# A Semantics for Network-Adaptive Execution of Distributed Objects

Karl Palmskog
palmskog@kth.se
KTH Royal Institute of Technology
Joint work with Mads Dam

2011-11-15

# Introduction

*"ABS [is] an abstract behavioral specification language for designing executable models of distributed object-oriented systems."*

```
class Node implements Peer {
  File getFile(Server sId, Filename fId) {
    Fut<Int> l1 = sId!getLength();
    await l1?;
    Int lth = l1.get;
    File file = new cog FileImpl(lth, fId);
    return file;
  }
}
```

# Properties of ABS

- Unit of concurrency: concurrent object group (cog)
- Cooperative scheduling of tasks inside a cog
- Asynchronous method calls translate to message passing
- Result of method calls are obtained by resolving futures
- Communication graph can be seen as a full mesh

# ABS Object Level Syntax

| | | | |
|---|---|---|---|
| $P$ | $::=$ | | program |
| | $\mid$ | $\overline{Dd}\ \overline{F}\ \overline{IF}\ \overline{CL}\{\overline{T}\ \overline{x};s;\}$ | |
| $IF$ | $::=$ | | interface definition |
| | $\mid$ | $\textbf{interface}\ I\{\overline{Sg}\}$ | |
| $CL$ | $::=$ | | class definition |
| | $\mid$ | $\textbf{class}\ C[(\overline{T}\ \overline{x})]\ [\textbf{implements}\ \overline{I}]\{\overline{T}\ \overline{x}';\overline{M}\}$ | |
| $M$ | $::=$ | | method definition |
| | $\mid$ | $Sg\{\overline{T}\ \overline{x};s;\ \textbf{return}\ e;\}$ | |
| $Sg$ | $::=$ | | method signature |
| | $\mid$ | $T\ m(\overline{T}\ \overline{x})$ | |
| $g$ | $::=$ | | guard |
| | $\mid$ | $b$ | |
| | $\mid$ | $e?$ | |
| | $\mid$ | $g \wedge g'$ | |

# ABS Object Level Syntax, continued

| | | | |
|---|---|---|---|
| $s$ | $::=$ | | statement |
| | $\mid$ | $s; s'$ | composition |
| | $\mid$ | $x = rhs$ | assignment |
| | $\mid$ | **suspend** | suspend |
| | $\mid$ | **await** $g$ | await guard |
| | $\mid$ | **skip** | skip |
| | $\mid$ | **if** $b\{s\}$ | if |
| | $\mid$ | **if** $b\{s\}$ **else** $\{s'\}$ | if else |
| | $\mid$ | **while** $b\{s\}$ | while loop |
| $rhs$ | $::=$ | | assignment right-hand side |
| | $\mid$ | $e$ | expression |
| | $\mid$ | **new** $C(\overline{e})$ | new object |
| | $\mid$ | **new cog** $C(\overline{e})$ | new object and cog |
| | $\mid$ | $e!m(\overline{e})$ | asynchronous call |
| | $\mid$ | $e.m(\overline{e})$ | synchronous call |
| | $\mid$ | $e.$ **get** | future value |

# ABS Runtime Syntax

| | | | |
|---|---|---|---|
| $cn$ | $::=$ | | configuration |
| | $\mid$ | $\epsilon$ | |
| | $\mid$ | $fut$ | |
| | $\mid$ | $object$ | |
| | $\mid$ | $invoc$ | |
| | $\mid$ | $cog$ | |
| | $\mid$ | $cn\ cn'$ | |
| $cog$ | $::=$ | | concurrent object group |
| | $\mid$ | $\mathrm{cog}\,(c, act)$ | |
| $fut$ | $::=$ | | future |
| | $\mid$ | $\mathrm{fut}\,(f, val)$ | |
| $val$ | $::=$ | | value |
| | $\mid$ | $v$ | |
| | $\mid$ | $\perp$ | |
| $object$ | $::=$ | | object |
| | $\mid$ | $\mathrm{ob}\,(o, a, p, q)$ | |

# ABS Runtime Syntax, continued

| | | | |
|---|---|---|---|
| $a, \; l$ | $::=$ | | substitution |
| | $\mid$ | $T\ x\ v$ | |
| | $\mid$ | $a, a'$ | |
| *process* | $::=$ | | process |
| | $\mid$ | $\{a \mid s_p\}$ | |
| | $\mid$ | **error** | |
| $p$ | $::=$ | | active process |
| | $\mid$ | *process* | |
| | $\mid$ | **idle** | |
| $q$ | $::=$ | | pool of suspended processes |
| | $\mid$ | $\emptyset$ | |
| | $\mid$ | *process* | |
| | $\mid$ | $q\ q'$ | |

