# Internet Computer Policy Formulas

This document describes all the policy formulas that we formalized for the Internet Computer (IC).

## 1 Common Fragments

We start with the formula $\texttt{InIC}(n, a)$, denoting that a node $n$ with an IP address $a$ is currently active in the IC. This formula does not represent a standalone policy. Instead, it is used as a building block for some of the policy formulas shown below. It is formalized as follows:

$$((\blacklozenge\, \texttt{OriginallyInIC}(n, a)) \land \neg \blacklozenge\, \texttt{RegistryRemoveNode}(n, a)) \lor \\ (\neg \texttt{RegistryRemoveNode}(n, a)\ \mathsf{S}\ \texttt{RegistryAddNode}(n, a))$$

The formula contains the input predicates $\texttt{OriginallyInIC}(n, a)$ (node $n$ with IP address $a$ belongs to the IC when monitoring originally started), $\texttt{RegistryAddNode}(n, a)$ (node $n$ with IP address $a$ is added to the IC), and $\texttt{RegistryRemoveNode}(n, a)$ (node $n$ with IP address $a$ is removed from the IC), each corresponding to events (log entries) in the trace. The predicates prefixed with `Registry` correspond to log entries from the IC registry app, which maintains the IC's configuration. In contrast, the events $\texttt{OriginallyInIC}(n, a)$ and $\texttt{OriginallyInSubnet}(n, a, s)$ (used below) are provided by the preprocessor which queries the IC registry before every run of the monitoring pipeline. The preprocessor also adds the IP address to all the events corresponding the above input predicates.

The formula $\texttt{InSubnet}(n, a, s)$, denoting that a node $n$ with an IP address $a$ is part of a subnet $s$, is defined similarly:

$$((\blacklozenge\, \texttt{OriginallyInSubnet}(n, a, s)) \land \neg \blacklozenge\, \texttt{RegistryRemoveNodeFrom}(n, a, s)) \lor \\ (\neg \texttt{RegistryRemoveNodeFrom}(n, a, s)\ \mathsf{S}\ \texttt{RegistryAddNodeTo}(n, a, s))$$

In the following policy formulas, we define custom predicates, such as `InIC` and `InSubnet`, using the **let** operator. In general, we note that without the **let** operator, some of our policies would have required large policy formulas that are hard for humans to get right or understand.

When using `InIC` and `InSubnet`, we typically ignore the IP address parameter using the wildcard symbol (_). Wildcards are syntactic sugar for an existential quantifier, i.e., $\texttt{P}(x, \_)$ is equivalent to $\exists y.\ \texttt{P}(x, y)$.

## 2 clean-logs

The `clean-logs` policy states that only *warning*- and *info*-level log entries are allowed, whereas *critical*- or *error*-level entries are not. It uses the input predicate

$\texttt{Log}(h, n, s, c, l, m)$, which is satisfied by every log message $m$ emitted by component $c$ running on node $n$ in subnet $s$ with host name $h$, where $l$ is the log level.

$$
\begin{array}{l}
\textbf{let } \texttt{InIC}(n, a) \coloneqq \cdots \textbf{ in} \\
\textbf{let } \texttt{ErrorLevel}(l) \coloneqq (l = \texttt{"CRITICAL"}) \lor (l = \texttt{"ERROR"}) \textbf{ in} \\
\texttt{InIC}(n, \_) \land \texttt{Log}(h, n, s, c, l, m) \land \texttt{ErrorLevel}(l)
\end{array}
$$

