Western Norway
University of
Applied Sciences

Resource Contracts for Active Objects

(Part of the *CroFlow* Project)

Charaf Eddine Dridi
Violet Ka I Pun
Volker Stolz

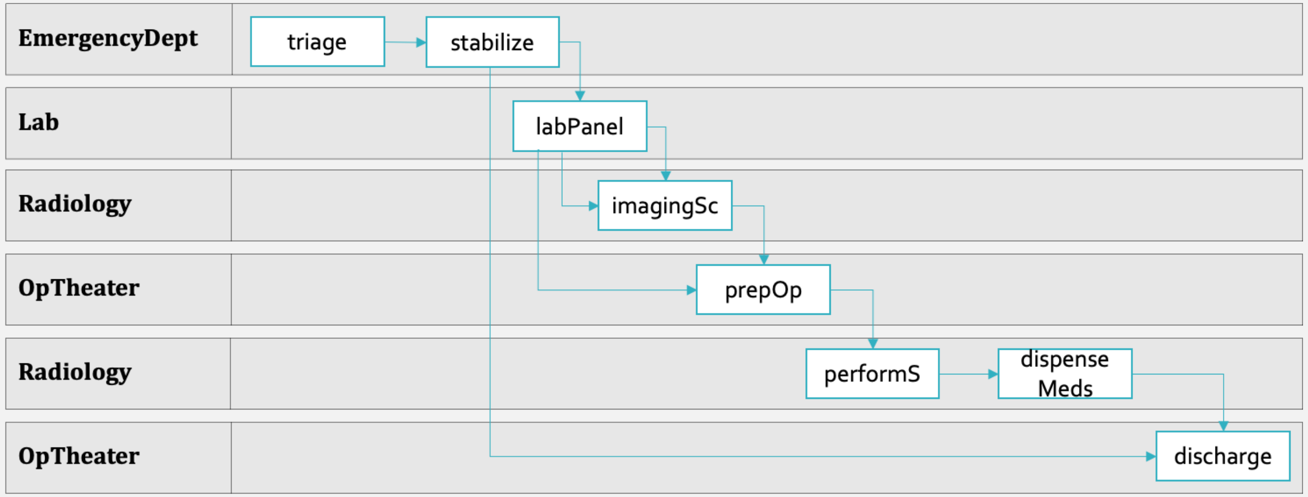
7th International Workshop on Asynchronous Programming Models
2nd October 2025, Porto, Portugal

Challenge of Concurrent Workflows

Require coordination of tasks across distributed units

Must handle:

- Dependencies
- Resources



EmergencyDept: triage, stabilize

Lab: labPanel

Radiology: imagingSc

OpTheater: prepOp

Radiology: performS, dispense Meds

OpTheater: discharge

Active objects coordinate tasks across cross-organizational workflows❓

How do we model and reason about these systems formally❓

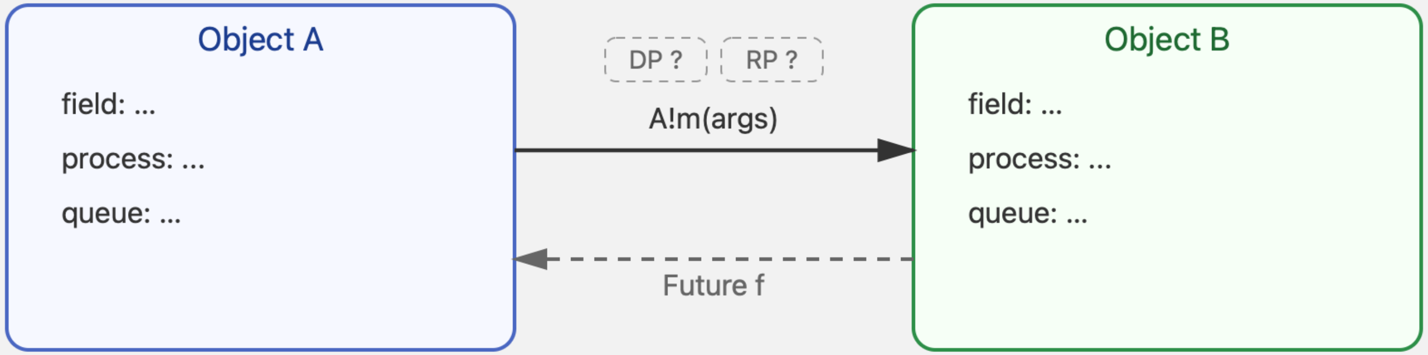
Why Active Objects?

Active Objects :

- Encapsulate process + queue
- Communicate via asynchronous calls (futures)

But: existing models lack built-in support for

- Declarative dependencies
- Resource constraints
- Scheduling policies



ReAct: Resource-Aware Active Objects

Extend active object paradigm:

- Dependencies (DP)
- Resource Profiles (RP)

Formalized in Maude:

- Enforce DP and RP
- Simulation & execution
- Analysis with model checking / search

ReAct : Syntax

Signature as a contract

$Sg ::= T\ m\ (\overline{T\ x})\ \overline{DP}\ \overline{RP}$

- Dependencies

  $\overline{DP} = \{\ DP_1, DP_2\ \},$

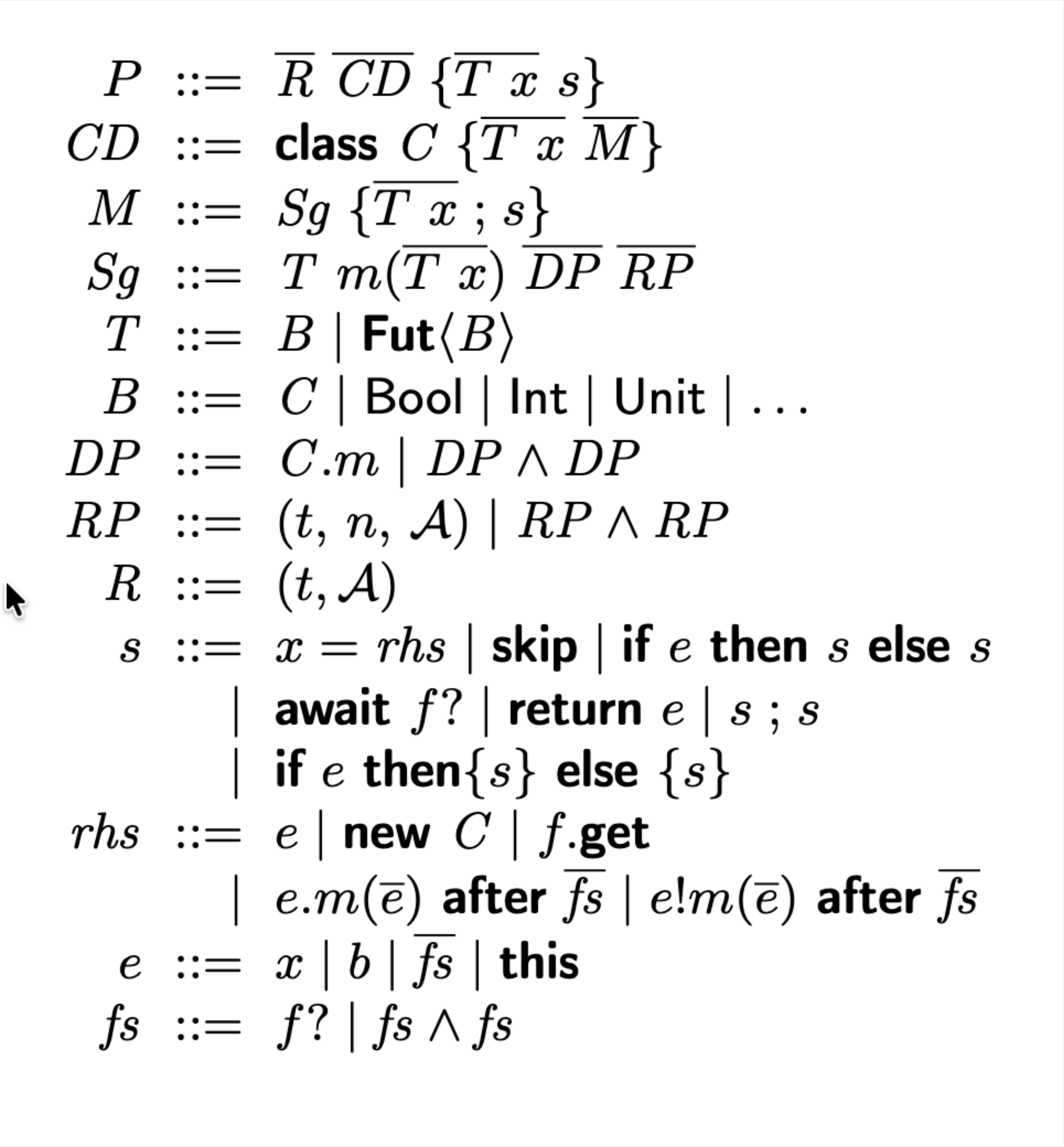
  $where\ \ DP_1 = C_1.m_1, DP_2\ = C_2.m_2 \wedge\ C'_2.m'_2$