# ABS Runtime Syntax, continued

| | | | |
|---|---|---|---|
| $v$ | $::=$ | | value |
| | $\mid$ | $o$ | |
| | $\mid$ | $c$ | |
| | $\mid$ | $f$ | |
| | $\mid$ | $t$ | |
| $invoc$ | $::=$ | | method invocation |
| | $\mid$ | $\text{invoc}\,(o, f, m, \overline{v})$ | |
| $act$ | $::=$ | | active object |
| | $\mid$ | $o$ | |
| | $\mid$ | $\varepsilon$ | |
| $s_p$ | $::=$ | | process statement |
| | $\mid$ | $\textbf{return}\ e$ | return |
| | $\mid$ | $\textbf{cont}\,(f)$ | continue other process |
| | $\mid$ | $s$ | statement |
| | $\mid$ | $s_p; s'_p$ | composition |
| | $\mid$ | $\epsilon$ | empty string |

# Start Configurations

$$\frac{}{\begin{array}{c}\text{start}(\overline{Dd}\ \overline{F}\ \overline{IF}\ \overline{CL}\{\overline{T}\ \overline{x}; s; \}) = \\ \mathsf{cog}\,(\text{main}, \text{start})\ \mathsf{ob}\,(\text{start}, \mathsf{cog}\ \mathbf{cog}\ \text{main}, \{\overline{T}\ \overline{x}\ \text{atts}\,(\overline{T})|s; \epsilon\}, \emptyset)\end{array}}\ \text{STAR}$$

# A Sample of ABS Operational Semantics

$$\frac{\begin{array}{l}\mathrm{fresh}\left(o'\right)\\ \mathrm{fresh}\left(c'\right)\\ \mathrm{init}\left(C\right) = p\\ \mathrm{atts}\left(C, [\![\overline{e}]\!]_{a \circ l}, o', c'\right) = a'\end{array}}{\begin{array}{c}\mathsf{ob}\left(o, a, \{l | x = \textbf{new cog } C(\overline{e}); s_p\}, q\right)\\ \rightarrow \mathsf{ob}\left(o, a, \{l | x = o'; s_p\}, q\right)\ \mathsf{ob}\left(o', a', p, \emptyset\right)\ \mathsf{cog}\left(c', o'\right)\end{array}} \quad \text{NEW\_COG\_OB}$$

# A Sample of ABS Operational Semantics, continued

$$\frac{\begin{array}{c}[\![e]\!]_{a \circ l} = o' \\ [\![\overline{e}]\!]_{a \circ l} = \overline{v} \\ \mathrm{fresh}\,(f)\end{array}}{\begin{array}{c}\mathsf{ob}\,(o, a, \{l|x = e!m(\overline{e}); s_p\}, q) \\ \to \mathsf{ob}\,(o, a, \{l|x = f; s_p\}, q)\ \mathsf{invoc}\,(o', f, m, \overline{v})\ \mathsf{fut}\,(f, \bot)\end{array}}\ \text{ASYNC\_CALL}$$

# The ABS-NET Semantics

Work enhances (Core) ABS semantics with

- nodes
- arcs
- message routing

When implemented, enables ABS programs to execute

- concurrently across a network
- adaptively based on resource availability

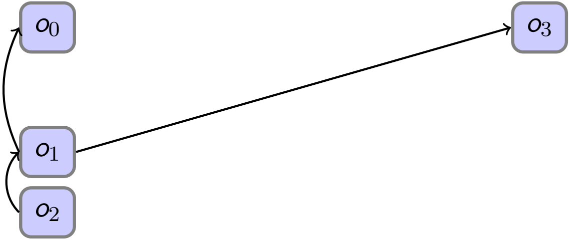
# Goals

To run efficiently run ABS programs in a network, we want:

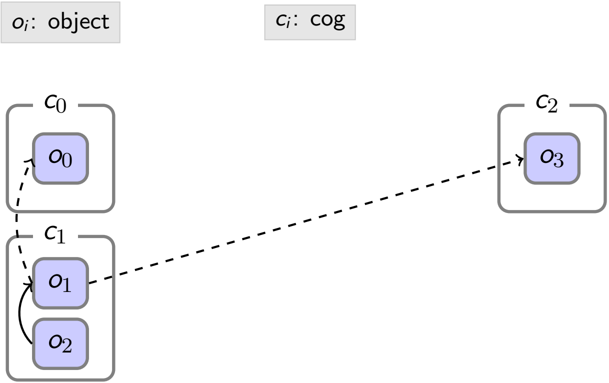
- Migration of cogs and objects between nodes
- Location-independent object addressing
- Transparent handling of method calls
- Preservation of Core ABS behaviour

