Compositional relational reasoning via operational game semantics

Guilhem Jaber (Univ. Nantes) & Andrzej Murawski (Univ. Oxford)

Concurrent Game Café

Motivation:

Compositional reasoning on higher-order programs
with shared ressources, that is modular w.r.t. effects

Realizability/Logical Relations?!

Advantages:

- provides models of type systems and program logics;
- defined directly using operational semantics;
- captures abstraction properties likes parametricity;
- provides Kripke-style reasoning (a.k.a forcing/presheaves construction) on shared ressources
- modular wrt observation (biorthogonality).

Realizability/Logical Relations?!

Advantages:

- provides models of type systems and program logics;
- defined directly using operational semantics;
- captures abstraction properties likes parametricity;
- provides Kripke-style reasoning (a.k.a forcing/presheaves construction) on shared ressources
- modular wrt observation (biorthogonality).

Drawbacks:

- complex inductive definition (step-indexing);
- extensional (no clear distinction between programs and environments);
- full-abstraction only via biorthogonality;
- hard to automate.

We Want Game Semantics!

Models built on representations of the interactions between the program and its environment...

We Want Game Semantics!

Models built on representations of the interactions between the program and its environment...

... but with some specific features:

- defined from operationnal semantics
- provide coinductive and Kripke-stype reasoning
- handle asymmetric settings (Programs and Environments written in different languages)

Operational Game Semantics!

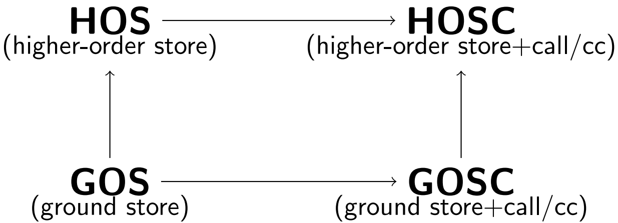
What's in this talk ?

- Fully-abstract operational game models
  - For simply-typed call-by-value λ-calculus with higher-order references and call/cc
  - Modular reasoning on the power of Opponent based on a asymmetric & uniform presentation of visibility and well-bracketing.

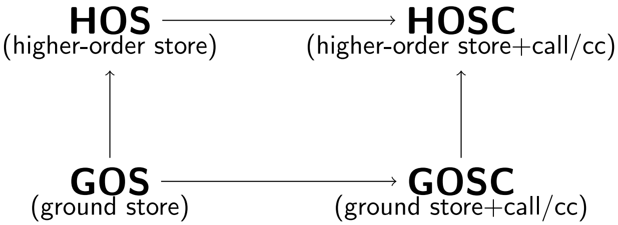
- Kripke normal-form bisimulations
  - Complete & tractable technique for proving contextual equivalence
  - Compositional reasoning on shared ressource (i.e. references)

Effectful fragments of ML

- Simply-typed call-by-value λ-calculus
- with references: ground (can store integers or locations) or higher-order (can also store functions)
- with or without call/cc control operator



Semantic studies



For these four languages we design:

- a fully-abstract model using **operational game semantics**
- a complete operational technique for proving contextual equivalence using **Kripke open bisimulations**

1. Operational Semantics

2. Operational Game Semantics

3. Kripke Normal-Form Bisimulations

Operational Semantics

We use *reduction semantics*:

- reduction relation $M \to M'$ between terms
- closed by evaluation contexts:

$$\frac{M \to M'}{K[M] \to K[M']}$$

In call-by-value:

$$\begin{array}{rcl} \text{Values } V & \triangleq & x \mid \lambda x.M \\ \text{Terms } M & \triangleq & V \mid MN \\ \text{Evaluation Contexts } K & \triangleq & \bullet \mid VK \mid KM \end{array}$$

Reduction relation

$$(\lambda x.M)V \to M\{V/x\}$$

Contextual equivalence

Definition

Two programs M_1, M_2 are contextually equivalent when for all context C, $C[M_1]$ and $C[M_2]$ are *observationally indistinguishable*.

This definition depends on:

- The language the contexts are written in
  - $\leadsto$ in general the same as the one of M_1, M_2;
  - $\leadsto$ In our CBV λ-calculus : $C \triangleq \bullet \mid \lambda x.C \mid MC \mid CM$
- the observation that is used
  - $\leadsto$ termination: reduces to a value
  - $\leadsto$ error: reduces to $K[err()]$ with err a distinguished, free variable.

Contextual equivalence

Definition

Two programs M_1, M_2 are contextually equivalent when for all context C, $C[M_1]$ and $C[M_2]$ are *observationally indistinguishable*.

