μse: programming multi-party sessions for SOC

Emilio Tuosto

et52@mcs.le.ac.uk

Roberto Bruni

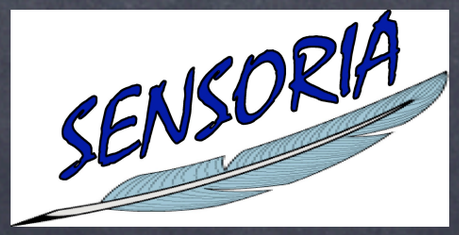
bruni@di.unipi.it

Ivan Lanese

lanese@cs.unibo.it

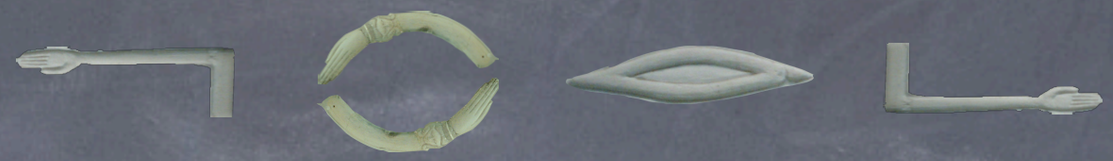
Hernán Melgratti

hmelgra@dc.uba.ar



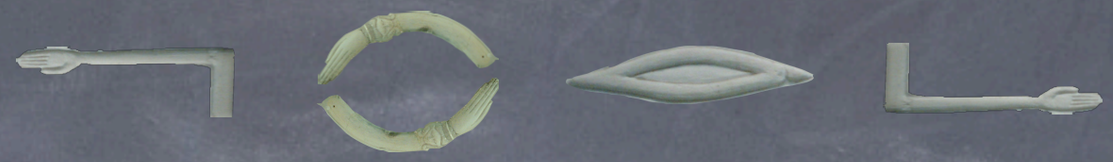
Munich 11-14 March 2008

The problem



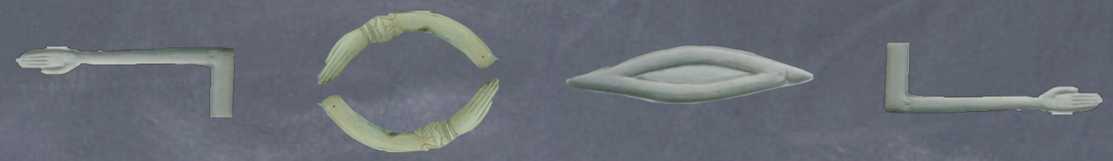
- SOC envisages systems as a combination of services $a_1 \Rightarrow P_1 \mid \ldots \mid a_n \Rightarrow P_n$
- many invocations to each $a_i \Rightarrow P_i$
- each invocation triggers a "new" instance of P_i
- different instances "should not interfere"

Some solutions

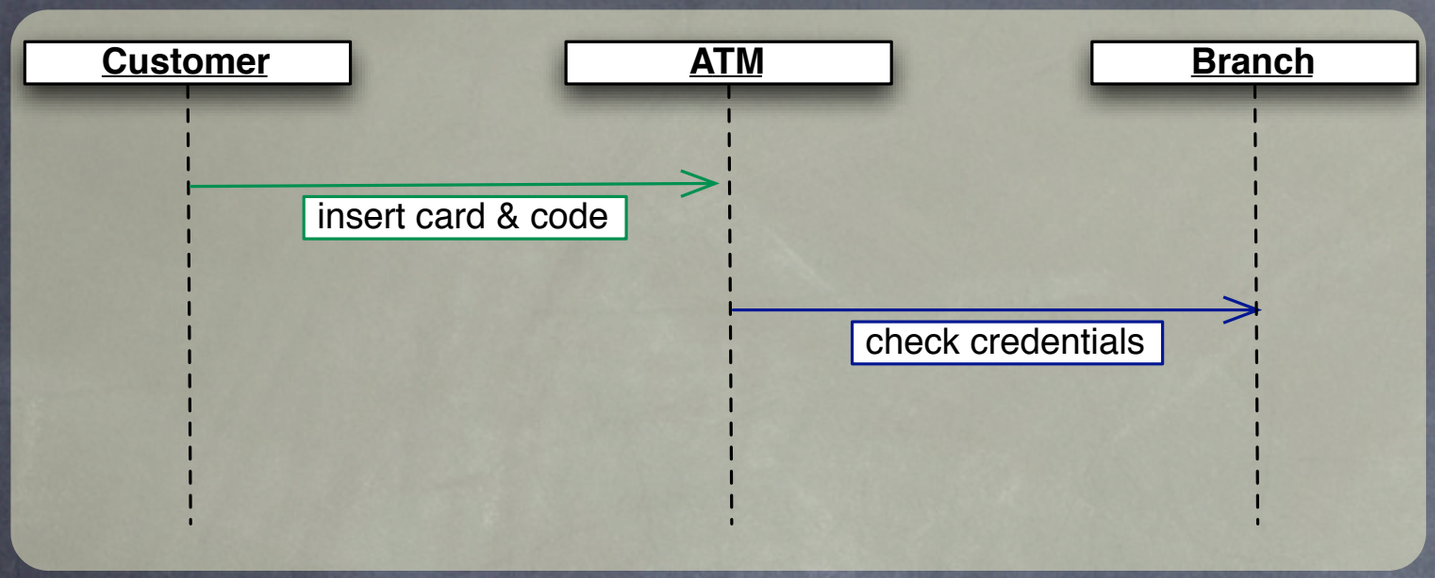


- Standards (WS-BPEL & WS-CDL) use correlation sets
  - too low level and not formally defined
  - reasoning on systems is hard (value-driven interactions)
  - interferences (different instances may use "right" values)
- "Fresh" sessions
  - more formal
  - abstract mechanism for scoping interactions
  - 2-parties / n-parties (n fixed)

Multiparty

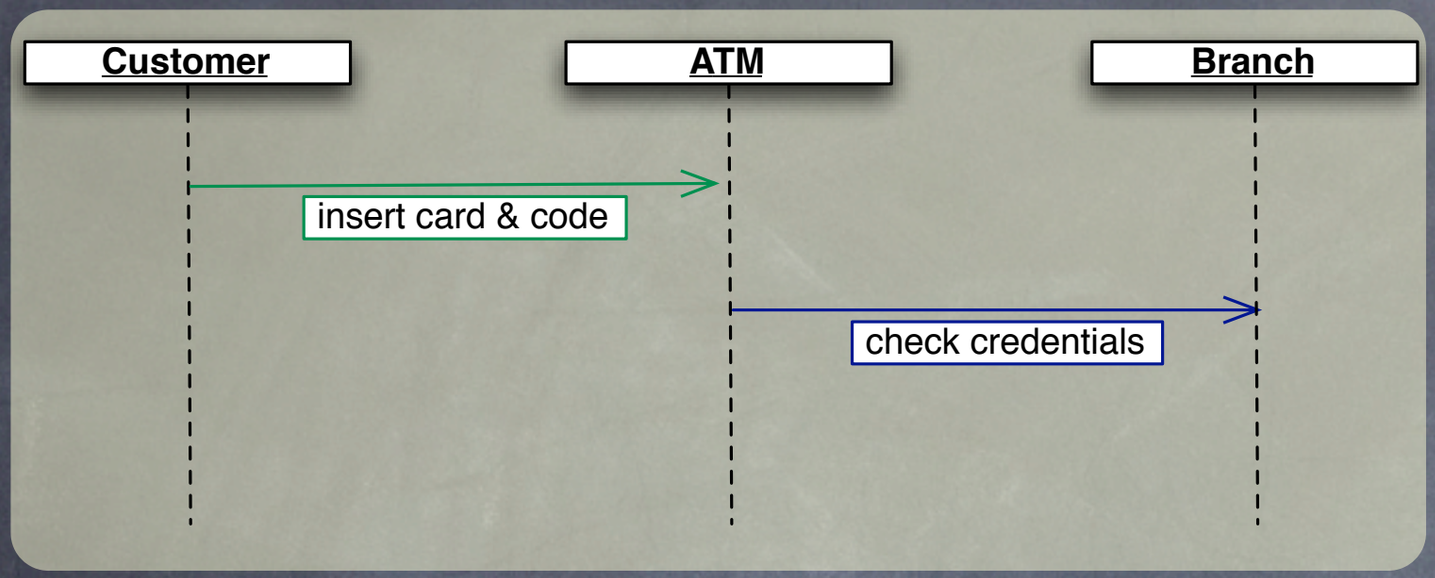


Multiparty



- designed as a 3-party session
- implemented with two 2-party sessions
  - a new session starts between ATM & Branch

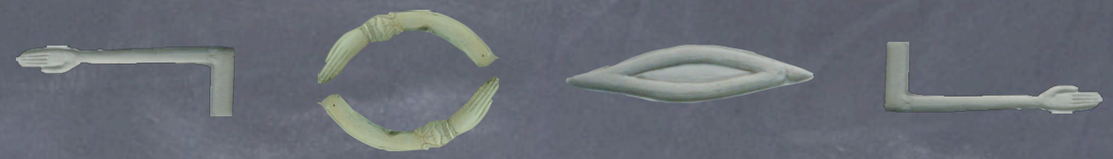
Multiparty



- designed as a 3-party session
- implemented with two 2-party sessions
  - a new session starts between ATM & Branch

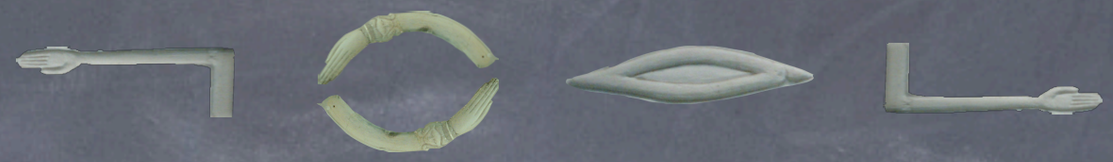
- 1st invoker has to wait
- gamblers dynamically join (and interact)
  - 55 (!) 2-party sessions