# Objects

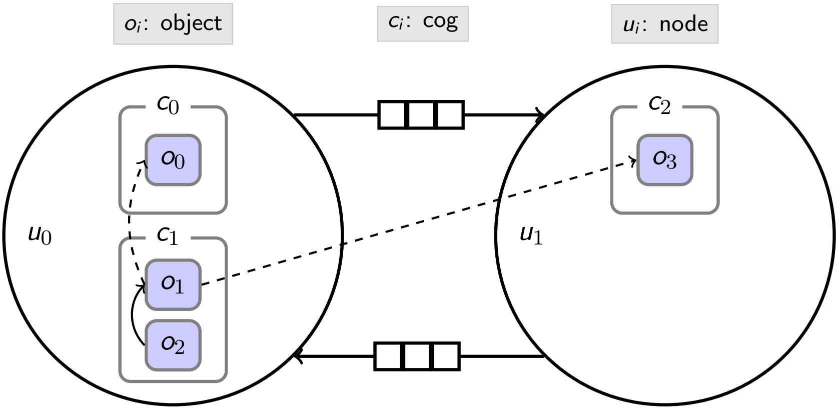
$o_i$: object



# Objects Inside Cogs



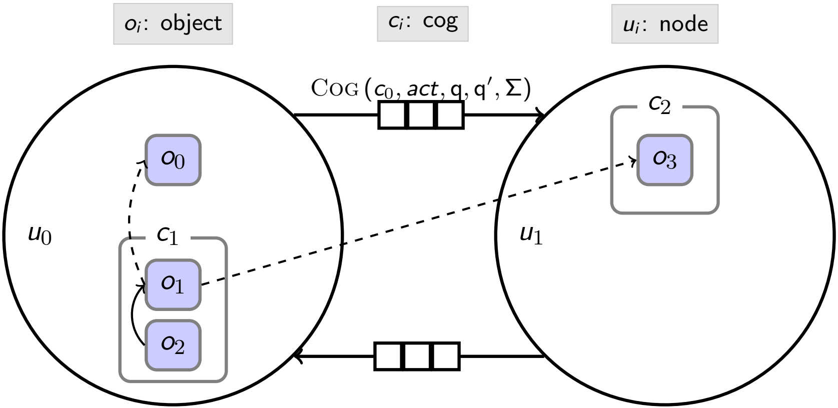
# Cogs Inside Nodes




$t_0 : c_0, c_1 \mapsto \langle u_0, 0 \rangle,$
$c_2 \mapsto \langle u_1, 1 \rangle$

$t_1 : c_0, c_1 \mapsto \langle u_0, 1 \rangle,$
$c_2 \mapsto \langle u_1, 0 \rangle$

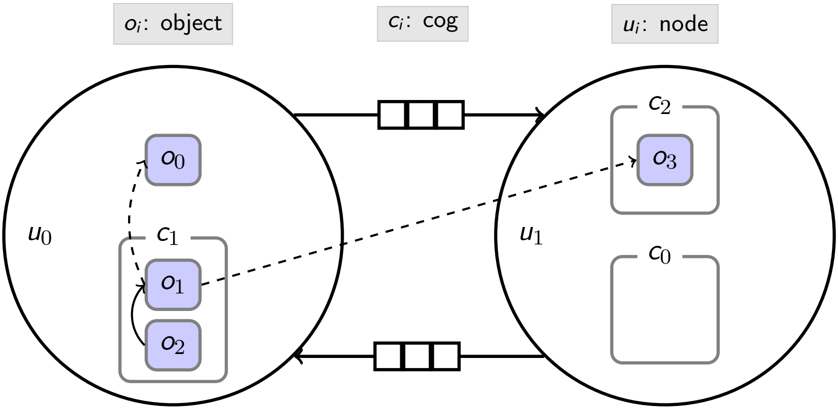
# Mobility




$\mathrm{t}_0 : c_1 \mapsto \langle u_0, 0\rangle,$
$c_2, c_0 \mapsto \langle u_1, 1\rangle$

$\mathrm{t}_1 : c_0, c_1 \mapsto \langle u_0, 1\rangle,$
$c_2 \mapsto \langle u_1, 0\rangle$

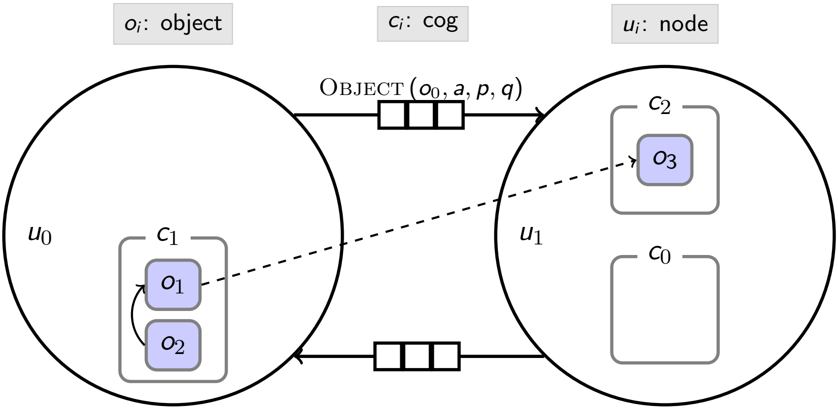
# Mobility, continued




$$t_0 : c_1 \mapsto \langle u_0, 0 \rangle, \quad c_2, c_0 \mapsto \langle u_1, 1 \rangle$$

$$t_1 : c_1 \mapsto \langle u_0, 1 \rangle, \quad c_2, c_0 \mapsto \langle u_1, 0 \rangle$$

