Outline

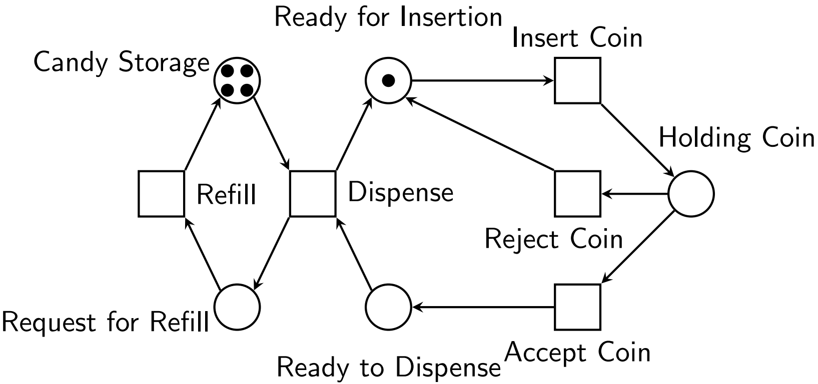
- Petri Nets
- Situation Calculus
- SCOPE (Situation Calculus Ontology of PEtri Nets)

Petri nets: Concepts

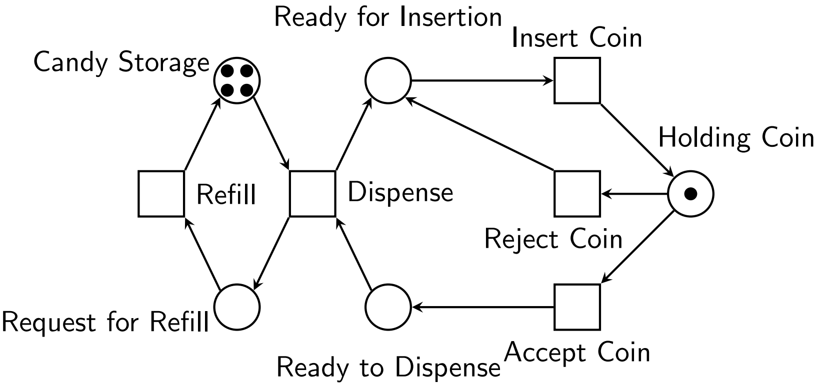
- **Petri net** a triple (P, T, F) such that
  - $-$ P is a finite set of node elements called places (represented by "$\bigcirc$");
  - $-$ T is a finite set of node elements called transitions (represented by "$\Box$");
  - $-$ and $F \subseteq (P \times T) \cup (T \times P)$ consists of ordered pairs ("$\longrightarrow$").
- **Marking** a mapping in the form $M\colon P \to \mathcal{N}$, indicating the assignment of k tokens (a token is represented by a "$\bullet$"), to each place p in P. M_0 is the initial marking.
- **Enabled Transition** A transition t is enabled in a marking M if all input places of t is marked in M.
- **Marking Transition** A marking transition from M to M' due to the firing of enabled t, for each place p, is defined by

$$M'(p) = \begin{cases} M(p) - 1 & \text{if } p \in {}^{b}t \textit{ and } p \notin t^{a} \\ M(p) + 1 & \text{else if } p \notin {}^{b}t \textit{ and } p \in t^{a} \\ M(p) & \text{otherwise.} \end{cases}$$

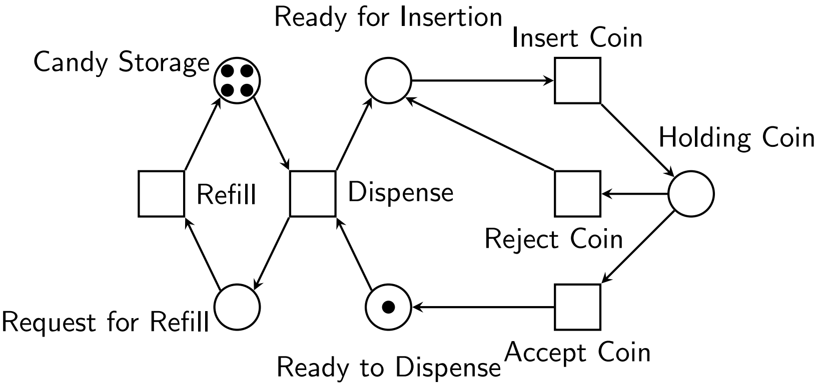
Petri nets: An Example of Vending Machine