Plan



- μse design principles
- Syntax & SOS semantics
- A few interesting (?) examples
- Considerations and concluding remarks

μse design

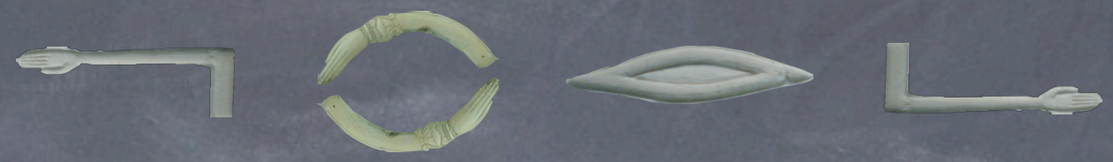


- Sessions unit of conversations among endpoints s▷P
  - session transparency
  - session merging through entry points merge e.Q
- Services a => P
  - invocation≠communication invoke a.Q
  - ephemeral s▷invoke a.Q | a => P ➠ s▷(Q | P)
  - communications (π-like)
    - intra- & extra-sessions
    - locations l::P delimit extra-session communications

µse syntax

S, T	$::=$	$l :: a \Rightarrow P$	Service definition
	$\mid$	$l :: P$	Located process
	$\mid$	$S \mid T$	Composition of systems
	$\mid$	$(\nu n)S$	New name
P, Q	$::=$	$\mathbf{0}$	Empty process
	$\mid$	$\overline{x}w.P$	Intra-session output
	$\mid$	$x(y).P$	Intra-session input
	$\mid$	$x!w.P$	Intra-site output
	$\mid$	$x?(y).P$	Intra-site input
	$\mid$	$\mathsf{install}[a \Rightarrow P].Q$	Service installation
	$\mid$	$\mathsf{invoke}\ a.P$	Service invocation
	$\mid$	$\mathsf{merge}\ e.P$	Entry point
	$\mid$	$r \triangleright P$	Endpoint
	$\mid$	$P \mid Q$	Parallel composition
	$\mid$	$(\nu n)P$	New name
	$\mid$	$\mathrm{rec}\ X.P$	Recursive process
	$\mid$	X	Recursive call

µse syntax



S, T	$::=$	$l :: a \Rightarrow P$	Service definition
	$\mid$	$l :: P$	Located process
	$\mid$	$S \mid T$	Composition of systems
	$\mid$	$(\nu n)S$	New name
P, Q	$::=$	$\mathbf{0}$	Empty process
	$\mid$	$\overline{x}w.P$	Intra-session output
	$\mid$	$x(y).P$	Intra-session input
	$\mid$	$x!w.P$	Intra-site output
	$\mid$	$x?(y).P$	Intra-site input
	$\mid$	$\mathsf{install}[a \Rightarrow P].Q$	Service installation
	$\mid$	$\mathsf{invoke}\ a.P$	Service invocation
	$\mid$	$\mathsf{merge}\ e.P$	Entry point
	$\mid$	$r \triangleright P$	Endpoint
	$\mid$	$P \mid Q$	Parallel composition
	$\mid$	$(\nu n)P$	New name
	$\mid$	$\mathrm{rec}\ X.P$	Recursive process
	$\mid$	X	Recursive call

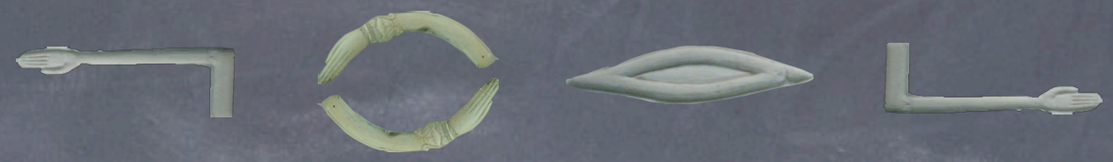
Names

- ✓services
- ✓sessions
- ✓entry points
- ✓locations
- ✓channels

Channels, services and entry points are communicable values

Usual structural congruence

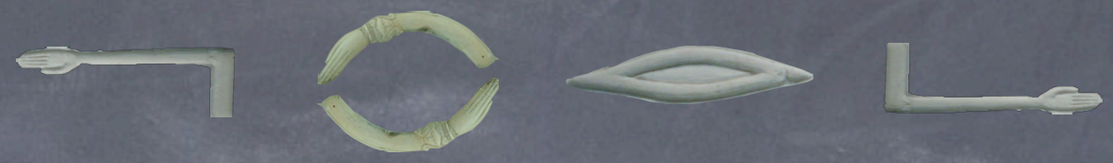
µse syntax



S, T	::=	$l :: a \Rightarrow P$	Service definition
	|	$l :: P$	Located process
	|	$S \mid T$	Composition of systems
	|	$(\nu n)S$	New name
P, Q	::=	$\mathbf{0}$	Empty process
	|	$\bar{x}w.P$	Intra-session output
	|	$x(y).P$	Intra-session input
	|	$x!w.P$	Intra-site output
	|	$x?(y).P$	Intra-site input
	|	$\mathsf{install}[a \Rightarrow P].Q$	Service installation
	|	$\mathsf{invoke}\ a.P$	Service invocation
	|	$\mathsf{merge}\ e.P$	Entry point
	|	$r \triangleright P$	Endpoint
	|	$P \mid Q$	Parallel composition
	|	$(\nu n)P$	New name
	|	$\mathrm{rec}\ X.P$	Recursive process
	|	X	Recursive call

Intra-session

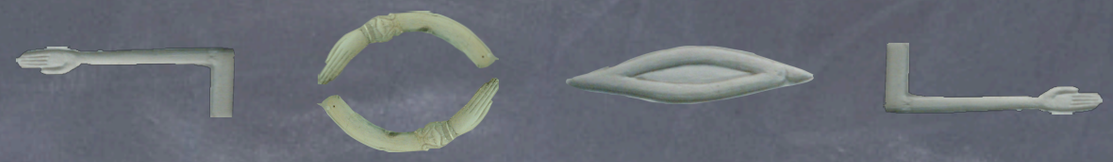
μse syntax



S, T	$::=$	$l :: a \Rightarrow P$	Service definition
	$\mid$	$l :: P$	Located process
	$\mid$	$S \mid T$	Composition of systems
	$\mid$	$(\nu n)S$	New name
P, Q	$::=$	$\mathbf{0}$	Empty process
	$\mid$	$\overline{x}w.P$	Intra-session output
	$\mid$	$x(y).P$	Intra-session input
	$\mid$	$x!w.P$	Intra-site output
	$\mid$	$x?(y).P$	Intra-site input
	$\mid$	$\mathsf{install}[a \Rightarrow P].Q$	Service installation
	$\mid$	$\mathsf{invoke}\ a.P$	Service invocation
	$\mid$	$\mathsf{merge}\ e.P$	Entry point
	$\mid$	$r \triangleright P$	Endpoint
	$\mid$	$P \mid Q$	Parallel composition
	$\mid$	$(\nu n)P$	New name
	$\mid$	$\mathrm{rec}\ X.P$	Recursive process
	$\mid$	X	Recursive call

Extra-session

μse semantics (1)



$$\overline{x}v.P \xrightarrow{\overline{x}v} P \qquad x(y).P \xrightarrow{xv} P\{v/y\}$$

$$x!v.P \xrightarrow{x!v} P \qquad x?(y).P \xrightarrow{x?v} P\{v/y\}$$

$$\text{invoke } a.P \xrightarrow{\perp a} P \qquad \text{install}[a \Rightarrow R].P \xrightarrow{a[R]} P$$

$$\text{merge } e.P \xrightarrow{\bowtie e} P$$

μse semantics (1)

$$\overline{x}v.P \xrightarrow{\overline{x}v} P \qquad x(y).P \xrightarrow{xv} P\{v/y\}$$

$$x!v.P \xrightarrow{x!v} P \qquad x?(y).P \xrightarrow{x?v} P\{v/y\}$$

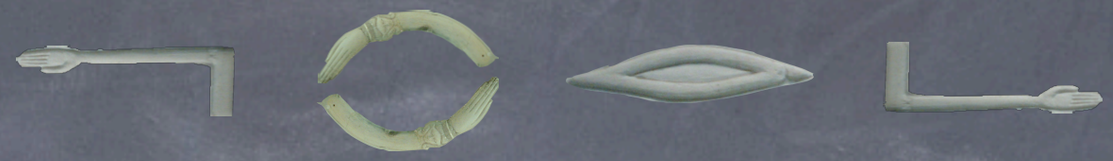
$$\text{invoke } a.P \xrightarrow{\perp a} P \qquad \text{install}[a \Rightarrow R].P \xrightarrow{a[R]} P$$

$$\text{merge } e.P \xrightarrow{\bowtie e} P$$

This is not code mobility;
just services at top level

early-style semantics

μse ATM



$$hiw :: r \triangleright C$$

$$|$$

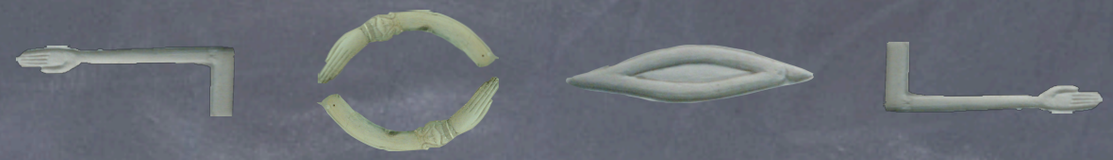
$$(\nu\ check, abort)(hiw :: *atm \Rightarrow A \mid branch :: *bank \Rightarrow B)$$

$$C \quad = \quad \text{invoke } atm.\overline{req}\langle \mathtt{c}, \mathtt{m}\rangle.(cash(x)|sms(y))$$

$$A \quad = \quad req(x,y).\text{invoke } bank.\overline{check}\langle x, y\rangle.(\ check().\overline{cash}\ y + abort().\overline{cash}\ 0\ )$$

$$B \quad = \quad check(x,y).\textbf{if } ok(x,y) \textbf{ then } \overline{check}.\overline{sms}\ \text{"}\mathtt{ok}\text{"} \textbf{ else } \overline{abort}.\overline{sms}\ \text{"}\mathtt{ko}\text{"}$$