- Resources Requirements

  $\overline{RP} = \{\ RP_1, RP_2\ \},$

  $where\ RP_1 = (t_1.n_1, A_1),$

  $RP_2\ = (t_2.n_2, A_2) \wedge\ (t'_2, n'_2, A'_2)$

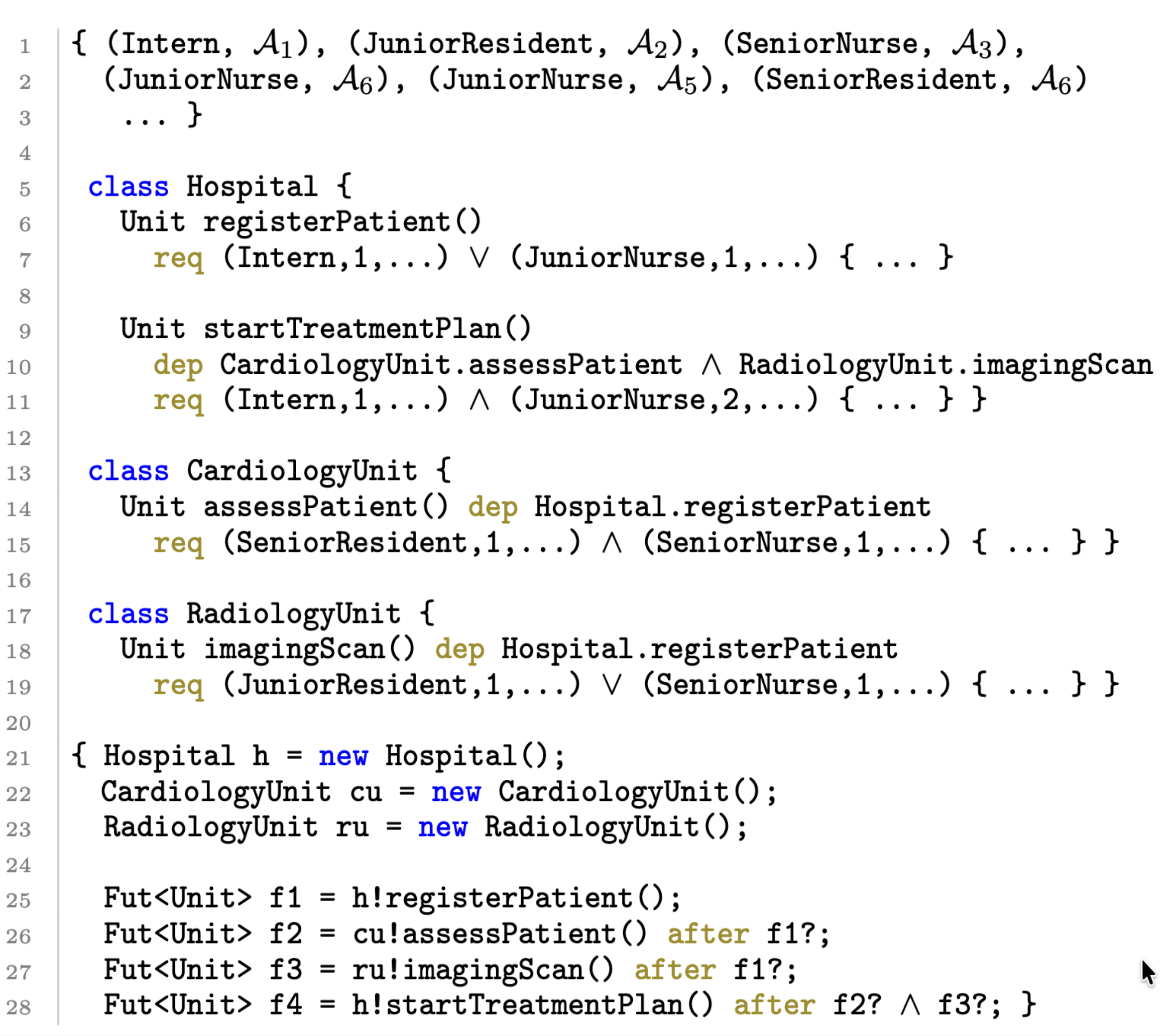


$$
\begin{array}{rcl}
P & ::= & \overline{R}\ \overline{CD}\ \{\overline{T\ x}\ s\} \\
CD & ::= & \textbf{class}\ C\ \{\overline{T\ x}\ \overline{M}\} \\
M & ::= & Sg\ \{\overline{T\ x}\ ;\ s\} \\
Sg & ::= & T\ m(\overline{T\ x})\ \overline{DP}\ \overline{RP} \\
T & ::= & B \mid \textbf{Fut}\langle B\rangle \\
B & ::= & C \mid \mathsf{Bool} \mid \mathsf{Int} \mid \mathsf{Unit} \mid \ldots \\
DP & ::= & C.m \mid DP \wedge DP \\
RP & ::= & (t,\ n,\ \mathcal{A}) \mid RP \wedge RP \\
R & ::= & (t, \mathcal{A}) \\
s & ::= & x = rhs \mid \textbf{skip} \mid \textbf{if}\ e\ \textbf{then}\ s\ \textbf{else}\ s \\
 & \mid & \textbf{await}\ f? \mid \textbf{return}\ e \mid s\ ;\ s \\
 & \mid & \textbf{if}\ e\ \textbf{then}\{s\}\ \textbf{else}\ \{s\} \\
rhs & ::= & e \mid \textbf{new}\ C \mid f.\textbf{get} \\
 & \mid & e.m(\overline{e})\ \textbf{after}\ \overline{fs} \mid e!m(\overline{e})\ \textbf{after}\ \overline{fs} \\
e & ::= & x \mid b \mid \overline{fs} \mid \textbf{this} \\
fs & ::= & f? \mid fs \wedge fs
\end{array}
$$