## 3 finalized-height

We next show the `finalized-height` policy formula:

$$
\begin{array}{l}
\textbf{let } \texttt{InSubnet}(n, a, s) \coloneqq \cdots \textbf{ in} \\
\textbf{let } \texttt{Growing}(s) \coloneqq \exists n_1, n_2.\ \texttt{InSubnet}(n_1, \_, s) \land \texttt{InSubnet}(n_2, \_, s) \land n_1 \neq n_2 \land \\
\qquad\qquad\qquad\qquad\qquad \neg(\neg\texttt{p2pRemoveNode}(n_1, s, n_2)\ \mathsf{S}\ \texttt{p2pAddNode}(n_1, s, n_2)) \textbf{ in} \\
\textbf{let } \texttt{Shrinking}(s) \coloneqq \exists n_1, n_2.\ \texttt{InSubnet}(n_1, \_, s) \land \\
\qquad\quad \big(\ (\neg\texttt{p2pRemoveNode}(n_1, s, n_2)\ \mathsf{S}\ \texttt{p2pAddNode}(n_1, s, n_2)) \lor \\
\qquad\qquad \texttt{InSubnet}(n_1, \_, s) \land (\blacklozenge\, \texttt{OriginallyInSubnet}(n_2, \_, s)) \land \\
\qquad\qquad \neg(\blacklozenge\, \texttt{p2pRemoveNode}(n_1, s, n_2)) \land \neg(\blacklozenge\, \texttt{p2pAddNode}(n_1, \_, n_2))\big) \land \\
\qquad\quad \neg\texttt{InSubnet}(n_2, \_, s) \textbf{ in} \\
\textbf{let } \texttt{Changing}(s) \coloneqq \texttt{Growing}(s) \lor \texttt{Shrinking}(s) \textbf{ in} \\
\textbf{let } \texttt{FirstFin}(n, s, h, b, v) \coloneqq \texttt{Finalized}(n, s, h, b, v) \land \neg \bullet \blacklozenge\, \exists n'.\texttt{Finalized}(n', s, h, b, v) \textbf{ in} \\
(\neg\texttt{Changing}(s)\ \mathsf{S}_{(80\text{s}, \infty)}\ \texttt{FirstFin}(n_1, s, h_1, b_1, v)) \land \texttt{FirstFin}(n_2, s, h_2, b_2, v) \land h_2 = h_1 + 1
\end{array}
$$

The policy checks that the block at height $h + 1$ in a subnet is finalized by some node no later than 80 seconds after the *earliest* finalization of the block at height $h$. The 80 seconds is a rather conservative upper bound as, in practice, the average time between two finalized blocks is around 1 second. The `finalized-height` policy relies on input predicates `p2pAddNode` and `p2pRemoveNode`. The corresponding events are obtained from the peer-to-peer communication layer and represent the nodes' own view on subnet membership. The $\texttt{Finalized}(n, s, h, b, v)$ event occurs whenever the consensus layer tries to deliver a batch to the message routing layer for the first time. The batch includes the block $b$ at height $h$ by some node $n$ in subnet $s$, running IC software version $v$.

## 4 finalization-consistency

We next show the `finalization-consistency` policy formula:

$$
\begin{array}{l}
\textbf{let } \texttt{InIC}(n, a) \coloneqq \cdots \textbf{ in} \\
(\blacklozenge\, \texttt{Finalized}(n_1, s, h, h_1, v)) \land \texttt{Finalized}(n_2, s, h, h_2, v)) \land h_1 \neq h_2 \land \\
\texttt{InIC}(n_1, \_) \land \texttt{InIC}(n_2, \_)
\end{array}
$$

It formalizes that whenever a node finalizes a block at a given height, no other node in the same subnet finalizes a different block at the same height.

## 5 unauthorized-connections

We next show the `unauthorized-connections` policy formula:

$$\begin{array}{l}\textbf{let}\ \texttt{InSubnet}(n,a,s) \coloneqq \cdots\ \textbf{in}\\ \texttt{TLSHandshakeFail}(da,sa) \land \texttt{InSubnet}(did,da,s) \land\\ \quad \neg((\exists s'.\ \texttt{InSubnet}(\_,sa,s') \land \texttt{InSubnet}(did,da,s) \land s \neq s') \land\\ \qquad \blacklozenge_{[0,15\mathsf{min}]}\ \texttt{InSubnet}(\_,sa,s))\end{array}$$

It states that no node participating in a subnet, logs unauthorized TLS connection attempts, except for connection attempts between nodes that were recently (up to 15 min ago) members of the same subnet. The `finalized-height` policy relies on input predicate $\texttt{TLSHandshakeFail}(da, sa)$ with the corresponding event signifies a failed TLS handshake between $sa$ and $da$ source and destination IP addresses.