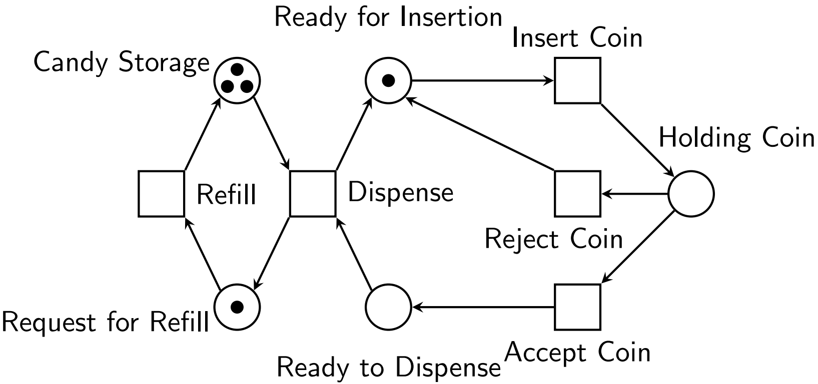
Petri nets: An Example of Vending Machine



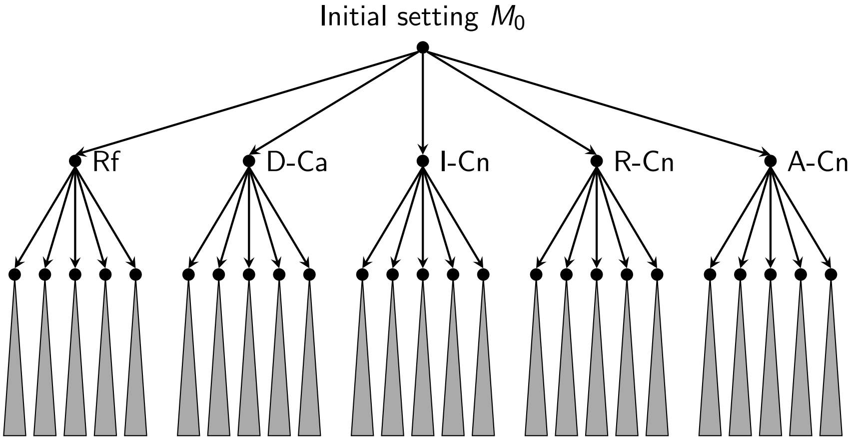
Petri nets: An Example of Vending Machine



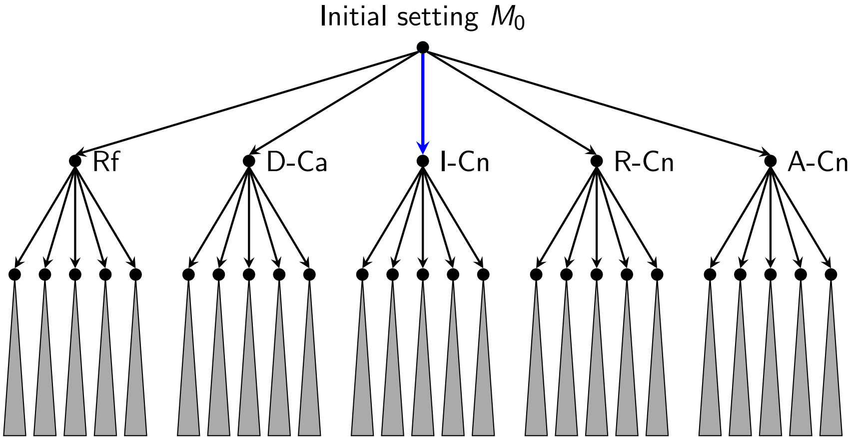
Petri nets: An Example of Vending Machine