ReAct : Syntax

Example : Hospital Workflow

- Resources Pool
- Methods
  - Dependencies – dep
  - Resources Profiles – req
- Workflow – after



```
{ (Intern, $\mathcal{A}_1$), (JuniorResident, $\mathcal{A}_2$), (SeniorNurse, $\mathcal{A}_3$),
  (JuniorNurse, $\mathcal{A}_6$), (JuniorNurse, $\mathcal{A}_5$), (SeniorResident, $\mathcal{A}_6$)
   ... }

 class Hospital {
   Unit registerPatient()
     req (Intern,1,...) ∨ (JuniorNurse,1,...) { ... }

   Unit startTreatmentPlan()
     dep CardiologyUnit.assessPatient ∧ RadiologyUnit.imagingScan
     req (Intern,1,...) ∧ (JuniorNurse,2,...) { ... } }

 class CardiologyUnit {
   Unit assessPatient() dep Hospital.registerPatient
     req (SeniorResident,1,...) ∧ (SeniorNurse,1,...) { ... } }

 class RadiologyUnit {
   Unit imagingScan() dep Hospital.registerPatient
     req (JuniorResident,1,...) ∨ (SeniorNurse,1,...) { ... } }

{ Hospital h = new Hospital();
  CardiologyUnit cu = new CardiologyUnit();
  RadiologyUnit ru = new RadiologyUnit();

  Fut<Unit> f1 = h!registerPatient();
  Fut<Unit> f2 = cu!assessPatient() after f1?;
  Fut<Unit> f3 = ru!imagingScan() after f1?;
  Fut<Unit> f4 = h!startTreatmentPlan() after f2? ∧ f3?; }
```

ReAct : Semantics

Calls Evaluation

- Async-Call-After
- Sync-Call-After
- Async-Call
- Sync-Call

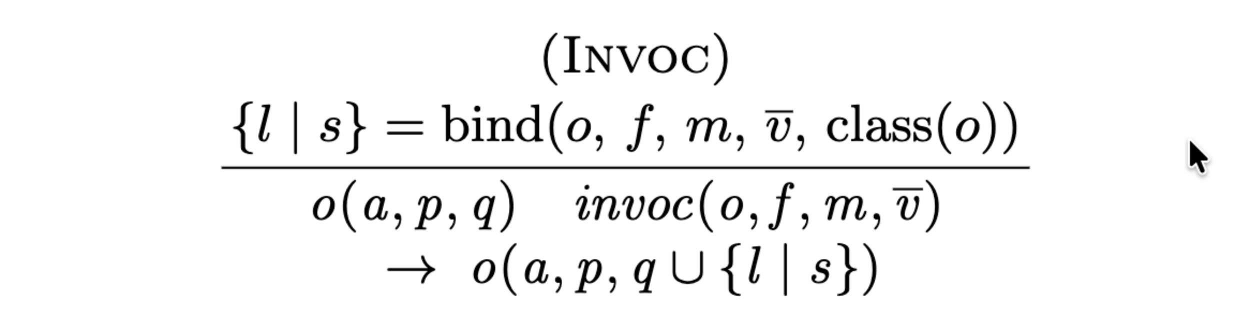
$$\begin{gathered}(\textsc{Sync-Call-After})\\ o(a, \{l \mid x = e.m(\overline{e})\ \textbf{after}\ \overline{fs}\ ;\ s\}, q)\\ \to o(a, \{l \mid \textbf{if}\ \overline{fs}\ \{x = e.m(\overline{e})\ ;\ s\}\ \textbf{else}\ \{\textbf{suspend}\ ;\ x = e.m(\overline{e})\ \textbf{after}\ \overline{fs}\ ;\ s\}\}, q)\end{gathered}$$

$$\begin{gathered}(\textsc{Async-Call-After})\\ o(a, \{l \mid x = e!m(\overline{e})\ \textbf{after}\ \overline{fs}\ ;\ s\}, q)\\ \to o(a, \{l \mid \textbf{if}\ \overline{fs}\ \{x = e!m(\overline{e})\ \ ;\ s\}\ \textbf{else}\ \{\textbf{suspend}\ ;\ x = e!m(\overline{e})\ \textbf{after}\ \overline{fs}\ ;\ s\}\}, q)\end{gathered}$$

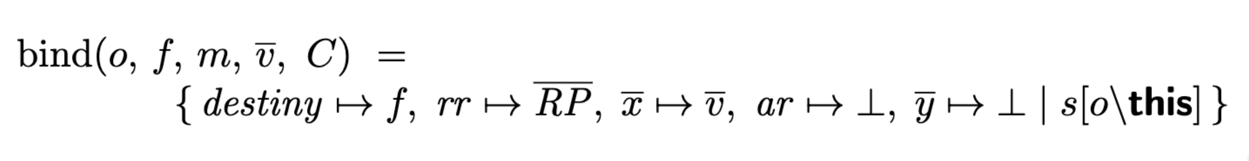
ReAct : Semantics

Invocations

- Invoc
- bind method



$$\textsc{(Invoc)}\quad \frac{\{l \mid s\} = \mathrm{bind}(o,\ f,\ m,\ \overline{v},\ \mathrm{class}(o))}{o(a,p,q)\quad invoc(o,f,m,\overline{v}) \\ \to\ o(a,p,q \cup \{l \mid s\})}$$

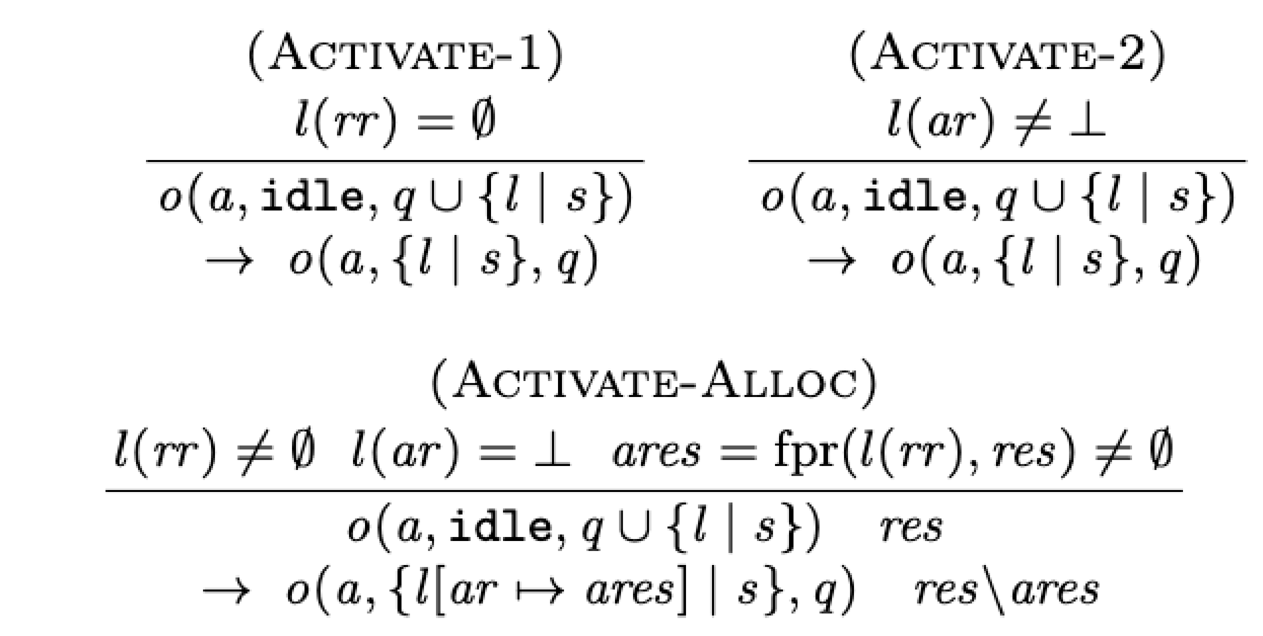


$$\begin{aligned}\mathrm{bind}(o,\ f,\ m,\ \overline{v},\ C)\ =&\\ \{\, destiny \mapsto f,\ rr \mapsto \overline{RP},\ \overline{x} \mapsto \overline{v},\ ar \mapsto \bot,\ \overline{y} \mapsto \bot \mid s[o\backslash \textbf{this}]\,\}&\end{aligned}$$

ReAct : Semantics

Activation and Resources Allocation

- Activate-1 : No resources needed
- Activate-2 : Already have resources
- Activate-Alloc : Need resources