μse ATM



$$hiw :: r \triangleright C$$

$$(\nu\ check, abort)(hiw :: *atm \Rightarrow A \mid branch :: *bank \Rightarrow B)$$

$$C \quad = \quad \text{invoke } atm.\overline{req}\langle \mathtt{c}, \mathtt{m}\rangle.(cash(x) \mid sms(y))$$

$$A \quad = \quad req(x,y).\text{invoke } bank.\overline{check}\langle x, y\rangle.(\ check().\overline{cash}\ y + abort().\overline{cash}\ 0\ )$$

$$B \quad = \quad check(x,y).\textbf{if}\ ok(x,y)\ \textbf{then}\ \overline{check}.\overline{sms}\ \text{"}\mathtt{ok}\text{"}\ \textbf{else}\ \overline{abort}.\overline{sms}\ \text{"}\mathtt{ko}\text{"}$$

μse ATM

$$hiw :: r \triangleright C$$

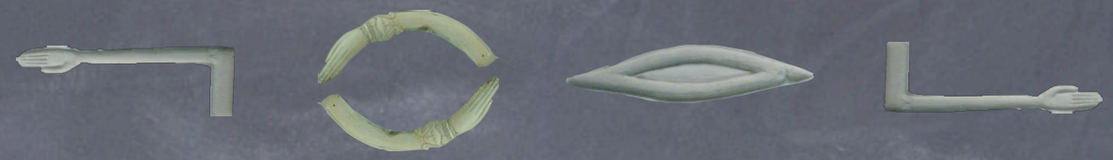
$$(\nu\ check, abort)(hiw :: *atm \Rightarrow A \mid branch :: *bank \Rightarrow B)$$

$$C \quad = \quad \text{invoke } atm.\overline{req}\langle \mathtt{c}, \mathtt{m}\rangle.(cash(x)|sms(y))$$

$$A \quad = \quad req(x,y).\text{invoke } bank.\overline{\overline{check}}\langle x, y\rangle.(\ check().\overline{\overline{cash}}\ y + abort().\overline{\overline{cash}}\ 0\ )$$

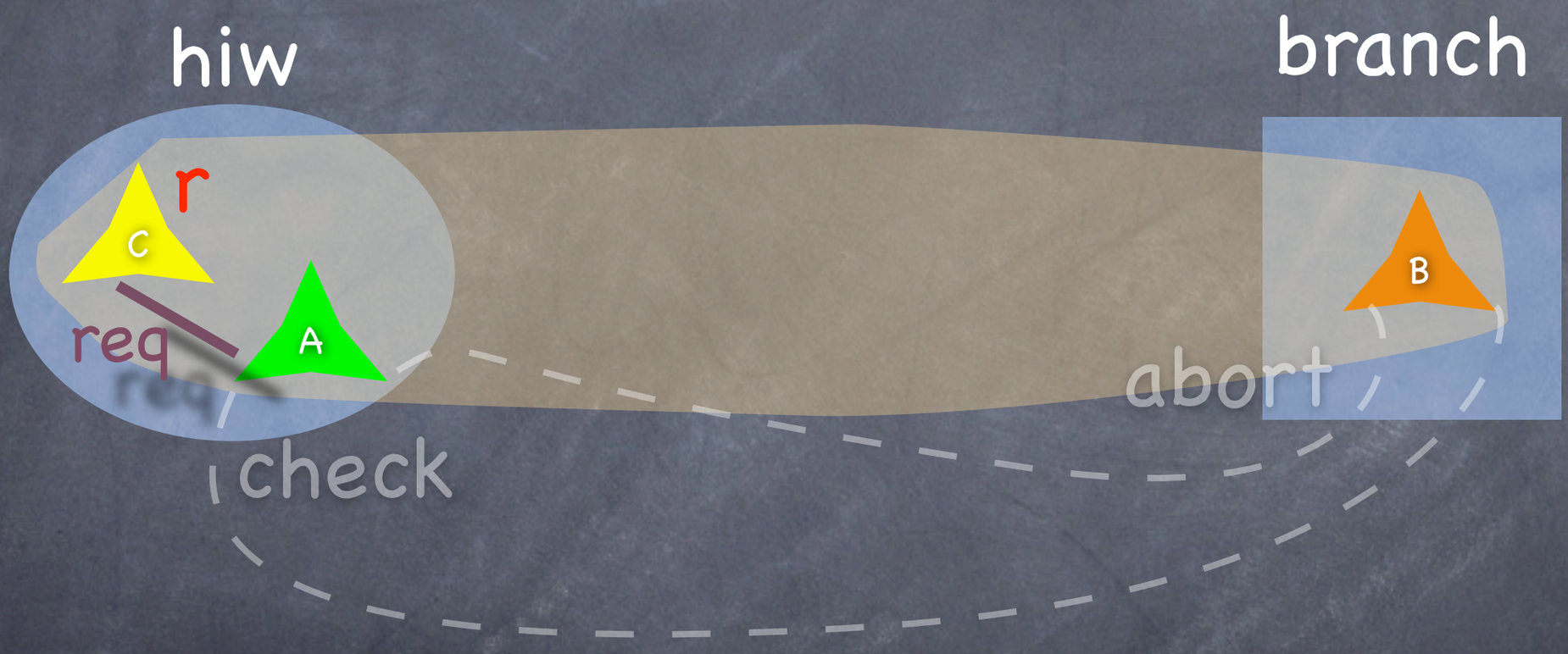
$$B \quad = \quad check(x,y).\textbf{if } ok(x,y) \textbf{ then } \overline{\overline{check}.\overline{sms}}\ \text{"}\mathtt{ok}\text{"} \textbf{ else } \overline{\overline{abort}.\overline{sms}}\ \text{"}\mathtt{ko}\text{"}$$

μse ATM



$$hiw :: r \triangleright C$$

$$(\nu\ check, abort)(hiw :: *atm \Rightarrow A \mid branch :: *bank \Rightarrow B)$$

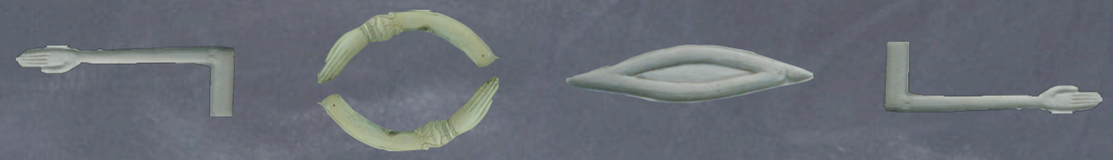


$$C \quad = \quad \text{invoke } atm.\overline{req}\langle \mathtt{c}, \mathtt{m} \rangle.(cash(x) | sms(y))$$

$$A \quad = \quad req(x, y).\text{invoke } bank.\overline{check}\langle x, y \rangle.(\ check().\overline{cash}\ y + abort().\overline{cash}\ 0\ )$$

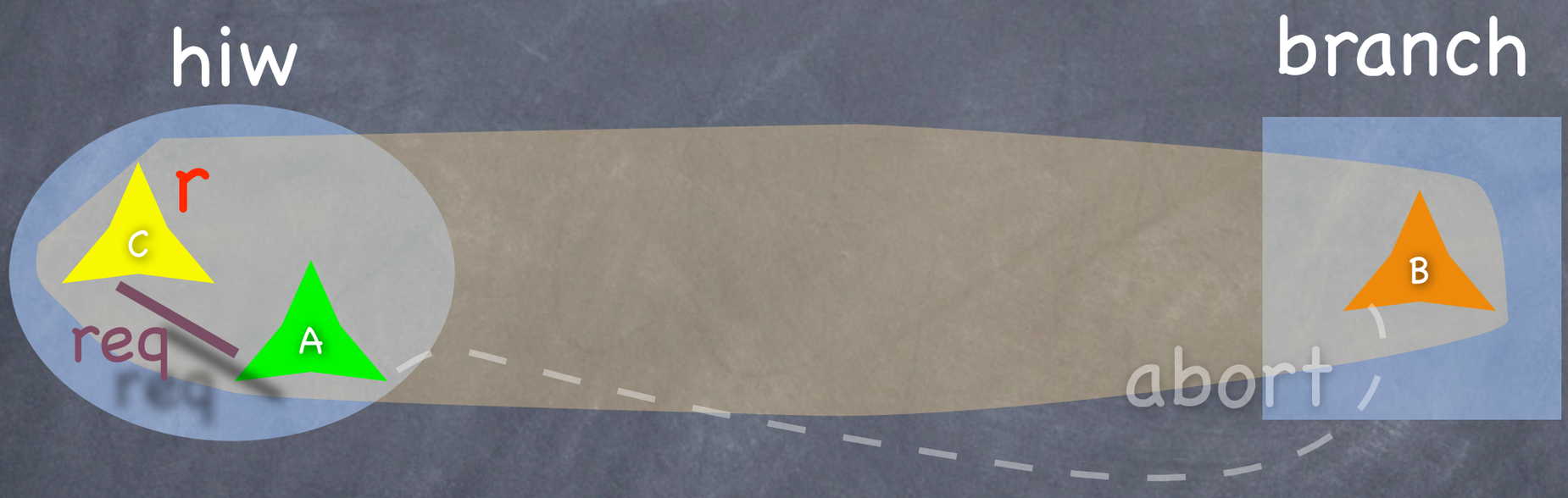
$$B \quad = \quad check(x, y).\textbf{if}\ ok(x, y)\ \textbf{then}\ \overline{check}.\overline{sms}\ \text{"}\mathtt{ok}\text{"}\ \textbf{else}\ \overline{abort}.\overline{sms}\ \text{"}\mathtt{ko}\text{"}$$