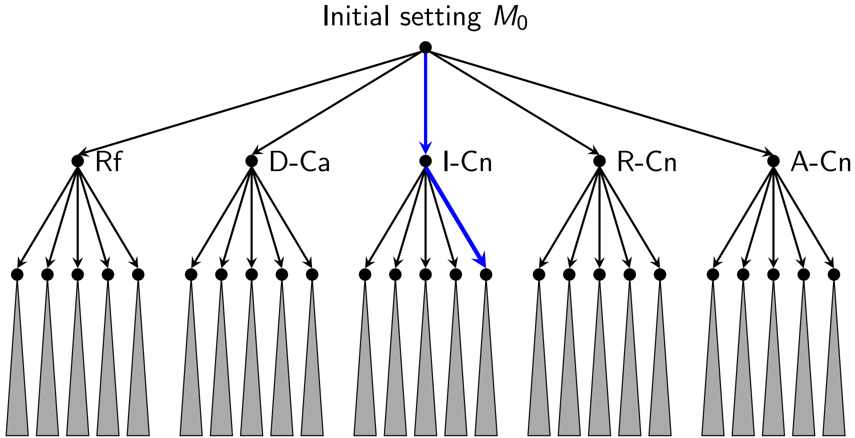
The Marking Transition Tree



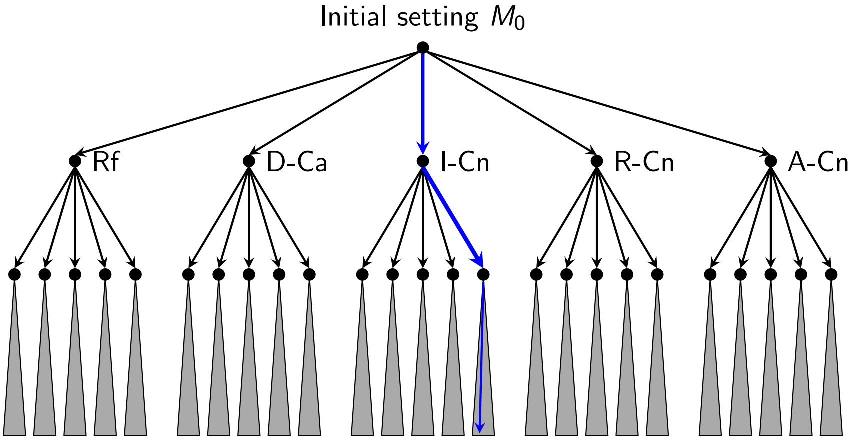
The Marking Transition Tree



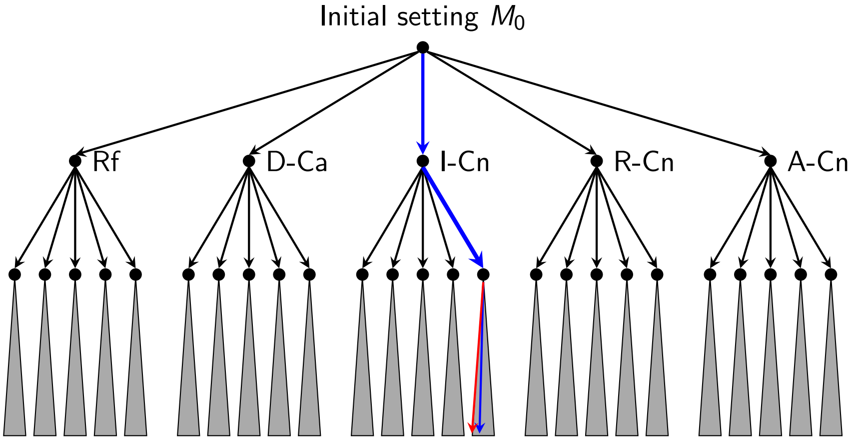
The Marking Transition Tree



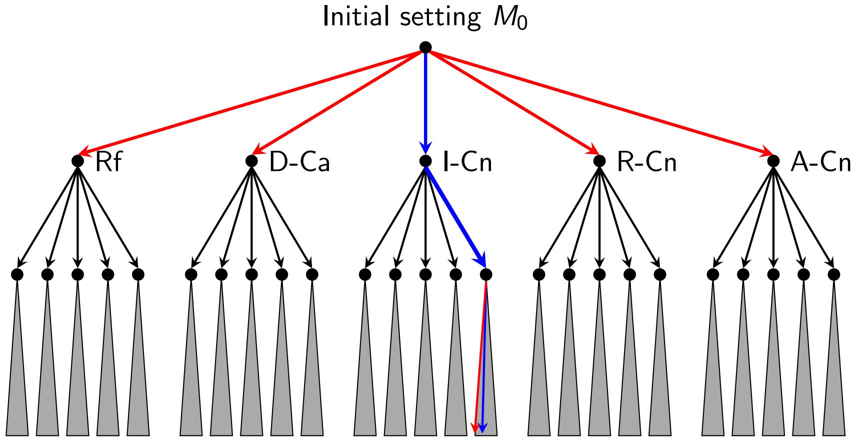
The Marking Transition Tree



The Marking Transition Tree



The Marking Transition Tree



Petri nets: Dynamical Properties

- **Reachability:** Given a Petri net (N, M_0) and a marking M, the problem involves finding an actual sequence of transition firing that leads to M from M_0.
- **k-boundedness:** The number of tokens in each place of any marking M, reachable from M_0, does not exceed the integer k.
- **Liveness:** A Petri net (N, M_0) is live if, for any marking M that can be reached by M_0, there exists another marking M', such that can be reached by M. In other words, at any reachable marking, some transition is enabled to fire.
- **Reversibility:** A Petri net (N, M_0) is reversible if M_0 is reachable from each M that is reachable from M_0.

Petri nets: Remarks

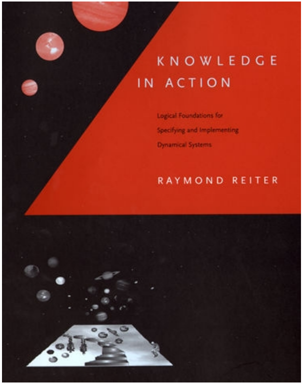
- Petri nets are used to describe dynamical systems. The formalism of Petri nets is simple. However a petri-net system can be complicated.
- As the complexity of a Petri-net system increases, traditional approaches (such as *simulation* and *testing*) are usually unable to explore all behaviors of the system, thus unable to theoretically validate claims on correctness.
- Major merits of ontological approach to Petri-net analysis can be two folds:
  - it enables deductive verification, guarantee on the correctness;
  - eventually it would enable automated deductive verification.

Outline

- Petri Nets