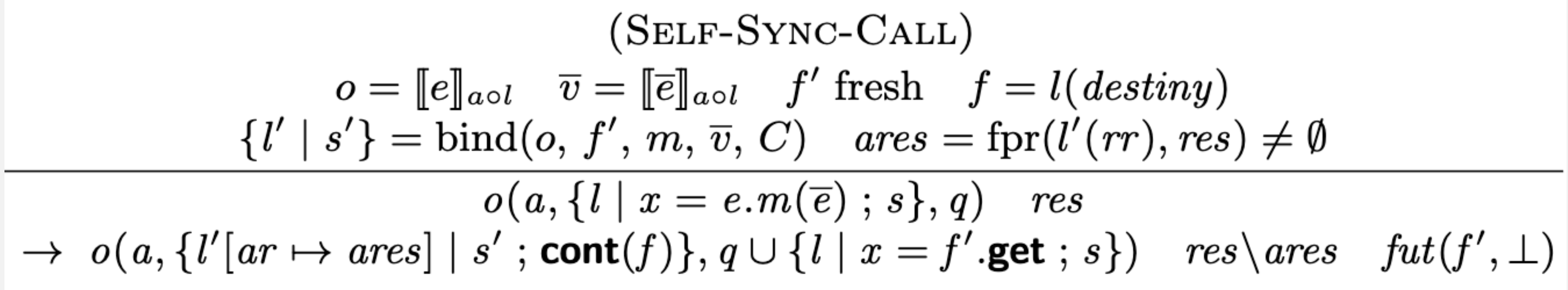
$$\textsc{(Activate-1)}\quad \frac{l(rr) = \emptyset}{\begin{array}{c} o(a, \texttt{idle}, q \cup \{l \mid s\}) \\ \to \; o(a, \{l \mid s\}, q) \end{array}} \qquad \textsc{(Activate-2)}\quad \frac{l(ar) \neq \bot}{\begin{array}{c} o(a, \texttt{idle}, q \cup \{l \mid s\}) \\ \to \; o(a, \{l \mid s\}, q) \end{array}}$$

$$\textsc{(Activate-Alloc)}\quad \frac{l(rr) \neq \emptyset \quad l(ar) = \bot \quad ares = \mathrm{fpr}(l(rr), res) \neq \emptyset}{\begin{array}{c} o(a, \texttt{idle}, q \cup \{l \mid s\}) \quad res \\ \to \; o(a, \{l[ar \mapsto ares] \mid s\}, q) \quad res \backslash ares \end{array}}$$

ReAct : Semantics

Self-Sync Calls

- Self-Sync-Call



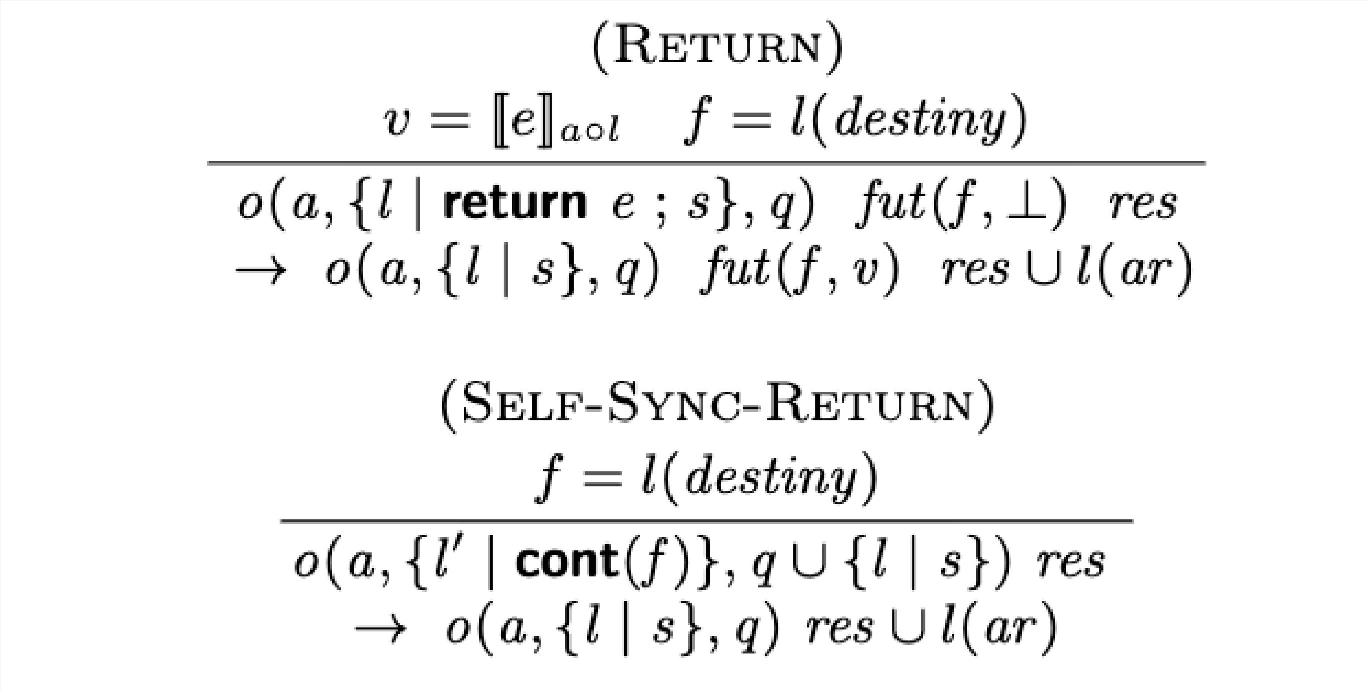
$$\textsc{(Self-Sync-Call)}$$

$$\frac{\begin{array}{c} o = [\![e]\!]_{a\circ l} \quad \overline{v} = [\![\overline{e}]\!]_{a\circ l} \quad f' \text{ fresh} \quad f = l(destiny) \\ \{l' \mid s'\} = \text{bind}(o,\ f',\ m,\ \overline{v},\ C) \quad ares = \text{fpr}(l'(rr), res) \neq \emptyset \end{array}}{\begin{array}{c} o(a, \{l \mid x = e.m(\overline{e})\ ;\ s\}, q) \quad res \\ \to\ o(a, \{l'[ar \mapsto ares] \mid s'\ ;\ \textbf{cont}(f)\}, q \cup \{l \mid x = f'.\textbf{get}\ ;\ s\}) \quad res \backslash ares \quad fut(f', \bot) \end{array}}$$

ReAct : Semantics

Return and Future Resolution

- Return
- Self-Sync-Return



(Return)

$$\frac{v = [\![e]\!]_{a \circ l} \quad f = l(destiny)}{\begin{array}{l} o(a, \{l \mid \textbf{return } e \; ; s\}, q) \;\; fut(f, \bot) \;\; res \\ \to \; o(a, \{l \mid s\}, q) \;\; fut(f, v) \;\; res \cup l(ar) \end{array}}$$

(Self-Sync-Return)

$$\frac{f = l(destiny)}{\begin{array}{l} o(a, \{l' \mid \textbf{cont}(f)\}, q \cup \{l \mid s\}) \; res \\ \to \; o(a, \{l \mid s\}, q) \; res \cup l(ar) \end{array}}$$

Formalizing *ReAct* in Maude

Why Maude?