μse ATM



$$hiw :: r \triangleright C$$

$$(\nu\ check, abort)(hiw :: *atm \Rightarrow A \mid branch :: *bank \Rightarrow B)$$



$$C \quad = \quad \text{invoke } atm.\overline{req}\langle \mathtt{c}, \mathtt{m}\rangle.(cash(x)|sms(y))$$

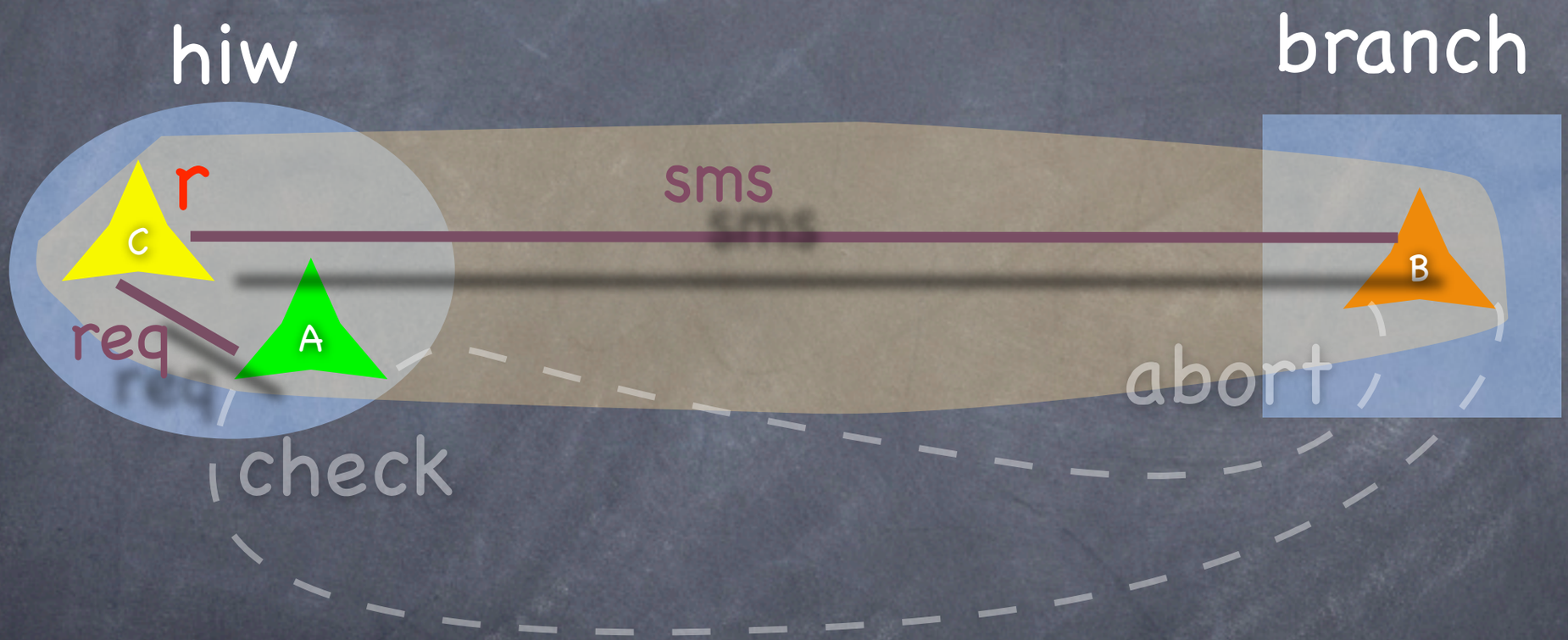
$$A \quad = \quad req(x,y).\text{invoke } bank.\overline{check}\langle x, y\rangle.(\ check().\overline{cash}\ y + abort().\overline{cash}\ 0\ )$$

$$B \quad = \quad check(x,y).\textbf{if } ok(x,y) \textbf{ then } \overline{check}.\overline{sms}\ \text{"}\mathtt{ok}\text{"} \textbf{ else } \overline{abort}.\overline{sms}\ \text{"}\mathtt{ko}\text{"}$$

μse ATM

$$hiw :: r \triangleright C$$

$$(\nu\ check, abort)(hiw :: *atm \Rightarrow A \mid branch :: *bank \Rightarrow B)$$



$$C \quad = \quad \text{invoke } atm.\overline{req}\langle \mathtt{c}, \mathtt{m}\rangle.(cash(x) | sms(y))$$

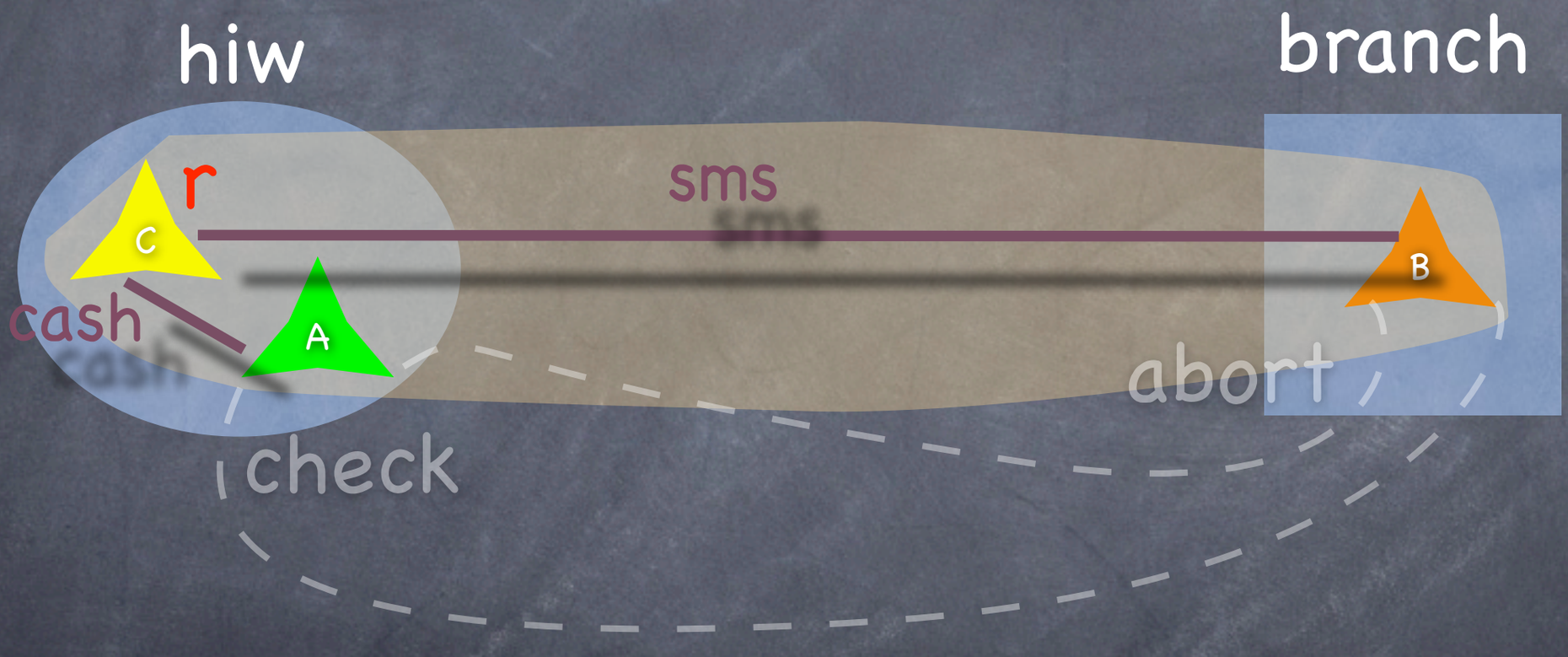
$$A \quad = \quad req(x,y).\text{invoke } bank.\overline{check}\langle x, y\rangle.(\ check().\overline{cash}\ y + abort().\overline{cash}\ 0\ )$$

$$B \quad = \quad check(x,y).\textbf{if } ok(x,y) \textbf{ then } \overline{check}.\overline{sms}\ \text{"}\mathtt{ok}\text{"} \textbf{ else } \overline{abort}.\overline{sms}\ \text{"}\mathtt{ko}\text{"}$$

μse ATM

$$hiw :: r \triangleright C$$

$$(\nu\ check, abort)(hiw :: *atm \Rightarrow A \mid branch :: *bank \Rightarrow B)$$