- Situation Calculus $\Leftarrow$
- SCOPE (Situation Calculus Ontology of PEtri Nets)
- TESCOPE
- SCAD
- Remarks

Situation Calculus: the Book

Knowledge In Action: *logical foundations for specifying and implementing dynamical systems*, Ray Reiter, MIT Press, 2001



Situation Calculus: Concepts

- **Situation Calculus:** A logical language for representing changes upon actions in a dynamical domain.
- **Three disjoint sorts**:
  - *action:* actions in the domain, e.g., $rain$, $putdown(x; y)$;
  - *situations:* S_0, the empty sequence of actions; $do(a; s)$, a sequence of actions, e.g., $do(sunshine; do(rain; s))$.
  - *object:* everything else in the specified domain.
- $\sqsubset$ the order between situations, e.g., $s \sqsubset s'$.
- $\mathbf{Poss(a; s)}$: the executability of performing action a in situation s, e.g., $Poss(rain; s) \supset cloudy(s)$.
- **Situation independent relations and functions** e.g., $confLocation(Toronto)$, $area(Room2) = 850$.
- **Relation fluent** e.g., $captain(John; do(catchFever; S_0))$.
- **Function fluent** e.g., $weight(John; do(recover; s)) = 175$.

Situation Calculus: Foundational Axioms

The set $\mathcal{D}_f$ consists of four foundational axioms:

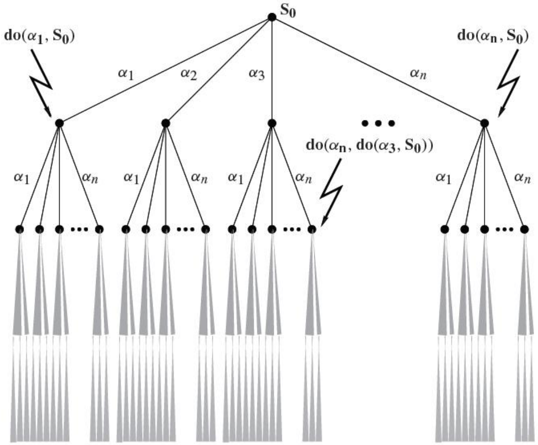
$$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2,$$

$$(\forall P).P(S_0) \wedge (\forall a, s)\big(P(s) \supset P(do(a, s))\big) \supset (\forall s)P(s),$$

$$\neg s \sqsubset S_0,$$

$$s \sqsubset do(a, s') \equiv s \sqsubseteq s'.$$

Situation Calculus: A Model of the Foundational Axioms[1]



[1]Source: Figure 4.1 of "Knowledge in action: logical foundations for specifying and implementing dynamical systems", Ray Reiter, MIT Press, 2001.