- **We require:**
  - Rich structural representation
  - Conditional behaviors
  - Executable specifications

- **Maude is based on rewriting logic**:
  - Expressive
  - Executable
  - Equipped with verification tools (search, model checking)

- **Execuatable *ReAct*:**
  - Signature (sorts, ops, classes)
  - Equations & attributes
  - Rewrite rules : Rules transform configurations using pattern matching and conditions

$$\mathrm{def}_{ReAct} = \{\Sigma_{ReAct}, (E \cup A)_{ReAct},\ R_{ReAct}\}$$

Formalizing *ReAct* in Maude

Syntax

<table>
<tr><th>ReAct</th><th>Maude</th></tr>
<tr><td><b>Object</b></td><td>class OBJECT | fields : Int, proc : ProcessState, suspended : ProcessPool .</td></tr>
<tr><td><b>Method</b></td><td>class METHOD | sig : Oid, body : Oid .</td></tr>
<tr><td><b>Signature</b></td><td>class SIGNATURE | ret : Int, name : MethodName, params : ParamList, dp : DP, requires : ResourceProfile .</td></tr>
<tr><td><b>Resource Profile</b></td><td>class RESOURCE | type : String, attrs : AttrSet, state : ResState, ResCost : Int .
op noneProfile : -> ResourceProfile .
op _and_ : ResourceProfile ResourceProfile -> ResourceProfile [ctor assoc comm id: noneProfile] .
op _or_ : ResourceProfile ResourceProfile -> ResourceProfile [ctor assoc comm id: noneProfile] .</td></tr>
<tr><td><b>Statement</b></td><td>sort Statement .
op _= _!_(_) : LocalVar Expr Oid Args -> Statement [ctor] .
op _= _. _(_) : LocalVar Expr Oid Args -> Statement [ctor] .
op _= _! _(_) after_ : LocalVar Expr Oid Args Clause -> Statement [ctor] .
op _= _._(_) after_ : LocalVar Expr Oid Args Clause -> Statement [ctor] .
ops skip suspend : -> Statement [ctor] .
op await : Oid -> Statement [ctor] .</td></tr>
<tr><td><b>Process State</b></td><td>sort ProcessState .
ops idle : -> ProcessState .
op { _ | _ } : LocalVarList Statement -> ProcessState [ctor] .</td></tr>
<tr><td><b>Future</b></td><td>sort FutureState .
class Future | value : ValueOption, state : FutureState .
ops unresolved resolved : -> FutureState [ctor]</td></tr>
</table>

Formalizing *ReAct* in Maude

Semantics

ReAct	Maude
Equations	clauseSatisfied(FS), bind(O,F,...), get(F), feasibleProfile(RP, RS) .
Messages	op invoc(O, F, M, A) : Oid FutureOid Oid Args -> Msg [ctor] .
Rules	crl [rewrite-rule-name] : State => State' if Equation .
Properties	search [n] in ModId : initial-state =>! pattern [such that cond] .

Formalizing *ReAct* in Maude

Rewriting Rules

```
rl [Async-Call-After] :
   < O : OBJECT | fields : Fld, proc : { LVL | ( X = E ! M(A)after Fset ) ; S},
                                  suspended : Q >
=>
   < O : OBJECT | fields : Fld, proc : { LVL | if clauseSatisfied(FSet)
                                                          then ( X = E ! M(A) ) ; S
                                                          else (suspend ; X = E ! M(A)after FSet) ; S
                                                          fi},
                                  suspended : Q > .
```

```
crl [activate-alloc] :
   < RP : RESOURCE-POOL | pool : ResS >
   < O : OBJECT | fields : Fld,
                                  proc : idle,
                                  suspended : { LVL ; (rr :== RP1) ; (ar :== noneProfile) | S } ; Q >
=>
   < RP : RESOURCE-POOL | pool : reserveResources(feasibleProfile(RP1, ResS), ResS) >
   < O : OBJECT | fields : Fld,
                                  proc : { LVL ; (rr :== RP1) ; (ar :== feasibleProfile(RP1, ResS)) | S },
                    suspended : Q >
if RP1 =/= noneProfile .
```

Simulation and Analysis

Example : Hospital Workflow

Main

```
< bmain : METHODBODY | stmt :
                ((fregRecord       = Hospital ! registerPatient()) ;
                (fcardioAssess     = CardiologyUnit ! assessPatient() after fregRecord) ;
                (fimagingScan      = RadiologyUnit ! imagingScan() after fregRecord) ;
                (finitiateTreatment                = Hospital ! startTreatmentPlan() after (fcardioAssess /\ fimagingScan))) >
```

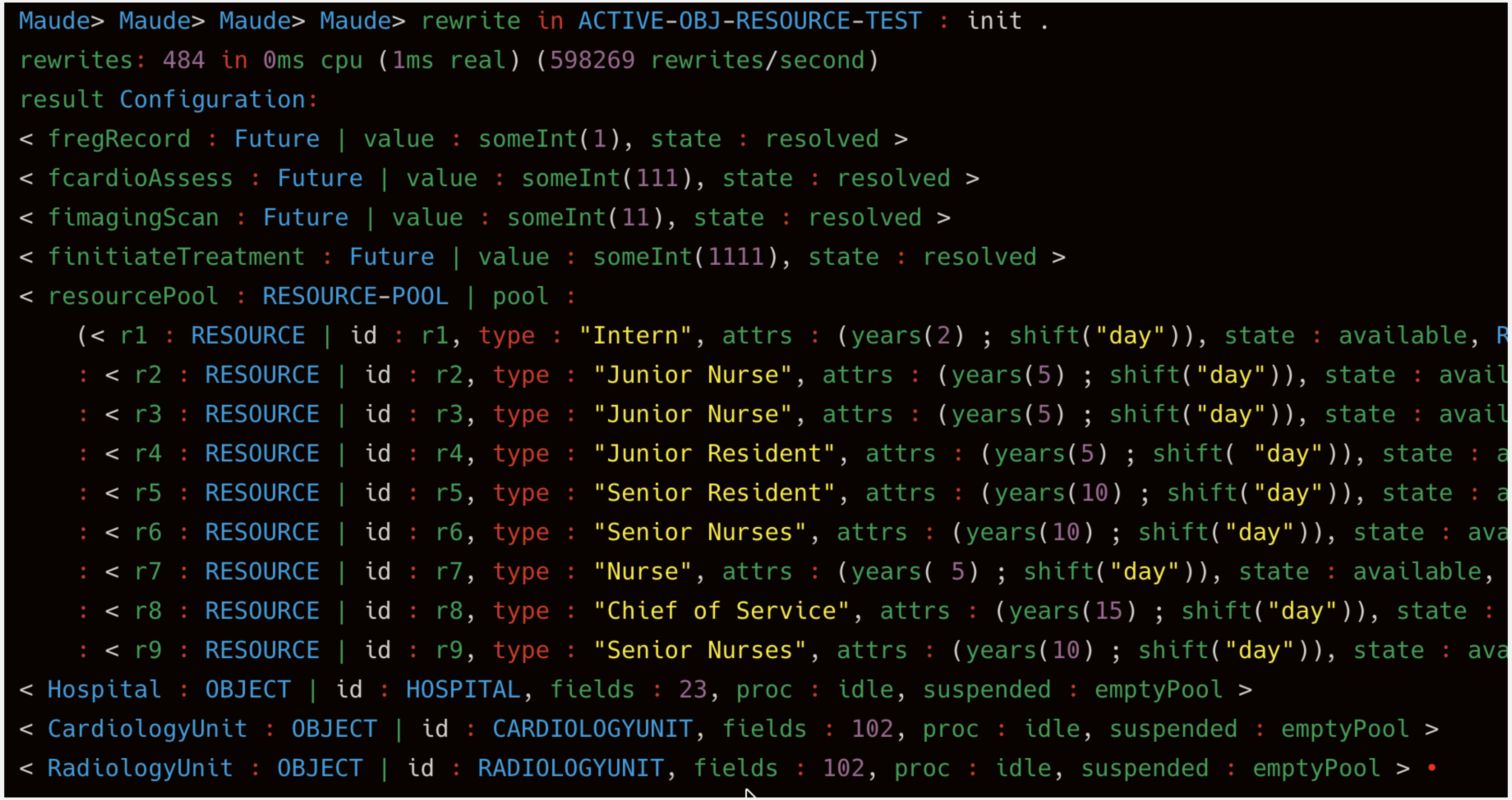
Signatures

```
< sigregisterPatient              : SIGNATURE | name : registerPatient, params : pl1,
                                  requires         : needs("Intern", 1, (years(2) ; shift("day")))
                                                   and needs("Junior Nurse", 1, (years(5) ; shift("day"))) >

< sigassessPatient                : SIGNATURE | name : assessPatient, params : pl2,
                                  depends          : (HOSPITAL . registerPatient),
                                  requires         : needs("Senior Resident", 1, (years(10) ; shift("day")))
                                                   and needs("Senior Nurse", 1, (years(10) ; shift("day"))) >

< sigimagingScan                  : SIGNATURE | name : imagingScan, params : pl3,
                                  depends          : (HOSPITAL . registerPatient),
                                  requires         : needs("Junior Resident", 1, (years(5) ; shift("day")))
                                                   or needs("Senior Nurse", 1, (years(10) ; shift("day"))) >

< sigstartTreatmentPlan : SIGNATURE | name : startTreatmentPlan, params : pl4,
                depends           : (CARDIOLOGYUNIT . assessPatient) ; (RADIOLOGYUNIT . imagingScan),
                                  requires         : needs("Intern", 1, (years(2) ; shift("day")))
                                                   or needs("Junior Nurse", 2, (years(5) ; shift("day"))) >
```