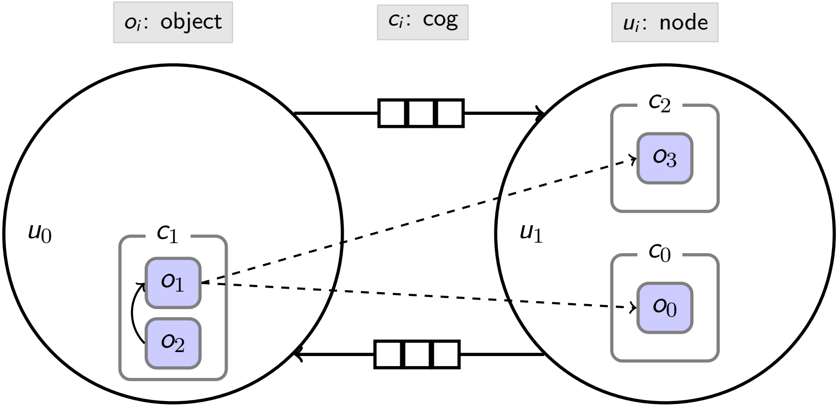
# Mobility, continued




$t_0 : c_1 \mapsto \langle u_0, 0 \rangle,$
$c_2, c_0 \mapsto \langle u_1, 1 \rangle$

$t_1 : c_1 \mapsto \langle u_0, 1 \rangle,$
$c_2, c_0 \mapsto \langle u_1, 0 \rangle$

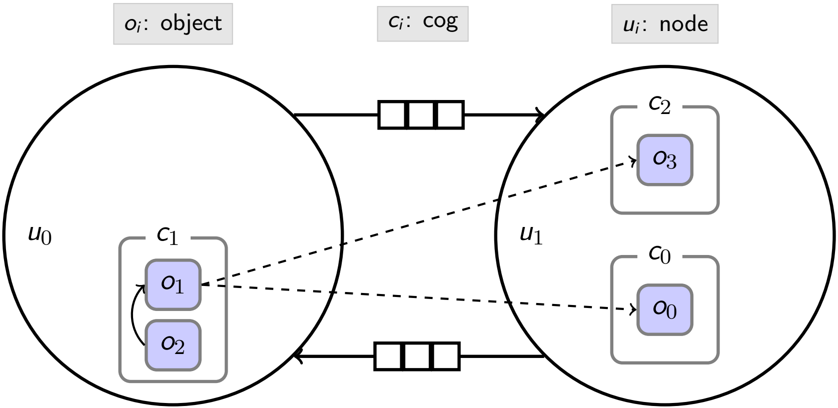
# Mobility, continued




$$\mathrm{t}_0 : c_1 \mapsto \langle u_0, 0 \rangle, \quad c_2, c_0 \mapsto \langle u_1, 1 \rangle$$

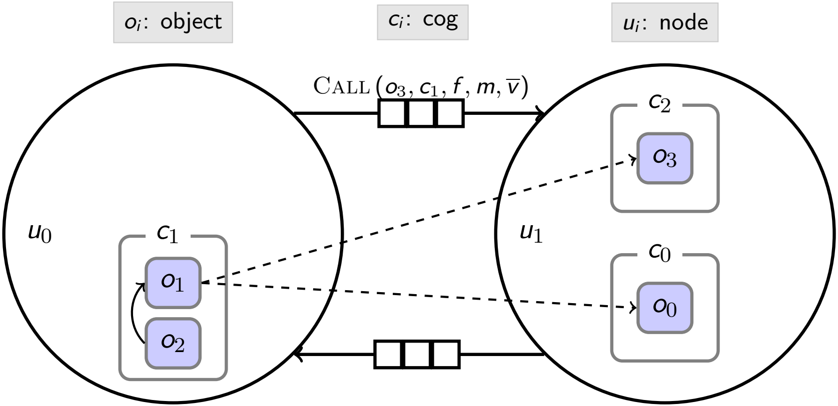
$$\mathrm{t}_1 : c_1 \mapsto \langle u_0, 1 \rangle, \quad c_2, c_0 \mapsto \langle u_1, 0 \rangle$$