Golog (alGOl in LOGic)

- Golog, a logic programming language for description and execution of complex actions using domain-specific Situation Calculus primitive actions as components for these complex actions.
- Additional symbols in Golog (**If, then, else, while**, etc) enable abbreviated expression of complex actions in the system through defining macros on top of the primitive actions in the theory.
- Golog Imperative programming constructs include
  - a, a primitive action;
  - $\alpha; \beta$, action α is followed by action β;
  - $p?$, test action on the condition p;
  - $\alpha|\beta$, nondeterministic choice of action α or action β;
  - $(\pi\, x)\alpha(x)$, nondeterministic choice of arguments;
  - $\alpha^\star$, nondeterministic iteration;
  - Conditionals, while-loops;
  - Procedures.

SCOPE: the ontology

$fire(t)$	the firing of the transition t
$Tkns(p, s)$	the number of tokens of place p at situation s
$pre(p, t)$	the place p is the input of the transition t
$post(p, t)$	the place p is the output of the transition t

SCOPE is defined as a basic action theory $\mathcal{S}$, which consists of several sets of axioms:

$$\mathcal{S} = \mathcal{D}_f \cup \mathcal{S}_{ap} \cup \mathcal{S}_{ss} \cup \mathcal{S}_{una} \cup \mathcal{S}_{S_0}$$

where

- $\mathcal{D}_f$ is the foundational axioms;
- $\mathcal{S}_{ap}$ (The Action Precondition Axiom for the action "fire")
- $\mathcal{S}_{ss}$ (The Successor State Axiom for the fluent "Tkns")
- $\mathcal{S}_{una}$ (The Unique Name Axiom) $fire(t_1) = fire(t_2) \supset (t_1 = t_2)$
- $\mathcal{S}_{S_0}$ (Initial Situation Axioms)

$-$ for each arc from transition t to place p, define $pre(p, t)$; similarly, define $pre(t, p)$;

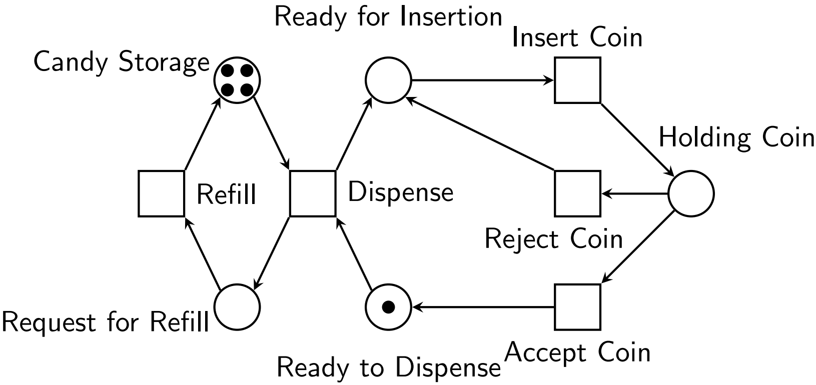
$-$ for each place p with initial marking k, define $Tkns(k, S_0) = k$.

SCOPE: the Action Precondition/Effect Axioms

$$Poss(fire(t), s)) \equiv pre(p, t) \supset Tkns(p, s) \geq 1,$$

$$pre(t, p) \land \neg post(t, p) \supset Tkns(p, do(fire(t), s)) = Tkns(p, s) + 1,$$

$$pre(p, t) \land \neg post(p, t) \supset Tkns(p, do(fire(t), s)) = Tkns(p, s) - 1$$

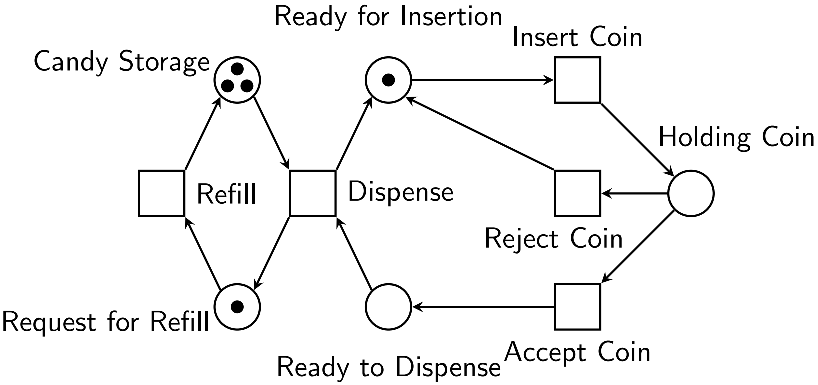


SCOPE: the Action Precondition/Effect Axioms

$$Poss(fire(t), s)) \equiv pre(p, t) \supset Tkns(p, s) \geq 1,$$

$$pre(t, p) \wedge \neg post(t, p) \supset Tkns(p, do(fire(t), s)) = Tkns(p, s) + 1,$$

$$pre(p, t) \wedge \neg post(p, t) \supset Tkns(p, do(fire(t), s)) = Tkns(p, s) - 1$$