Simulation and Analysis

Execution

- All tasks completed
- Correct resource discipline
- All futures resolved
- No pending invocations
- No suspended processes

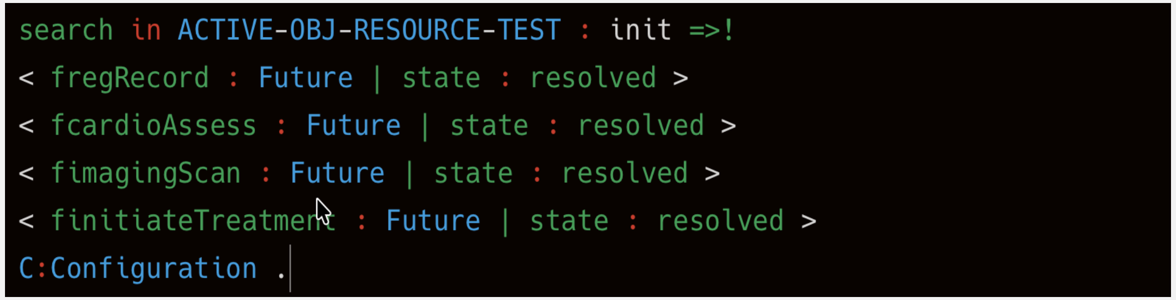


```
Maude> Maude> Maude> Maude> rewrite in ACTIVE-OBJ-RESOURCE-TEST : init .
rewrites: 484 in 0ms cpu (1ms real) (598269 rewrites/second)
result Configuration:
< fregRecord : Future | value : someInt(1), state : resolved >
< fcardioAssess : Future | value : someInt(111), state : resolved >
< fimagingScan : Future | value : someInt(11), state : resolved >
< finitiateTreatment : Future | value : someInt(1111), state : resolved >
< resourcePool : RESOURCE-POOL | pool :
    (< r1 : RESOURCE | id : r1, type : "Intern", attrs : (years(2) ; shift("day")), state : available, R
    : < r2 : RESOURCE | id : r2, type : "Junior Nurse", attrs : (years(5) ; shift("day")), state : avail
    : < r3 : RESOURCE | id : r3, type : "Junior Nurse", attrs : (years(5) ; shift("day")), state : avail
    : < r4 : RESOURCE | id : r4, type : "Junior Resident", attrs : (years(5) ; shift( "day")), state : a
    : < r5 : RESOURCE | id : r5, type : "Senior Resident", attrs : (years(10) ; shift("day")), state : a
    : < r6 : RESOURCE | id : r6, type : "Senior Nurses", attrs : (years(10) ; shift("day")), state : ava
    : < r7 : RESOURCE | id : r7, type : "Nurse", attrs : (years( 5) ; shift("day")), state : available,
    : < r8 : RESOURCE | id : r8, type : "Chief of Service", attrs : (years(15) ; shift("day")), state :
    : < r9 : RESOURCE | id : r9, type : "Senior Nurses", attrs : (years(10) ; shift("day")), state : ava
< Hospital : OBJECT | id : HOSPITAL, fields : 23, proc : idle, suspended : emptyPool >
< CardiologyUnit : OBJECT | id : CARDIOLOGYUNIT, fields : 102, proc : idle, suspended : emptyPool >
< RadiologyUnit : OBJECT | id : RADIOLOGYUNIT, fields : 102, proc : idle, suspended : emptyPool >
```

Simulation and Analysis

Reachability analysis

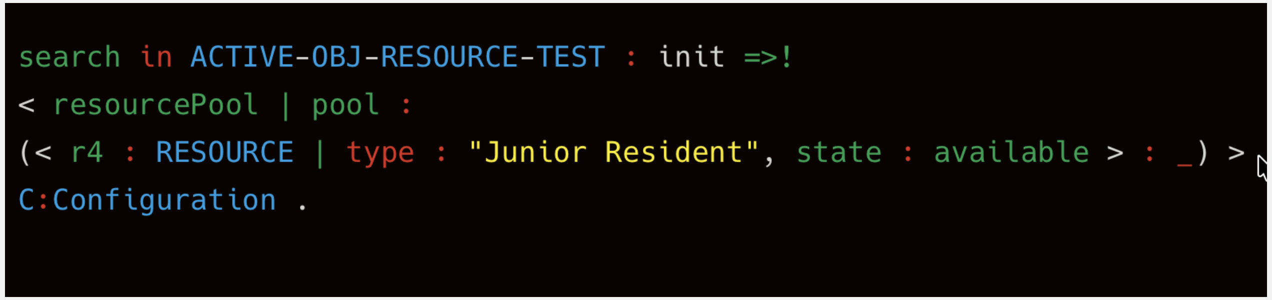
- All futures Resolved



```
search in ACTIVE-OBJ-RESOURCE-TEST : init =>!
< fregRecord : Future | state : resolved >
< fcardioAssess : Future | state : resolved >
< fimagingScan : Future | state : resolved >
< finitiateTreatment : Future | state : resolved >
C:Configuration .
```

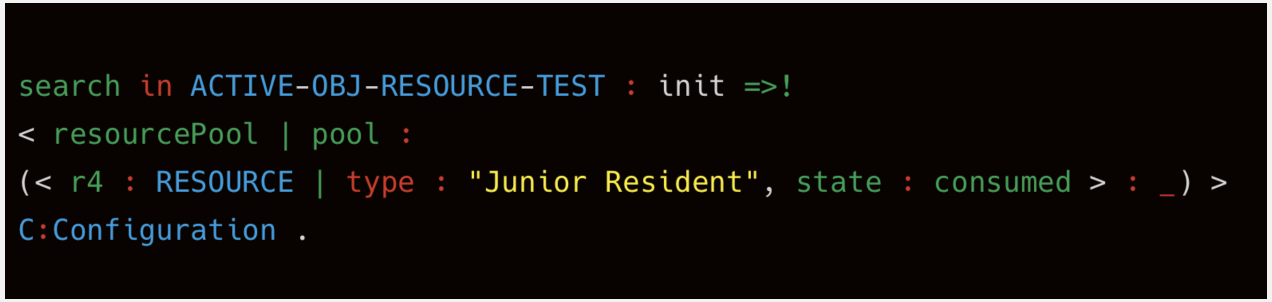
→ **Succeeds**

- Resource Release



```
search in ACTIVE-OBJ-RESOURCE-TEST : init =>!
< resourcePool | pool :
(< r4 : RESOURCE | type : "Junior Resident", state : available > : _) >
C:Configuration .
```