# Asynchronous Calls



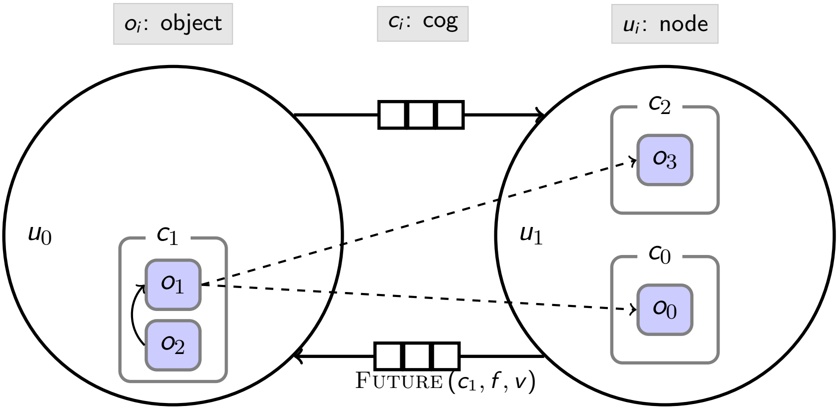

$\text{ob}\,(o_1, a, \{l|x = o_3!m(\overline{v}); s_p\}, q, u_0)$

# Asynchronous Calls, continued



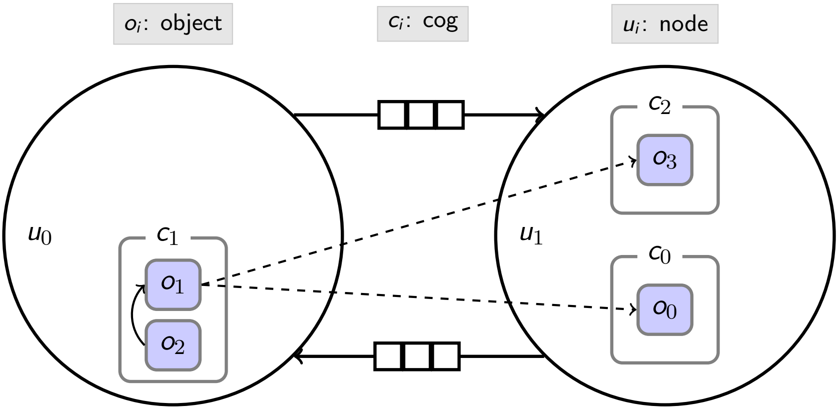

$\text{ob}\,(o_1, a, \{l | x = f; s_p\}, q, u_0)$

# Asynchronous Calls, continued




$\text{ob}\,(o_1, a, \{l'|y = f.\textbf{get}; s'_p\}, q, u_0)$

# Asynchronous Calls, continued




$\mathrm{ob}\,(o_1, a, \{l' | y = v; s'_p\}, q, u_0)$

# Runtime Configurations

| Type | Core ABS | ABS-NET |
|---|---|---|
| node | - | $\mathrm{nd}\,(u, \mathrm{t})$ |
| arc | - | $\mathrm{ar}\,(u, \mathrm{q}, u')$ |
| cog | $\mathrm{cog}\,(c, act)$ | $\mathrm{cog}\,(c, act, u, \mathrm{q}_{in}, \mathrm{q}_{out}, \Sigma)$ |
| object | $\mathrm{ob}\,(o, a, p, q)$ | $\mathrm{ob}\,(o, a, p, q, u)$ |
| fut | $\mathrm{fut}\,(f, val)$ | - |
| invoc | $\mathrm{invoc}\,(o, f, m, \overline{v})$ | - |

# Runtime Syntax

| | | | |
|---|---|---|---|
| $s_p$ | $::=$ | | process statement |
| | $\mid$ | **return** $e$ | return |
| | $\mid$ | **cont** $(f)$ | continue other process |
| | $\mid$ | $s$ | statement |
| | $\mid$ | $s_p; s'_p$ | composition |
| | $\mid$ | $\epsilon$ | empty string |
| | $\mid$ | **forward** $(f, c)$ | forward future value to cog |

# Well-Typed Configurations

$$\frac{\begin{array}{l}\mathit{\Delta} \vdash_R cn\ \mathbf{ok} \\ \text{cn2graph}(cn) = G \\ \text{proper}\,(G) \\ \text{connected}\,(G) \\ \text{symmetric}\,(G)\end{array}}{\mathit{\Delta} \vdash_N cn\ \mathbf{ok}} \quad \text{NET\_T\_CONFIGURATION}$$