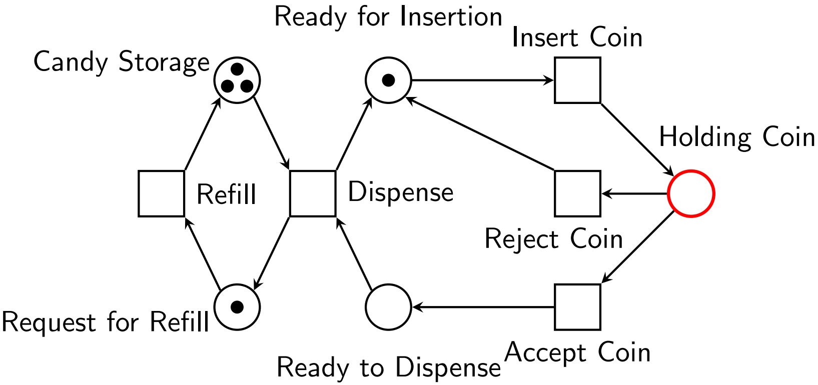


SCOPE: the Action Precondition/Effect Axioms

$$Poss(fire(t), s)) \equiv pre(p, t) \supset Tkns(p, s) \geq 1,$$

$$pre(t, p) \wedge \neg post(t, p) \supset Tkns(p, do(fire(t), s)) = Tkns(p, s) + 1,$$

$$pre(p, t) \wedge \neg post(p, t) \supset Tkns(p, do(fire(t), s)) = Tkns(p, s) - 1$$



SCOPE: The Successor State Axiom

$$Tkns(p, do(a, s)) = n \equiv$$
$$\gamma_f(p, t, n, a, s) \vee (Tkns(p, s) = n \wedge \neg(\exists n')\, \gamma_f(p, t, n', a, s)),$$

where

$$\gamma_f(p, t, n, a, s) \stackrel{def}{=} \gamma_{f_a}(p, t, n, a, s) \vee \gamma_{f_b}(p, t, n, a, s)$$

are simply abbreviations:

- $\gamma_{f_a}(p, t, n, a, s)$ is defined as

  $$(\exists t).pre(t, p) \wedge \neg post(t, p) \wedge n = Tkns(p, s) + 1 \wedge a = fire(t)$$

- $\gamma_{f_b}(p, t, n, a, s)$ is defined as

  $$(\exists t).pre(p, t) \wedge \neg post(p, t) \wedge n = Tkns(p, s) - 1 \wedge a = fire(t)$$

SCOPE: Subclasses

- **abbreviation**

$$exec(s) \stackrel{def}{=} (\forall a, s^*).do(a, s^*) \sqsubseteq s \supset Poss(a, s^*)$$

- **Reachability** Given a marking M_n, we have

$$(\exists s).\ exec(s) \wedge Tkns(p_0, s) = n_0 \wedge \ldots Tkns(p_i, s) = n_i,$$

  where n_i is the number of tokens at the place p_i in M_n.
- **K-Boundedness**

$$exec(s) \supset Tkns(p, s) \leq k$$

- **Liveness**

$$exec(s) \supset (\exists s_1).\ s \sqsubset s_1 \wedge exec(s_1)$$

- **Reversibility**

$$exec(s) \supset (\exists s_1).\ s \sqsubset s_1 \wedge exec(s_1) \wedge Tkns(p, s_1) = Tkns(p, S_0)$$

SCOPE Example Procedures

The procedure *handleCurrentCoin* is defined w.r.t. *insertOneCoin*:

proc *insertOneCoin*
  fire(*ICn*)
endProc
proc *handleCurrentCoin*
  insert(*ICn*) ; *fire*(*ACn*) | *fire*(*RCn*)
endProc

2) Keep inserting a coin until one is accepted to the machine:

proc *insertCoinRepeatedly*
  while $Tkns(RdyInsert) = 1$ **do**
    handleCurrentCoin
  endWhile
endProc