## 6 replica-divergence

We now show the `replica-divergence` policy formula:

$$\begin{array}{l}\textbf{let}\ \texttt{RelevantNode}(n,s) \coloneqq \texttt{OriginallyInSubnet}(n,\_,s) \lor \texttt{RegistryAddNodeTo}(n,\_,s) \lor\\ \qquad\qquad\qquad\qquad\qquad \texttt{p2pAddNode}(\_,s,n)\ \textbf{in}\\ \texttt{EndTest}() \land \texttt{RelevantNode}(n,s) \land ((\neg\texttt{CupShareProposed}(n,s))\ \mathsf{S}\ \texttt{ReplicaDiverged}(n,s,\_))\end{array}$$

The intended consensus protocol property captured by `replica-divergence` is that diverged nodes must eventually propose a catch-up package share. It relies on input predicate $\texttt{CupShareProposed}(a, s)$ that holds when a catch-up package share is proposed by node $a$ of subnet $s$; input predicate $\texttt{diverged}(a, s)$ indicating that $a$ has reported a state divergence; and `end` which matching the corresponding last event in the trace (added by the preprocessor). Using this event and the fact that traces are finite in practice, we monitor this policy as a safety property.

## 7 reboot-count

Next we show the `reboot-count` policy formula:

$$\begin{array}{l}\textbf{let}\ \texttt{InIC}(n,a) \coloneqq \cdots\ \textbf{in}\\ \textbf{let}\ \texttt{TrueReboot}(ip,dc) \coloneqq \texttt{InIC}(\_,ip) \land \texttt{Reboot}(ip,dc) \land \CIRCLE\blacklozenge\,\texttt{Reboot}(ip,dc)\ \textbf{in}\\ \textbf{let}\ \texttt{UnintendedReboot}(ip,dc) \coloneqq \texttt{TrueReboot}(ip,dc) \land\\ \qquad\qquad\qquad\qquad\qquad\qquad \neg\CIRCLE(\neg\texttt{Reboot}(ip,dc)\ \mathsf{S}\ \texttt{RebootIntent}(ip,dc))\ \textbf{in}\\ \textbf{let}\ \texttt{NumReboots}(dc,n) \coloneqq n \leftarrow \text{CNT}\ i; dc\ \ \blacklozenge_{[0,30\mathsf{min}]}\ \texttt{UnintendedReboot}(ip,dc) \land \texttt{tp}(i)\ \textbf{in}\\ \texttt{NumReboots}(dc,n) \land n > 2\end{array}$$

The policy is satisfied whenever there are more than 2 (unintended) reboots of IC hosts in the same data center within a 30 minute window. The first reboot is ignored, as every machine is (re)booted when a testnet starts. Event $\texttt{Reboot}(ip, dc)$ indicates that machine with IP address $ip$ in the datacenter $dc$ has been rebooted, whereas event $\texttt{RebootIntent}(ip, dc)$ the machine is about to perform a reboot requested by an administrator. Besides the new input predicates, this formula also uses a input predicate $\texttt{tp}(i)$ supported by MonPoly, such that given a trace $\rho$, valuation $v$, and index $i$, $v, i \models_\rho \texttt{tp}(t)$ if $v(t) = i$. In other words $\texttt{tp}(i)$ can be used to bind the current trace index to $i$.