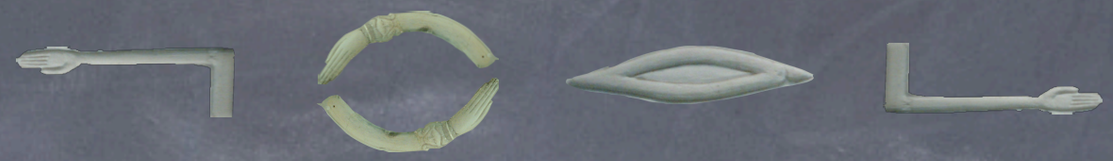


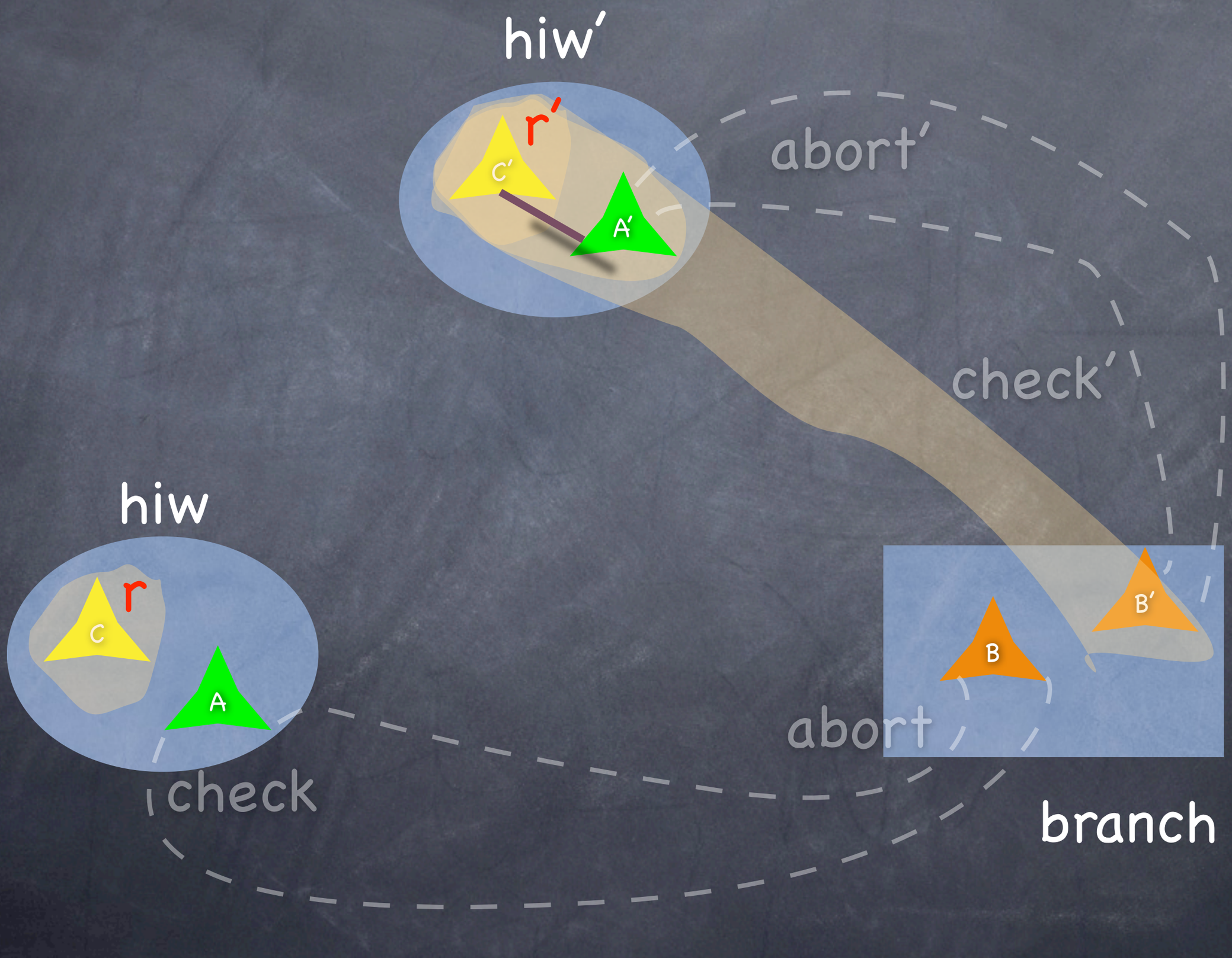
$$C \quad = \quad \text{invoke } atm.\overline{req}\langle \mathtt{c}, \mathtt{m}\rangle.(cash(x) \mid sms(y))$$

$$A \quad = \quad req(x,y).\text{invoke } bank.\overline{check}\langle x, y\rangle.(\ check().\overline{cash}\ y + abort().\overline{cash}\ 0\ )$$

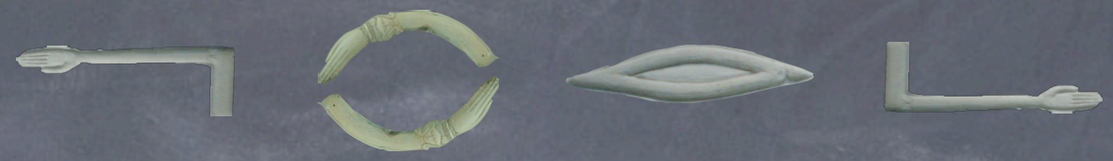
$$B \quad = \quad check(x,y).\textbf{if } ok(x,y) \textbf{ then } \overline{check}.\overline{sms}\ \text{"}\mathtt{ok}\text{"} \textbf{ else } \overline{abort}.\overline{sms}\ \text{"}\mathtt{ko}\text{"}$$

Real life is harder

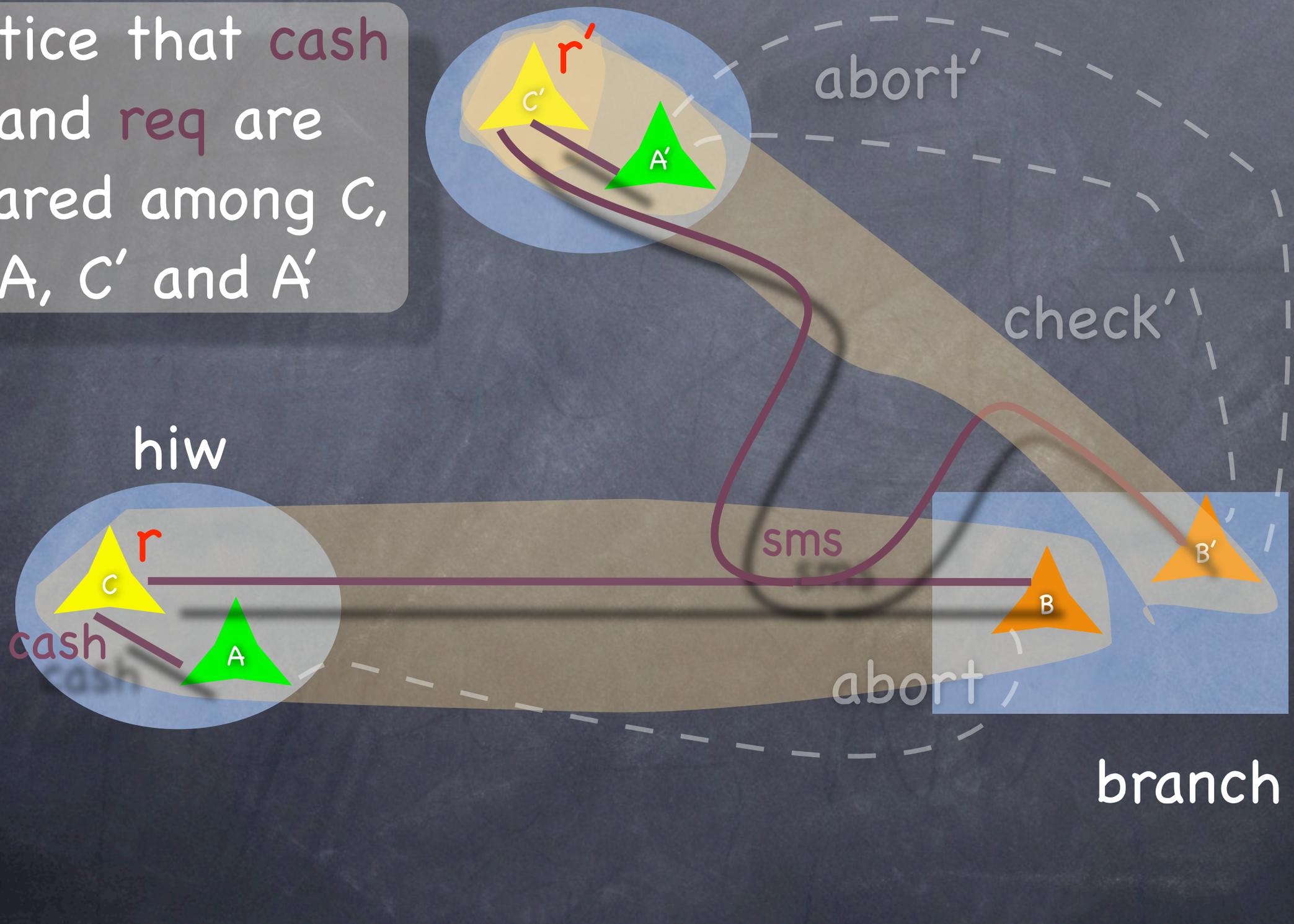




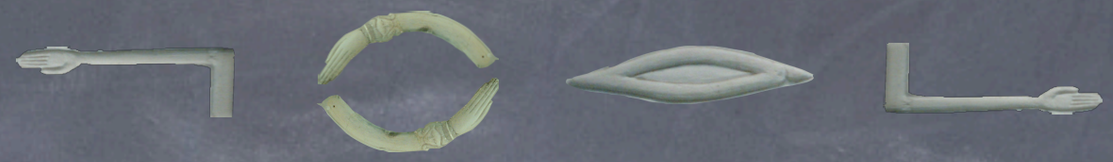
Real life is harder



Notice that cash and req are shared among C, A, C′ and A′



Play time



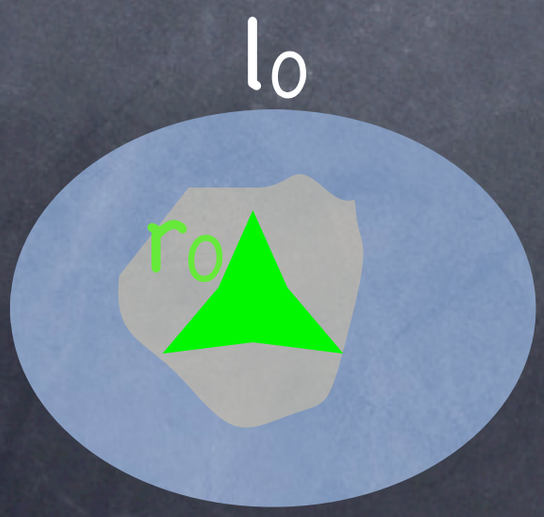
$l :: s \Rightarrow \mathsf{merge}\ e.\ \overline{start}.\ \mathsf{rec}\ X.(\mathsf{merge}\ e.X)$