→ **Succeeds**



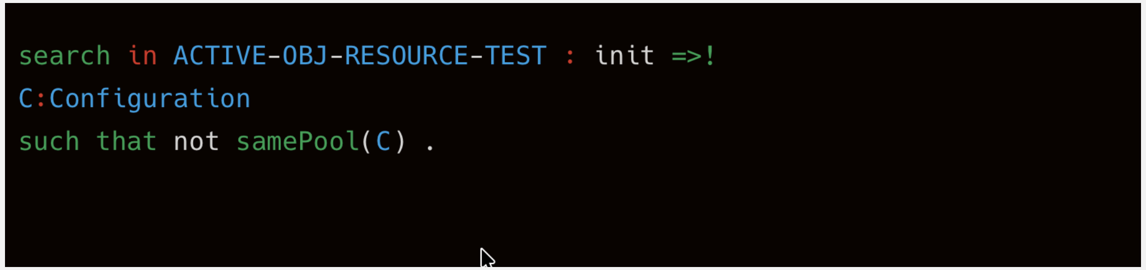
```
search in ACTIVE-OBJ-RESOURCE-TEST : init =>!
< resourcePool | pool :
(< r4 : RESOURCE | type : "Junior Resident", state : consumed > : _) >
C:Configuration .
```

→ **Fails**

Simulation and Analysis

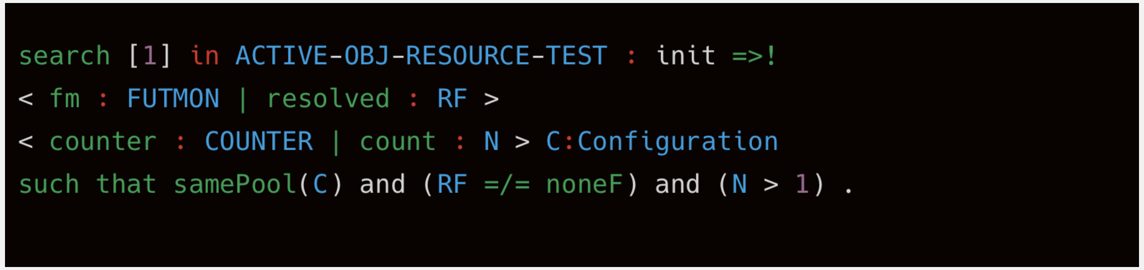
Reachability analysis

- Global pool restoration



```
search in ACTIVE-OBJ-RESOURCE-TEST : init =>!
C:Configuration
such that not samePool(C) .
```

→ **Fails**



```
search [1] in ACTIVE-OBJ-RESOURCE-TEST : init =>!
< fm : FUTMON | resolved : RF >
< counter : COUNTER | count : N > C:Configuration
such that samePool(C) and (RF =/= noneF) and (N > 1) .
```

→ **Succeeds**

Conclusion

Each method declares a contract

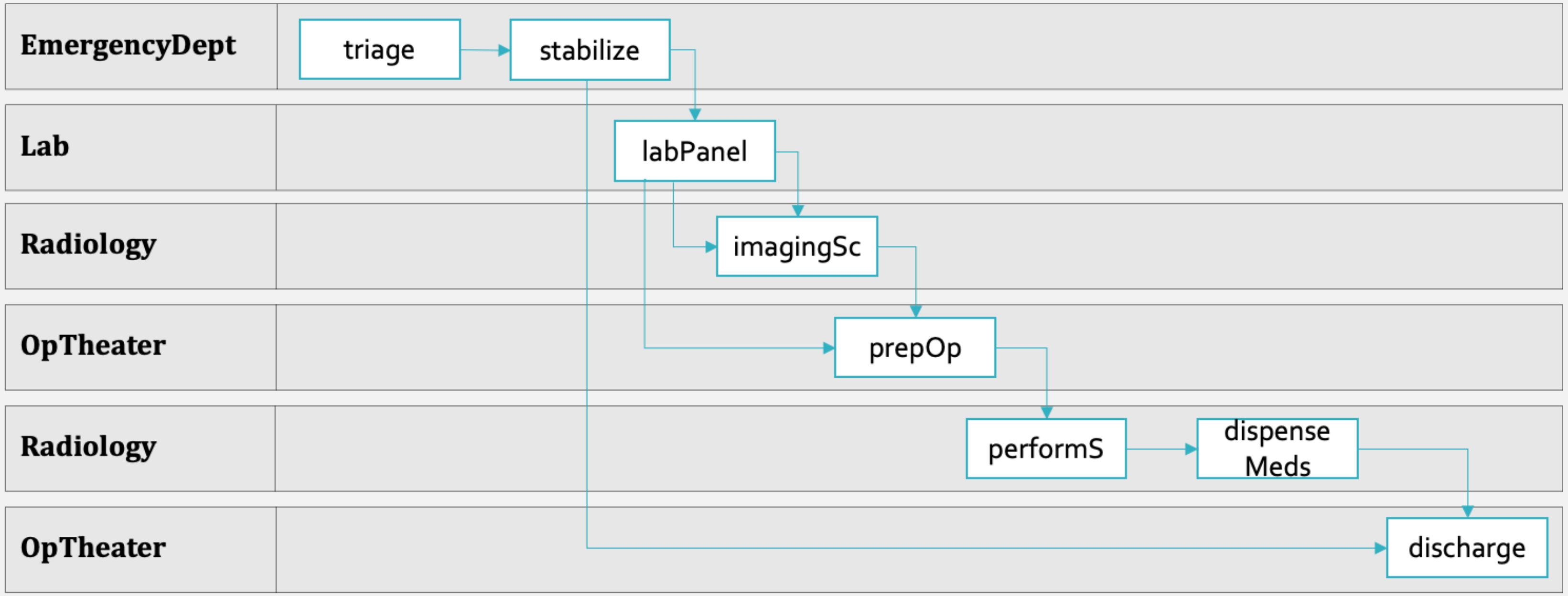
- Dependencies (DP): Invoke a method only when DP holds
- Resource Profile (RP): Allocate resources atomically on activation; release them on return

Encoded in Maude (rewriting logic)