**block-validation-latency** The `block-validation-latency` policy formula is defined as follows:

$$
\begin{aligned}
&\textbf{let } \texttt{InSubnet}(n, a, s) \coloneqq \cdots \textbf{ in}\\
&\textbf{let } \texttt{SubnetMap}(n, s) \coloneqq \texttt{InSubnet}(n, \_, s) \textbf{ in}\\
&\textbf{let } \texttt{SubnetSize}(s, ns) \coloneqq ns \leftarrow \text{CNT } n; s\ \texttt{SubnetMap}(n, s) \textbf{ in}\\
&\textbf{let } \texttt{BlockAdded}(n, s, b, t) \coloneqq \Diamond_{[0,0]}\ \texttt{validatedBPAdded}(n, s, b) \land\\
&\texttt{SubnetMap}(n, s) \land \texttt{ts}(t) \textbf{ in}\\
&\textbf{let } \texttt{Validated}(b, s, t) \coloneqq \exists nv, n.\ (nv \leftarrow \text{CNT } vn; b, s, t\\
&\qquad\qquad \blacklozenge\, \texttt{BlockAdded}(vn, s, b, t) \lor\\
&\qquad\qquad\qquad\qquad (\exists an.\ (\blacklozenge\, \texttt{BlockAdded}(an, s, b, t)) \land\\
&\qquad\qquad\qquad\qquad\qquad\quad \texttt{validatedBPMoved}(vn, s, b) \land\\
&\qquad\qquad\qquad\qquad\qquad\quad \texttt{SubnetMap}(vn, s))) \land\\
&\qquad\quad \texttt{SubnetSize}(s, ns) \land nv > 2 * ns/3 \textbf{ in}\\
&\textbf{let } \texttt{TimePerBlock}(b, s, t) \coloneqq \exists ta, tv.\ \texttt{Validated}(b, s, ta) \land (\neg \bullet \blacklozenge\, \texttt{Validated}(b, s, t)) \land\\
&\qquad\qquad\qquad\qquad\qquad\quad \texttt{ts}(tv) \land t = tv - ta \textbf{ in}\\
&\textbf{let } \texttt{SubnetType}(si, st) \coloneqq \texttt{SubnetType}_0(si, st) \lor \texttt{RegistryCreateSubnet}(si, st) \lor\\
&\qquad\qquad\qquad\qquad\qquad \texttt{RegistryUpdateSubnet}(si, st) \textbf{ in}\\
&\textbf{let } \texttt{SubnetMap}(si, st) \coloneqq \neg\texttt{SubnetType}(si, st)\ \mathsf{S}\ \texttt{SubnetType}(si, st) \textbf{ in}\\
&(\texttt{TimePerBlock}(b, s, t) \land \texttt{SubnetMap}(s, \texttt{"System"}) \land t > 3\mathsf{s}) \lor\\
&(\texttt{TimePerBlock}(b, s, t) \land\\
&\qquad (\texttt{SubnetMap}(s, \texttt{"Application"}) \lor \texttt{SubnetMap}(s, \texttt{"VerifiedApplication"})) \land\\
&\qquad t > 3\mathsf{s})
\end{aligned}
$$

The `block-validation-latency` policy formula states that the time difference between the moment when a block proposal has been added to the validated pool by some node and the moment when more than two thirds of the nodes in the same subnet has done so too must not exceed 3 seconds. Input predicate $\texttt{SubnetType}_0(si, st)$ indicates that subnet with ID $si$ and type $st$ exists at the start of monitoring. Input predicate $\texttt{RegistryCreateSubnet}(si, st)$ indicates that subnet with ID $si$ has been created and it has type $st$. Input predicate $\texttt{RegistryUpdateSubnet}(si, st)$ indicates that subnet with ID $si$ has been modified to have type $st$. This formula also uses a special input predicate $\texttt{ts}(i)$ supported by MonPoly. Given a trace $\rho$, valuation $v$, and index $i$, $v, i \models_\rho \texttt{ts}(t)$ if $v(t) = \tau_i$. In other words $\texttt{ts}(i)$ can be used to bind the current timestamp to $i$.