$\mid$

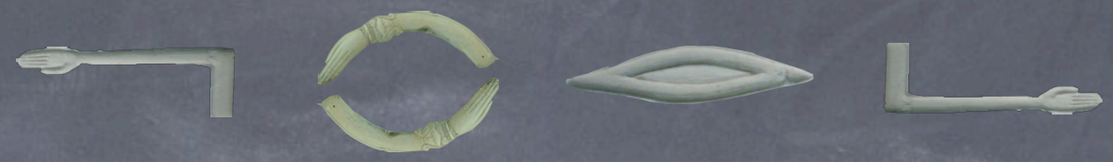
$\mathsf{install}[*s \Rightarrow \mathsf{merge}\ e.\overline{start}]$

$l_0 :: \quad r_0 \triangleright \mathsf{invoke}\ s.\ start().\ P$

$l_1 :: \quad r_1 \triangleright \mathsf{invoke}\ s.\ start().\ P$



Play time



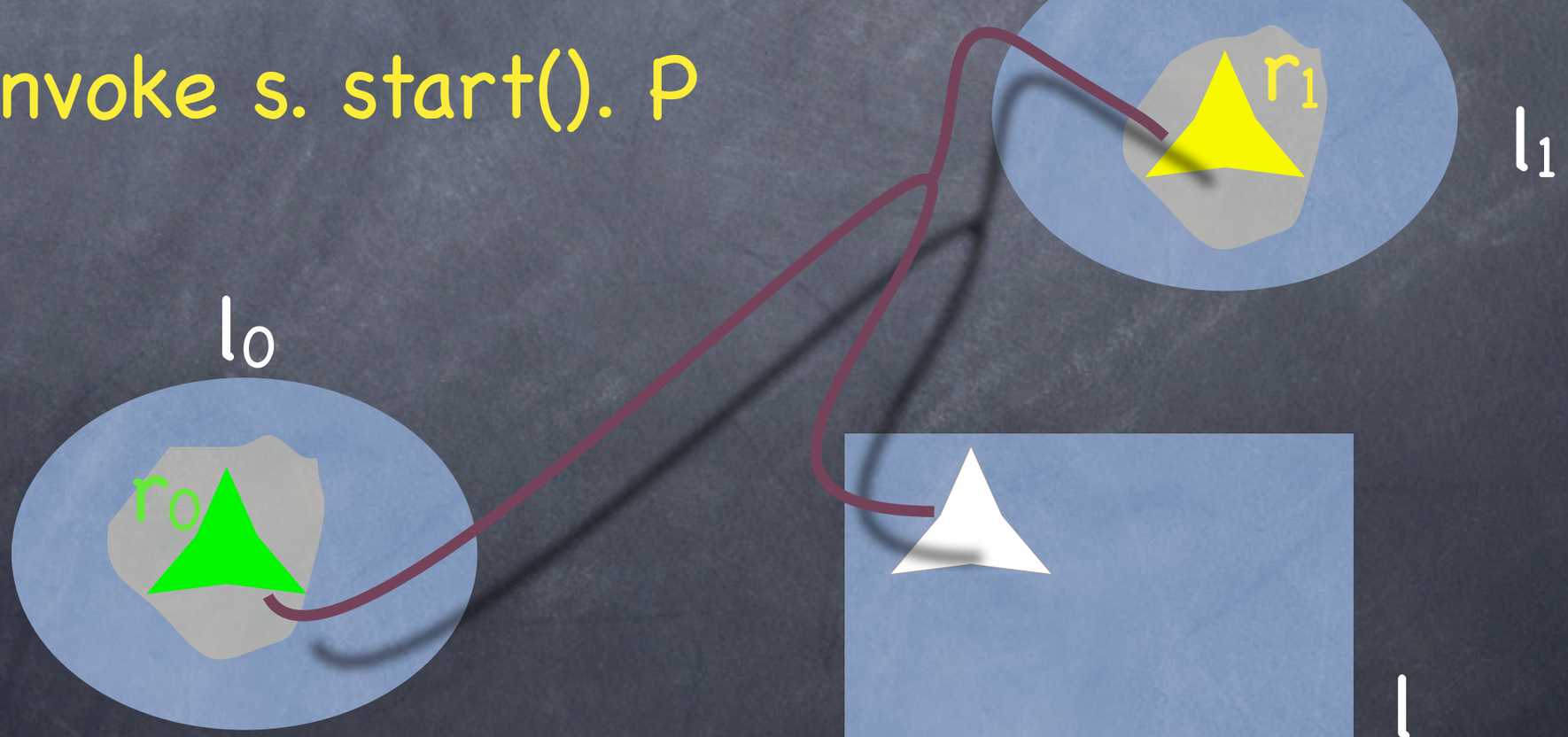
l :: s => merge e. $\overline{\text{start}}$. rec X.(merge e.X)

|

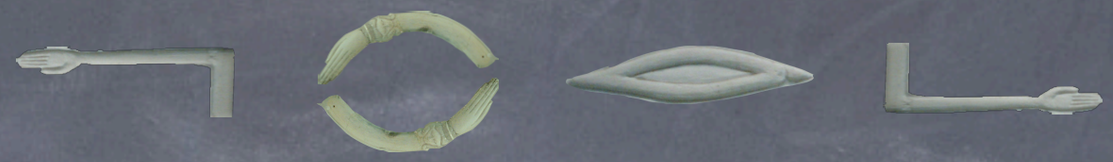
install [*s => merge e.$\overline{\text{start}}$]

l_0 :: r_0 ▷ invoke s. start(). P

l_1 :: r_1 ▷ invoke s. start(). P



Play time



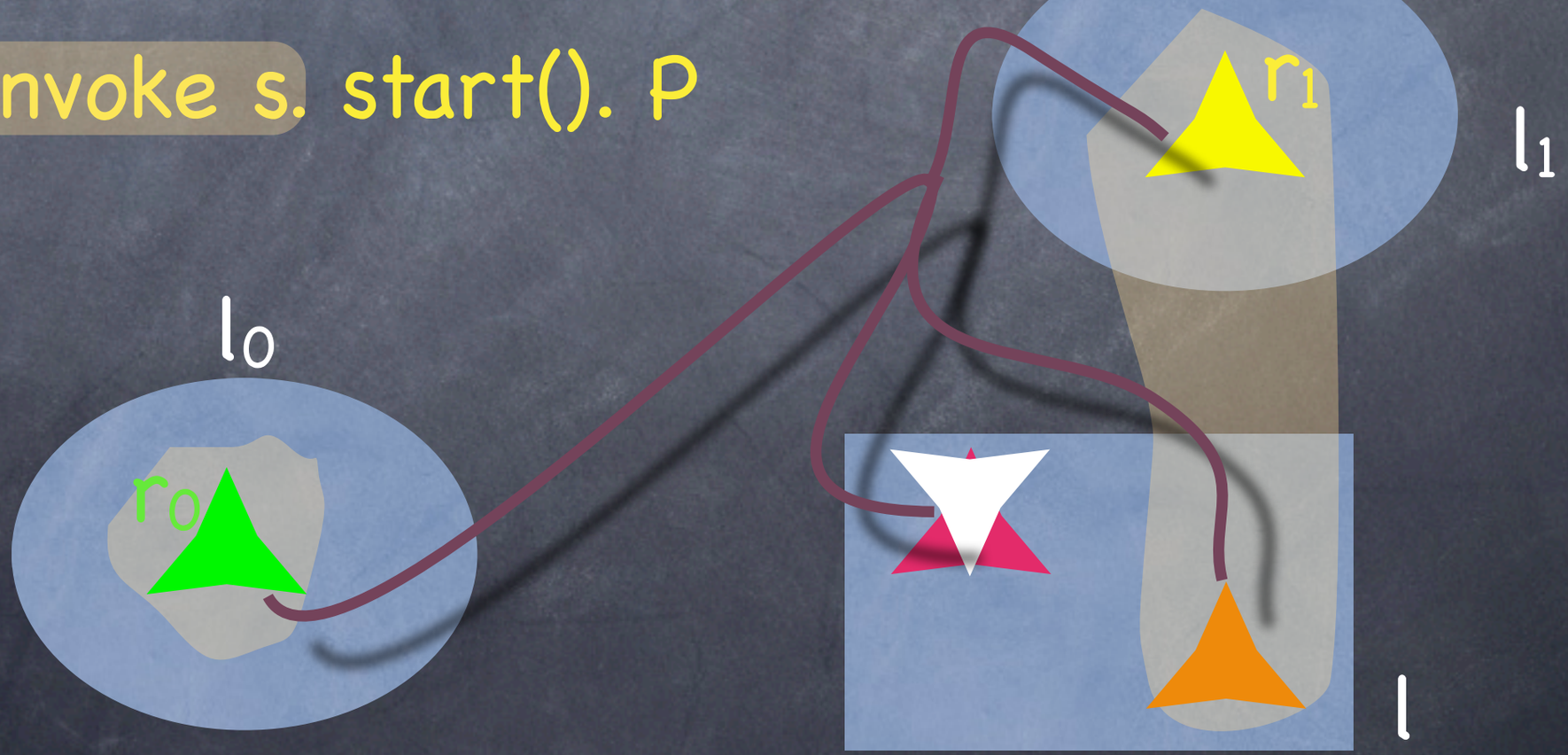
$l :: s \Rightarrow \text{merge } e.\ \overline{start}.\ \text{rec } X.(\text{merge } e.X)$