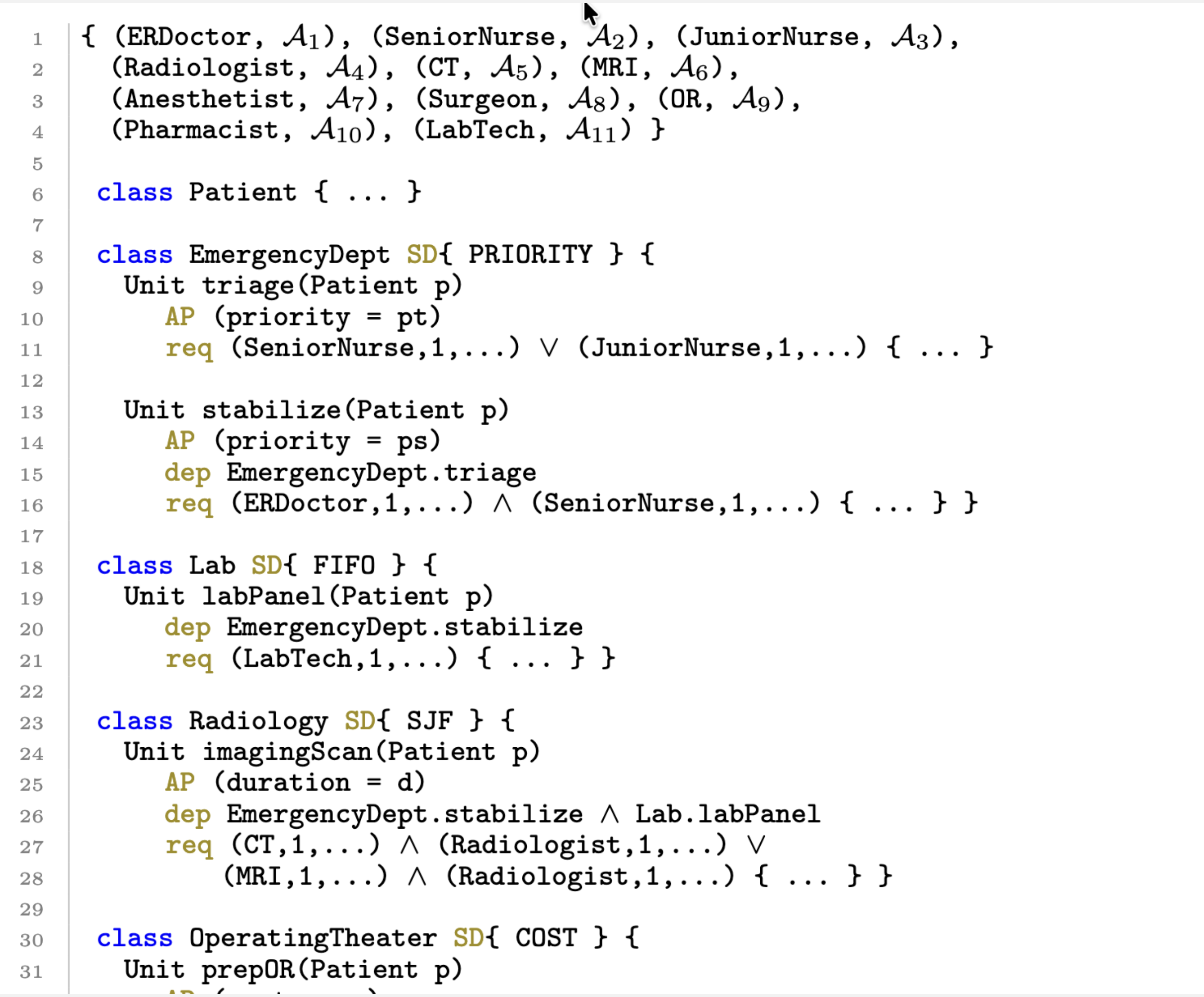
- Executable semantics
- Simulation and analysis

Future Work

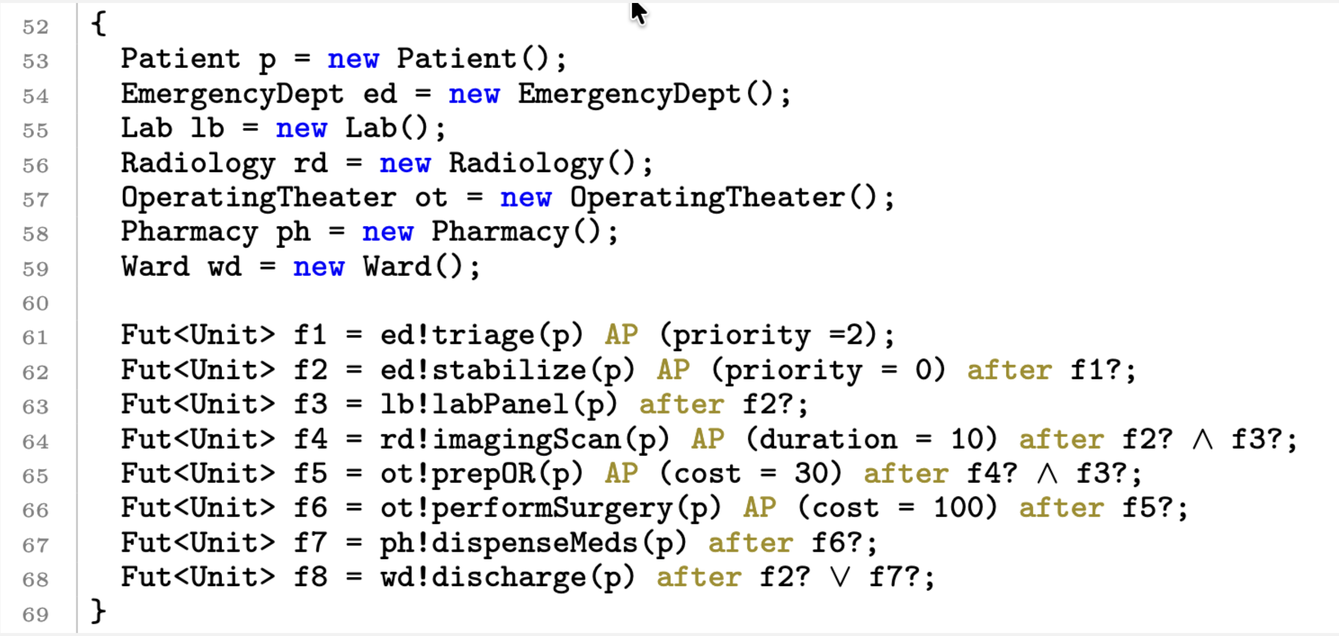
- Activation Policy



Future Work



```
{ (ERDoctor, $\mathcal{A}_1$), (SeniorNurse, $\mathcal{A}_2$), (JuniorNurse, $\mathcal{A}_3$),
  (Radiologist, $\mathcal{A}_4$), (CT, $\mathcal{A}_5$), (MRI, $\mathcal{A}_6$),
  (Anesthetist, $\mathcal{A}_7$), (Surgeon, $\mathcal{A}_8$), (OR, $\mathcal{A}_9$),
  (Pharmacist, $\mathcal{A}_{10}$), (LabTech, $\mathcal{A}_{11}$) }

 class Patient { ... }

 class EmergencyDept SD{ PRIORITY } {
   Unit triage(Patient p)
      AP (priority = pt)
      req (SeniorNurse,1,...) ∨ (JuniorNurse,1,...) { ... }

   Unit stabilize(Patient p)
      AP (priority = ps)
      dep EmergencyDept.triage
      req (ERDoctor,1,...) ∧ (SeniorNurse,1,...) { ... } }

 class Lab SD{ FIFO } {
   Unit labPanel(Patient p)
      dep EmergencyDept.stabilize
      req (LabTech,1,...) { ... } }

 class Radiology SD{ SJF } {
   Unit imagingScan(Patient p)
      AP (duration = d)
      dep EmergencyDept.stabilize ∧ Lab.labPanel
      req (CT,1,...) ∧ (Radiologist,1,...) ∨
          (MRI,1,...) ∧ (Radiologist,1,...) { ... } }

 class OperatingTheater SD{ COST } {
   Unit prepOR(Patient p)
```



```
{
  Patient p = new Patient();
  EmergencyDept ed = new EmergencyDept();
  Lab lb = new Lab();
  Radiology rd = new Radiology();
  OperatingTheater ot = new OperatingTheater();
  Pharmacy ph = new Pharmacy();
  Ward wd = new Ward();

  Fut<Unit> f1 = ed!triage(p) AP (priority =2);
  Fut<Unit> f2 = ed!stabilize(p) AP (priority = 0) after f1?;
  Fut<Unit> f3 = lb!labPanel(p) after f2?;
  Fut<Unit> f4 = rd!imagingScan(p) AP (duration = 10) after f2? ∧ f3?;
  Fut<Unit> f5 = ot!prepOR(p) AP (cost = 30) after f4? ∧ f3?;
  Fut<Unit> f6 = ot!performSurgery(p) AP (cost = 100) after f5?;
  Fut<Unit> f7 = ph!dispenseMeds(p) after f6?;
  Fut<Unit> f8 = wd!discharge(p) after f2? ∨ f7?;
}
```