## 8 logging-behavior

Finally, we show the `logging-behavior` policy formula:

```
$\textbf{let}\ \texttt{NodeAdded}(n,s) \coloneqq \texttt{OriginallyInSubnet}(n,\_,s) \lor \texttt{RegistryAddNodeTo}(n,\_,s)\ \textbf{in}$
$\textbf{let}\ \texttt{NodeRemoved}(n,s) \coloneqq \texttt{RegistryAddNodeTo}(n,\_,s)\ \textbf{in}$
$\textbf{let}\ \texttt{InSubnet}(n,s) \coloneqq \neg\texttt{NodeRemoved}(n,s)\ \mathsf{S}\ \texttt{NodeAdded}(n,s)\ \textbf{in}$
$\textbf{let}\ \texttt{ProperTP}() \coloneqq \blacklozenge_{[1,\infty]}\ \top\ \textbf{in}$
$\textbf{let}\ \texttt{RelevantNode}(n,s) \coloneqq \texttt{InSubnet}(n,s)\ \mathsf{S}_{\mathsf{10min},\infty}\ \texttt{ProperTP}()\ \textbf{in}$
$\textbf{let}\ \texttt{RelevantLog}(n,s,l,m,i) \coloneqq \exists h,c.\ \texttt{Log}(h,n,s,c,l,m) \land$
        $c \overset{\mathsf{RE}}{\Leftarrow} \texttt{"orchestrator"} \land$
        $n = \texttt{""} \land \texttt{tp}(i)\ \textbf{in}$
$\textbf{let}\ \texttt{MsgCount}(n,s,c) \coloneqq c \leftarrow \mathrm{SUM}\ c';n,s\ (($
        $(c' \leftarrow \mathrm{CNT}i;n,s\ \blacklozenge_{[0,\mathsf{10min}]}\ \texttt{RelevantLog}(n,s,l,m,i)) \land$
        $\texttt{RelevantNode}(n,s)) \lor \texttt{RelevantNode}(n,s) \land c = 0)\ \textbf{in}$
$\textbf{let}\ \texttt{TypicalBehavior}(s,m) \coloneqq (m \leftarrow \mathrm{MED}\ c;s\ \texttt{MsgCount}(n,s,c)) \land$
        $\exists n'.\ (n' \leftarrow \mathrm{CNT}\ n;s\ \texttt{RelevantNode}(n,s)) \land n' \geq 3\ \textbf{in}$
$\textbf{let}\ \texttt{TypicalBehaviors}(s,m) \coloneqq \blacklozenge_{[0,\mathsf{10min}]}\ \texttt{TypicalBehavior}(s,m) \lor$
        $\Diamond_{[0,\mathsf{10min}]}\ \texttt{TypicalBehavior}(s,m)\ \textbf{in}$
$\textbf{let}\ \texttt{Compute}(s,n,c,mn,mx) \coloneqq (\neg\Diamond_{[0,\mathsf{10min}]}\ \texttt{EndTest}) \land$
        $(\exists c'.\ \texttt{MsgCount}(n,s,c') \land c = c') \land$
        $(mn \leftarrow \mathrm{MIN}\ m;s\ \texttt{TypicalBehaviors}(s,m)) \land$
        $(mn \leftarrow \mathrm{MAX}\ m;s\ \texttt{TypicalBehaviors}(s,m))\ \textbf{in}$
$\textbf{let}\ \texttt{Exceeds}(s,n,c,mn,mx) \coloneqq (\texttt{Compute}(s,n,c,mn,mx) \land c > mx * 1.1) \lor$
        $(\texttt{Compute}(s,n,c,mn,mx) \land c < mn * 0.9)\ \textbf{in}$
$\texttt{Exceeds}(s,n,c,mn,mx) \land \neg \bullet_{[0,\mathsf{10min}]}(\exists a,b,c.\ \texttt{Exceeds}(s,n,a,b,c))$
```

For each subnet, the `logging-behavior` policy compares its nodes' logging frequencies computed over a sliding window against the median logging frequency over all nodes in the subnet. Intuitively formula $x \overset{\mathsf{RE}}{\Leftarrow} r$ is satisfied for all strings $x$ that regular expression $r$ successfully matches.