$\mid$

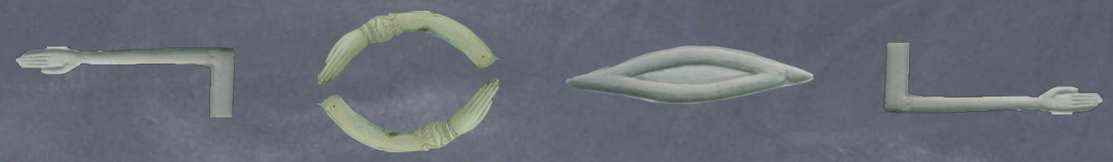
$\text{install}[*s \Rightarrow \text{merge } e.\overline{start}]$

$l_0 :: r_0 \triangleright \text{invoke } s.\ start().\ P$

$l_1 :: r_1 \triangleright \text{invoke } s.\ start().\ P$



Play time



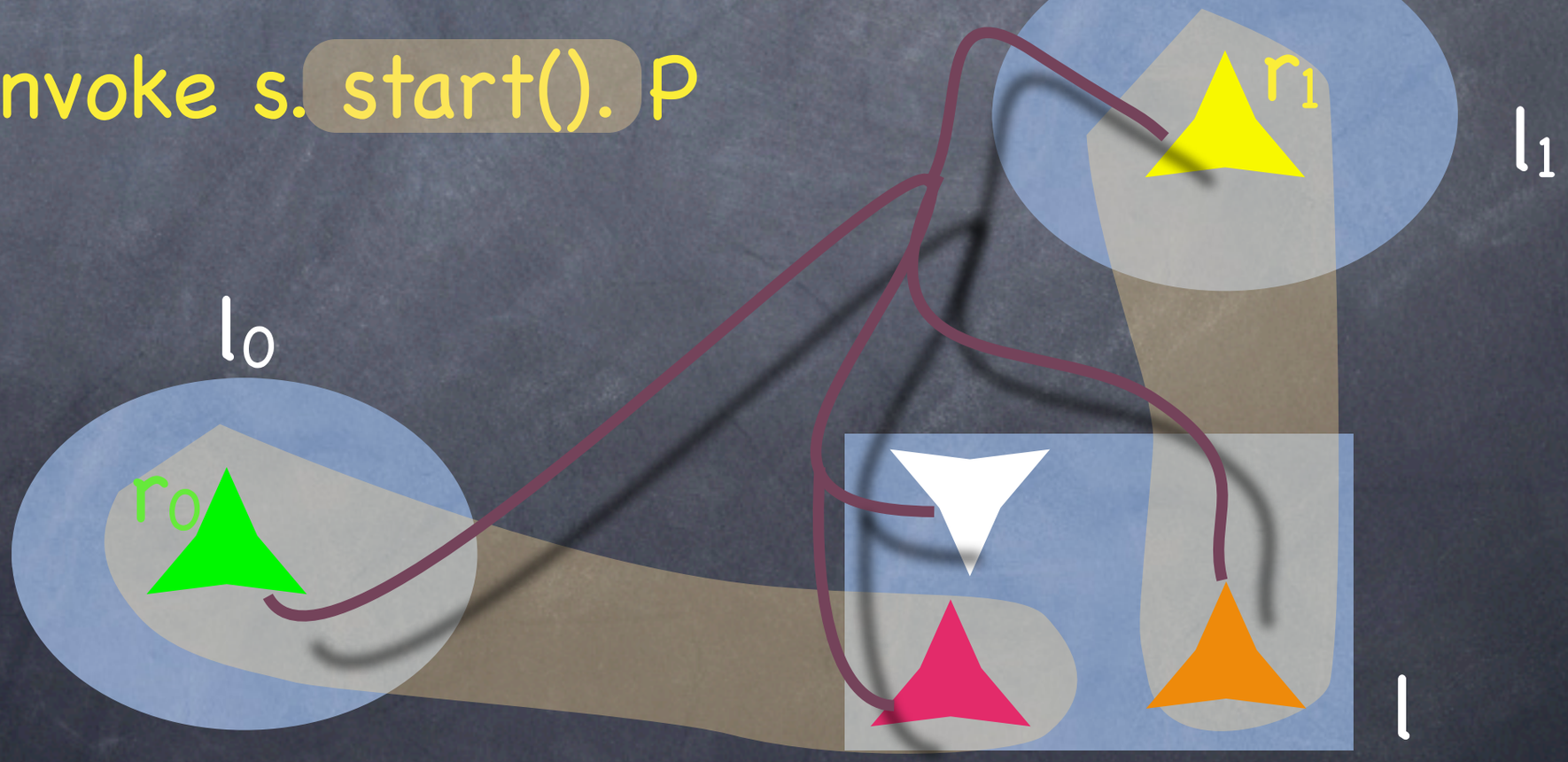
$l :: s \Rightarrow \text{merge } e.\ \overline{start}.\ \text{rec } X.(\text{merge } e.X)$

|

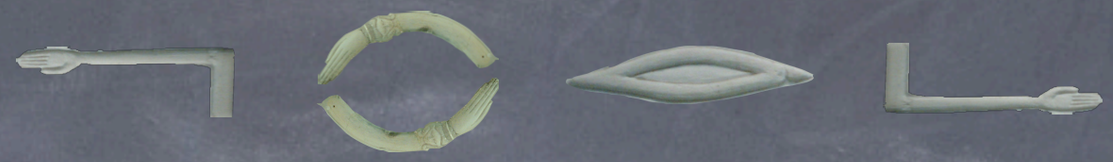
install [*s => merge e.start]

$l_0 :: \quad r_0 \triangleright \text{invoke } s.\ start().\ P$

$l_1 :: \quad r_1 \triangleright \text{invoke } s.\ start().\ P$



Play time



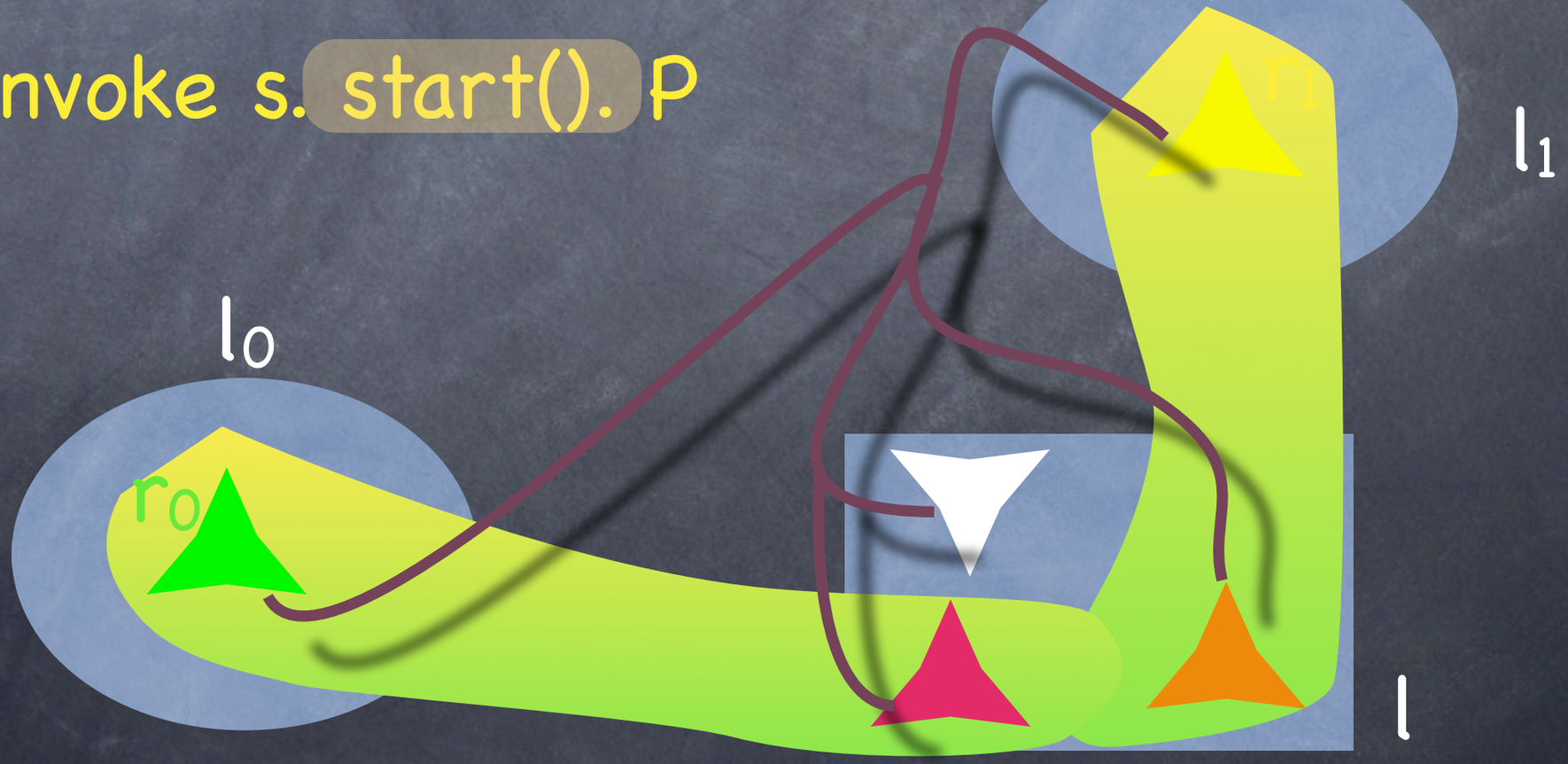
l :: s => merge e. $\overline{\text{start}}$. rec X.(merge e.X)

