omega_{\mathsf{oth}}(\ell)$ and $\omega_{\mathsf{inv}}(\ell)$ to the net
- Forms of interference
  - Action on 'other' owned locations
  - Enter and leave critical regions
  - Allocation and deallocation
- 'Synchronize' the original events of the process so that they correspond to interference
- 'Violation' if there is no synchronized event with concession but the un-synchronized event could occur
- Interference net with synchronized events simulates the original net

Validity and soundness

A marking satisfies φ in Γ if the owned locations satisfy φ and the invariants for available resources are met.

Definition (Validity)

$\Gamma \models \{\varphi\}\ t\ \{\psi\}$ if for any initial marking that satisfies φ in Γ, in the net with interference events and synchronized events of t, no violating marking is reachable and any reachable terminal marking satisfies ψ in Γ.

Theorem (Soundness)

$\Gamma \vdash \{\varphi\}\ t\ \{\psi\}$ implies $\Gamma \models \{\varphi\}\ t\ \{\psi\}$.

Independence

Enabled, control independent events arise only from parallel composition

Theorem (Separation)

Suppose that $\emptyset \vdash \{\varphi\}\ t\ \{\psi\}$ and that σ is a state in which the heap satisfies φ. If M is a marking reachable from $(\mathrm{Ic}(t), \sigma)$ in $\mathcal{N}[\![t]\!]$ and e_1 and e_2 are control independent events then:

- If $M \xrightarrow{e_1} M_1$ and $M \xrightarrow{e_2} M_2$ then either e_1 and e_2 are independent or e_1 and e_2 compete to take a resource or to allocate the same location.
- If $M \xrightarrow{e_1} M_1 \xrightarrow{e_2} M'$ then either e_1 and e_2 are independent or e_1 releases a resource that e_2 takes or deallocates a location that e_2 allocates.

Independence

- Permits refinement of heap operations
- ℓ_0 points to ℓ_1, which is initially current

$$
\begin{array}{l}
t_1; \\
\texttt{dealloc}(\ell_0)
\end{array}
\;\Big\|\;
\begin{array}{l}
\texttt{alloc}(k); \\
\texttt{while } ([k] \neq \ell_1) \texttt{ do} \\
\quad \texttt{dealloc}(k); \texttt{alloc}(k) \\
\texttt{od}; \\
t_2
\end{array}
$$

- The loop exit event causally depends on a prior $\texttt{alloc}(k)$ event that allocates the location that was initially stored in ℓ
- These events causally depend on the event $\texttt{dealloc}(\ell)$ in the other process, so t_2 can only run *after* t_1
- $\implies$ interaction through allocation seen in separation theorem

Conclusions

- Independence is a useful tool in describing features of programming languages and their logics
- Future work:
  - Extended models to deal with e.g. name binding
  - Liveness properties and refinement
  - Modal logic using $*$