# Rule Async-Call-Send

$$\dfrac{\begin{array}{l}[\![e]\!]_{a \circ l} = o' \\ [\![\overline{e}]\!]_{a \circ l} = \overline{v} \\ \mathrm{forwards}\,(\overline{v}, c, \mathrm{cogof}\,(o')) = q' \\ \mathrm{fresh}\,(f) \\ \mathsf{q}_{out} \xrightarrow{\mathrm{enqueue}\,(\textsc{Call}\,(o', c, f, m, \overline{v}))} \mathsf{q}'_{out}\end{array}}{\begin{array}{c}\mathsf{cog}\,(c, o, u, \mathsf{q}_{in}, \mathsf{q}_{out}, \Sigma)\ \mathsf{ob}\,(o, a, \{l | x = e!m(\overline{e}); s_p\}, q, u) \\ \rightarrow \mathsf{cog}\,(c, o, u, \mathsf{q}_{in}, \mathsf{q}'_{out}, \Sigma)\ \mathsf{ob}\,(o, a, \{l | x = f; s_p\}, q \cup q', u)\end{array}} \ \text{NET\_ASY}$$

# Rule Async-Call-Recv

$$\frac{\mathsf{q}_{in} \xrightarrow{\text{dequeue}\left(\textsc{Call}\left(o,c',f,m,\overline{v}\right)\right)} \mathsf{q}'_{in} \qquad \text{bind}\left(o, f, m, \overline{v},\ \text{class}\left(o\right)\right) = \{l|s_p; \epsilon\}}{\begin{array}{c}\mathsf{cog}\left(c, act, u, \mathsf{q}_{in}, \mathsf{q}_{out}, \Sigma\right)\ \mathsf{ob}\left(o, a, p, q, u\right)\\ \to \mathsf{cog}\left(c, act, u, \mathsf{q}'_{in}, \mathsf{q}_{out}, \Sigma\right)\ \mathsf{ob}\left(o, a, p, q \cup \{l|s_p; \textbf{forward}\left(f, c'\right)\}, u\right)\end{array}}$$

# Rule Future-Send

$$\frac{\begin{array}{l} \mathsf{q}_{out} \xrightarrow{\text{enqueue}\,(\textsc{Future}\,(c',f,v))} \mathsf{q}'_{out} \\ \Sigma(f) = v \end{array}}{\begin{array}{c} \mathsf{cog}\,(c, o, u, \mathsf{q}_{in}, \mathsf{q}_{out}, \Sigma)\ \mathsf{ob}\,(o, a, \{l|\mathbf{forward}\,(f, c')\}, q, u) \\ \rightarrow \mathsf{cog}\,(c, o, u, \mathsf{q}_{in}, \mathsf{q}'_{out}, \Sigma)\ \mathsf{ob}\,(o, a, \mathbf{idle}, q, u) \end{array}} \quad \text{NET\_FUTUR}$$

# Rule Future-Recv

$$\frac{\mathsf{q}_{in} \xrightarrow{\text{dequeue}\,(\textsc{Future}\,(c,f,v))} \mathsf{q}'_{in}}{\begin{array}{c}\mathsf{cog}\,(c, act, u, \mathsf{q}_{in}, \mathsf{q}_{out}, \Sigma) \\ \rightarrow \mathsf{cog}\,(c, act, u, \mathsf{q}'_{in}, \mathsf{q}_{out}, \Sigma[f \mapsto v])\end{array}} \quad \text{NET\_FUTURE\_RECV}$$

# Rule Read-Fut

$$\frac{\begin{array}{c}[\![e]\!]_{a \circ l} = f \\ \Sigma(f) = v\end{array}}{\begin{array}{c}\mathrm{cog}\,(c, o, u, \mathrm{q}_{in}, \mathrm{q}_{out}, \Sigma)\ \mathrm{ob}\,(o, a, \{l | x = e.\,\mathbf{get}; s_p\}, q, u) \\ \rightarrow \mathrm{cog}\,(c, o, u, \mathrm{q}_{in}, \mathrm{q}_{out}, \Sigma)\ \mathrm{ob}\,(o, a, \{l | x = v; s_p\}, q, u)\end{array}} \quad \text{NET\_READ\_F}$$

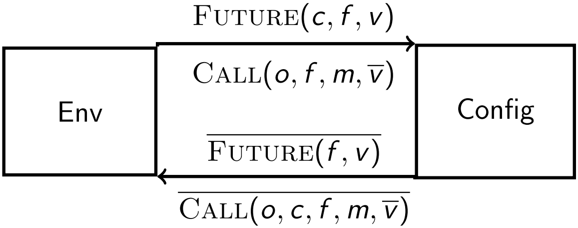
# Starting Configurations

$$\frac{\begin{array}{l}\mathrm{graph2cn}((V, E)) = cn \\ u \in V\end{array}}{\begin{array}{c}\mathrm{start}(\overline{Dd}\ \overline{F}\ \overline{IF}\ \overline{CL}\{\overline{T}\ \overline{x}; s;\}, (V, E), u) = \\ \mathsf{cog}\,(\mathrm{main}, \mathrm{start}, u, nil, nil, [\,])\ \mathsf{ob}\,(\mathrm{start}, \mathsf{cog}\ \mathbf{cog}\ \mathrm{main}, \{\overline{T}\ \overline{x}\ \mathrm{atts}\,(\overline{T})|s; \epsilon\}, \emptyset, u)\end{array}}$$