|

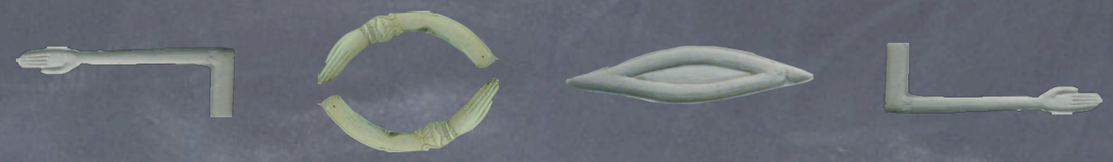
install [*s => merge e.$\overline{\text{start}}$]

l_0 :: r_0 ▷ invoke s. start(). P

l_1 :: r_1 ▷ invoke s. start(). P



Play time



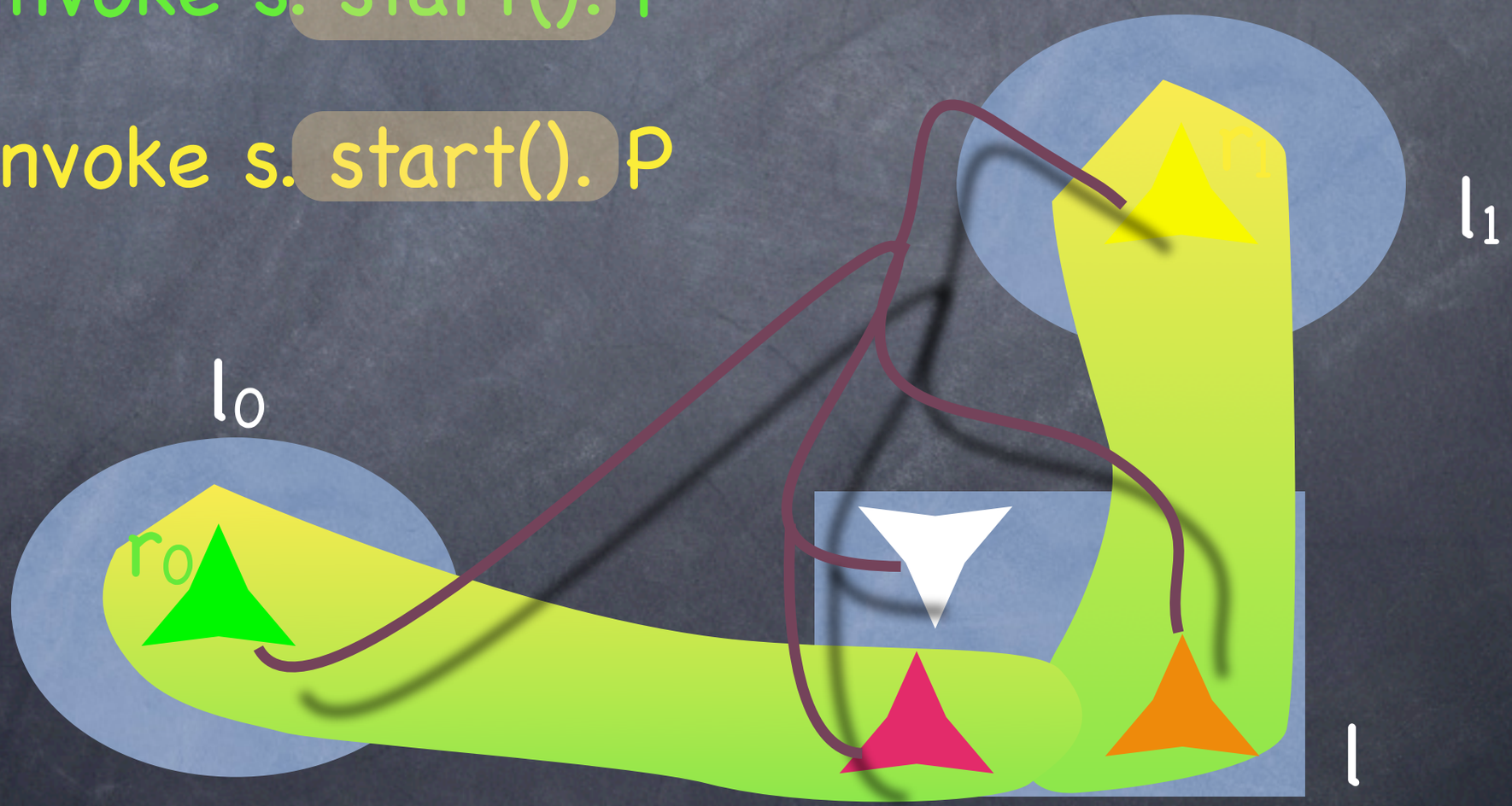
l :: s => merge e. start. rec X.(merge e.X)

|

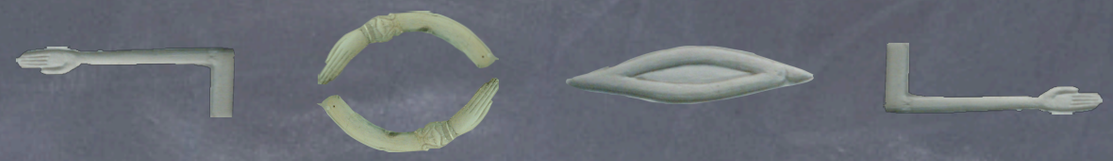
install [*s => merge e.start]

l_0 :: r_0 ▷ invoke s. start(). P

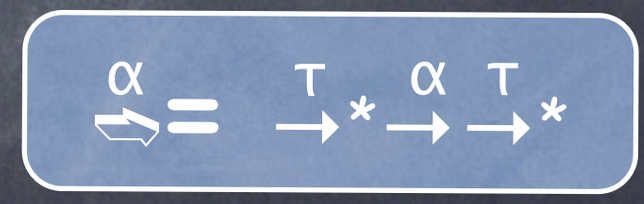
l_1 :: r_1 ▷ invoke s. start(). P



μse bisimulation

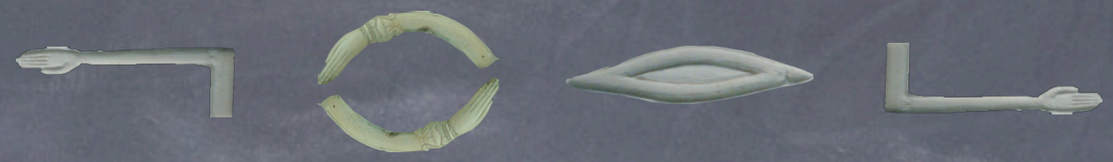


- A binary relation B on μse systems is a weak bisimulation if
  - B is symmetric
  - whenever $(S, T) \in B$, for each transition $S \xrightarrow{\alpha} S'$ such that $bn(\alpha) \cap fn(T) = \varnothing$, there is a transition $T \stackrel{\alpha}{\Rightarrow} T'$ and $(S', T') \in B$



- Bisimilarity is the largest bisimulation

Bisimulation at work



Specification

$$l :: *a \Rightarrow data(\mathtt{x}).\overline{ret}\ \mathtt{fun}(\mathtt{x})$$

Implementation 1

$$l :: (\nu a_1, a_2)\big((\nu av)(*a \Rightarrow av?(u).\mathsf{invoke}\ u) \mid$$

$$\mathrm{rec}\ X.av!a_1.X \mid \mathrm{rec}\ X.av!a_2.X$$

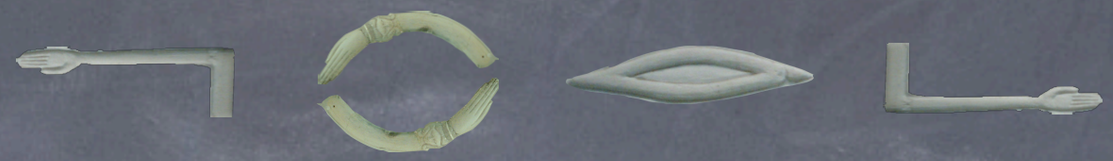
$$*a_1 \Rightarrow data(\mathtt{x}).\overline{ret}\ \mathtt{fun}(\mathtt{x}) \mid *a_2 \Rightarrow data(\mathtt{x}).\overline{ret}\ \mathtt{fun}(\mathtt{x})\big)$$

Implementation 2

$$(\nu e)l :: a \Rightarrow \mathrm{rec}\ Y.(\mathsf{merge}\ e.\mathsf{install}[a \Rightarrow Y]) \mid$$

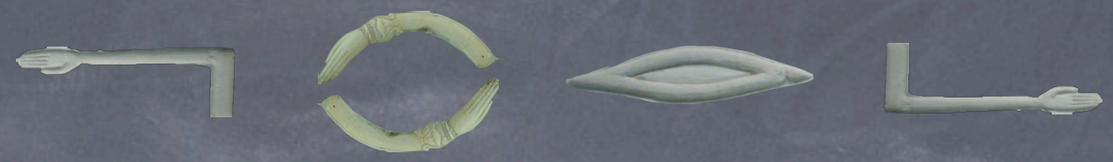
$$\mathrm{rec}\ X.(\nu r)r \rhd \mathsf{merge}\ e.(data(\mathtt{x}).\overline{ret}\ \mathtt{fun}(\mathtt{x}) \mid X)$$

Conclusions



- SCC
  - similar primitive for service invocation
  - only 2-party sessions
  - on service invocation, both client and service instance are in a freshly generated session
- Bonelli, Compagnoni (TGC07)
  - correspondence assertions to relate many 2-party sessions
- Carbone, Honda, Yoshida (POPL08)
  - statically fixed number of participants
  - delegation
  - (distributed) randez-vous
- Caires, Viera, Seco (ESOP08) conversation calculus
  - exception handling
  - nesting of sessions

Future directions



- Choreographic view of dynamic multiparty sessions
- Session types for controlling progress properties of multiparty sessions (conditional liveness)
- "Sophisticated" communication primitives (e.g., multi/broad-cast)
- Implementation
  - Clustering P2P networks with μse's primitives