# Main Problems

1. Preservation of Core ABS behaviour by ABS-NET
   - define runtime interaction with the environment
   - prove bisimilarity for behaviour
2. Applications of ABS-NET for adaptibility
   - nodes as resource-bounded deployment components
   - migration of cogs based on resource availability
3. Efficiency of an ABS-NET implementation
   - publish/subscribe system for futures
   - garbage collection of behaviourally irrelevant entities

# Environmental Transitions for ABS-NET



# Observability

## Definition

For a config *cn* and an environment transition $\mu$, the **observability predicate** $\downarrow_\mu$ is defined by

(1) *cn* $\downarrow_\mu$ if *cn* can perform the input transition $\mu$

(2) *cn* $\downarrow_{\overline{\mu}}$ if *cn* can perform the output transition $\overline{\mu}$

## Definition

For a config *cn* and an environment transition $\mu$, the **weak observability predicate** $\Downarrow_\mu$ is defined by

(1) *cn* $\Downarrow_\mu$ if *cn* can perform the input transition $\mu$ after zero or more "silent" transitions ($\rightarrow$)

(2) *cn* $\Downarrow_{\overline{\mu}}$ if *cn* can perform the output transition $\overline{\mu}$ after zero or more "silent" transitions ($\rightarrow$)

# Simulation

## Definition

A relation $\mathcal{R}$ is a (weak) **barbed simulation** if whenever $(cn_i, cn_j) \in \mathcal{R}$,

(1) $cn_i \downarrow_\mu$ implies $cn_j \Downarrow_\mu$

(2) $cn_i \rightarrow cn'_i$ implies $cn_j \rightarrow^* cn'_j$ for some $cn'_j$ with $(cn'_i, cn'_j) \in \mathcal{R}$.

We say that $cn_j$ **simulates** $cn_i$.

## Definition

A relation $\mathcal{R}$ is a (weak) **barbed bisimulation** if both $\mathrm{R}$ and $\mathrm{R}^{-1}$ is a (weak) barbed simulation. The largest such relation is called (weak) **bisimilarity**.

# Main Result (in progress)

## Theorem

*Let*

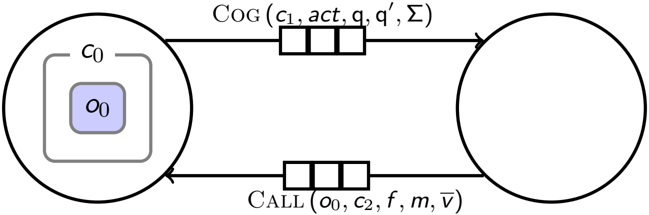
- $P$ *be an ABS program,*
- $G$ *be a directed graph,*
- $u$ *be a node,*
- $\Delta$ *be a typing context,*
- $cn$ *be the ABS-NET start configuration* $\mathrm{start}\,(P, G, u)$*,*
- $cn'$ *be the ABS start configuration* $\mathrm{start}\,(P)$*.*

*Then, if* $\Delta \vdash P$ *and* $\Delta \vdash_N cn\,\mathbf{ok}$*,* $cn$ *simulates* $cn'$*.*

# Bisimulation for Message-Free Configurations

**Lemma**

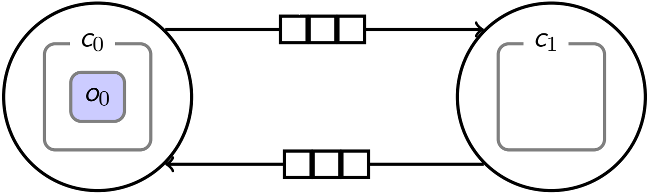
*The ABS-NET configuration cn is bisimilar to the configuration cn′ which is obtained from cn by first replacing all routes with optimal routes and then processing all remaining messages.*



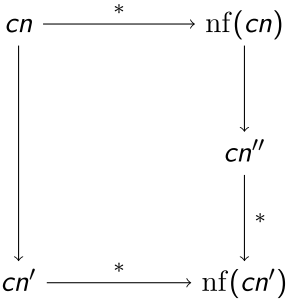
# Bisimulation for Message-Free Configurations

## Lemma

*The ABS-NET configuration cn is bisimilar to the configuration cn′ which is obtained from cn by first replacing all routes with optimal routes and then processing all remaining messages.*



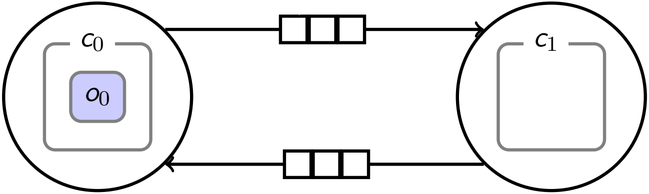
# Proof Idea



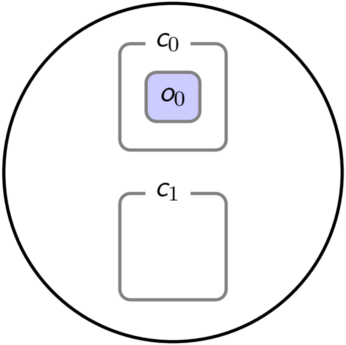
# Simulation of Single-Node Configurations

**Lemma**

*The ABS-NET configuration cn with stable routing and only empty queues is simulates the configuration $cn'$ which is obtained from cn by coalescing all nodes into one and removing all arcs.*



# Simulation of Single-Node Configurations



# Simulation of Core ABS Configurations

**Lemma**

*A single-node ABS-NET configuration cn with empty queues simulates the corresponding Core ABS configuration* $cn'$.

# Nodes as Deployment Components

Nodes can act as deployment components[1] by adding the currently available resources $r$ to the runtime configuration:

$$\frac{\begin{array}{l} x \in \mathrm{dom}(l) \\ [\![e]\!]_{a \circ l} = v \\ vp = l(x) \\ \mathrm{cost}(e) \leq r \end{array}}{\begin{array}{c} \mathsf{nd}(u, \mathsf{t}, r)\ \mathsf{ob}(o, a, \{l | x = e; s_p\}, q, u) \\ \rightarrow \mathsf{nd}(u, \mathsf{t}, r + |vp| - |v|)\ \mathsf{ob}(o, a, \{l[x \mapsto v] | s_p\}, q, u) \end{array}} \quad \text{NET\_ASSIGN\_I}$$

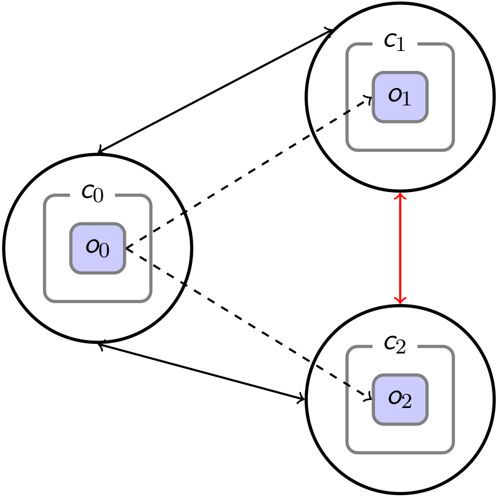
[1] E. Albert, S. Genaim, M. Gómez-Zamalloa, E. B. Johnsen, R. Schlatte, S. L. Tapia Tarifa. Simulating Concurrent Behaviors with Worst-Case Cost Bounds.

# Resource-Based Migration of Cogs

A cog can migrate to a less loaded node when the available resources are below a threshold $K$:

$$\frac{\begin{array}{l} r < K \\ r < \text{available}\,(\mathsf{t}, u') \\ \mathsf{t} \xrightarrow{\text{replace}\,(c,u',1)} \mathsf{t}' \\ \mathsf{q} \xrightarrow{\text{enqueue}\,(\textsc{Cog}\,(c,act,\mathsf{q}_{in},\mathsf{q}_{out},\Sigma))} \mathsf{q}' \end{array}}{\begin{array}{c} \mathsf{nd}\,(u,\mathsf{t},r)\ \mathsf{ar}\,(u,\mathsf{q},u')\ \mathsf{cog}\,(c,act,u,\mathsf{q}_{in},\mathsf{q}_{out},\Sigma) \\ \to \mathsf{nd}\,(u,\mathsf{t}',r+|\Sigma|)\ \mathsf{ar}\,(u,\mathsf{q}',u') \end{array}} \quad \text{NET\_COG\_SEND\_RES}$$

# Publish/Subscribe Systems for Futures




1. $o_0$ calls $o_1$, obtains $f$
2. $o_0$ calls $o_2$ with $f$
3. $o_1$ sends $v$ to $o_0$
4. $o_0$ forwards $v$ to $o_2$
5. $o_2$ sends $v'$ to $o_0$

Channels between $c_1$ and $c_2$ are never used!

$c_2$ must **subscribe** to future $f$.

# Garbage Collection

- Large programs need garbage collection of obsolete entities
- Similar to garbage collection for actors
- Currently investigated option: distributed reference counting

# Current and Future Work

1. Detailed proofs of bisimulations
2. Extend work on deployment components
3. Implementation of ABS-NET on top of IoC's Scala ABS backend
4. Formalize garbage collection
5. Network is currently assumed to be static
    - add rules for node crashes, node joins, add replication
    - extend behavioural similarity proofs