This definition depends on:

- The language the contexts are written in
  - $\rightsquigarrow$ in general the same as the one of M_1, M_2;
  - $\rightsquigarrow$ In our CBV λ-calculus : $C \triangleq \bullet \mid \lambda x.C \mid MC \mid CM$
- the observation that is used
  - $\rightsquigarrow$ termination: reduces to a value
  - $\rightsquigarrow$ error: reduces to $K[err()]$ with *err* a distinguished, free variable.

Talcott's *CIU-equivalence*:

- only consider evaluation contexts;
- substitute free variables with values;

CIU and contextual equivalence are equal (in a symmetric setting) !

Operational Semantics for references

- We consider heaps: partial maps from *locations* ℓ to values.
- Evaluation reduction works on pairs (M, h):

$$\begin{array}{lcl} (\mathrm{ref}\, V, h) & \to & (\ell, h \cdot [\ell \mapsto V]) \\ (\ell := V, h) & \to & ((), h[\ell \mapsto V]) \\ (!\ell, h) & \to & (h(\ell), h) \end{array}$$

New evaluation contexts:

$$K \triangleq \ldots \mid \mathrm{ref}\, K \mid K := M \mid \ell := K \mid !K$$

Operational Semantics for call/cc

We embed evaluation contexts into terms using $\mathrm{cont}\ K$ construct.

$$
\begin{array}{lcl}
(K[\mathrm{call/cc}(x.M)], h) & \rightarrow & (K[M\{\mathrm{cont}\ K/x\}], h) \\
(K[\mathrm{throw\ V\ to\ cont}\ K'], h) & \rightarrow & (K'[V], h)
\end{array}
$$

1. Operational Semantics

2. Operational Game Semantics

3. Kripke Normal-Form Bisimulations

Introducing Operational Game Semantics

Interactions between the program and the context are represented by traces generated by a bipartite labelled transition system.

- Bipartite: Program=Player; Context=Opponent
- Player behavior is fully determined by the program;
- Opponent behavior represents all possible contexts.

- Functional values and continuations are represented using free variables called names;
- Configurations of the LTS have a dynamic environment ("inventory") γ that keeps track of the functional values and continuations associated to these names.

Exchanging Values

Player and Opponent exchanges *abstract values*:

$$A, B \triangleq () \mid \mathsf{tt} \mid \mathsf{ff} \mid n \mid f \mid \langle A, B \rangle$$

- avoid being too intentional
- negative values are represented by names: they are opaque
  - functional names f
- similar to Levy's ultimate patterns.

- In this talk: no exchange of locations or continuations !

Abstracting values:

$$(A, \gamma) \in \mathsf{AVal}_\sigma(V) \text{ when } A : \sigma \text{ and } A\{\gamma\} = V$$

Actions

Four kind of actions

- Player Answer $\bar{c}(A)$: an abstract value A is sent through a continuation name c.
- Player Question $\bar{f}(A, c)$: an abstract value A and a continuation name c are sent through a function name f.
- Opponent Answer (OA) $c(A)$: an abstract value A is received from the environment via the continuation name c.
- Opponent Question (OQ) $f(A, c)$: an abstract value A and a continuation name c are received from the environment through a function name f.

Traces

Traces are sequence of actions alternating between Player and Opponent

- similar to traces generated by the (Internal) π-calculus;
- justification pointer used in game semantics can be reconstructed from the binding structure of actions;
- removing continuation names in traces for HOS: back to direct style.

Configurations

The state of the bipartite LTS are:

- Player (active) configurations $\langle M, c, \gamma, \xi, h \rangle$;
- Opponent (passive) configurations $\langle \gamma, \xi, h \rangle$.

They are formed by:

- a term M;
- a continuation name c;
- an dynamic environment γ:
  a map from names to functional values or evaluation contexts;
- a continuation structure ξ:
  a map from continuation names to continuation names;
- a heap h.

LTS for HOSC

$$
\begin{array}{l|l}
(P\tau) & \langle M, c, \gamma, \xi, h\rangle \quad \xrightarrow{\tau} \quad \langle N, c', \gamma, \xi, h'\rangle \\
& \text{when } (M, c, h) \to (N, c', h') \\[1em]
(PA) & \langle V, c, \gamma, \xi, h\rangle \quad \xrightarrow{\bar{c}(A)} \quad \langle \gamma \cdot \gamma', \xi, h\rangle \\
& \text{when } c : \sigma \text{ and } (A, \gamma') \in \mathsf{AVal}_\sigma(V) \\[1em]
(PQ) & \langle K[fV], c, \gamma, \xi, h\rangle \quad \xrightarrow{\bar{f}(A,c')} \quad \langle \gamma \cdot \gamma' \cdot [c' \mapsto K], \xi \cdot [c' \mapsto c], h\rangle \\
& \text{when } c' : \sigma', f : \sigma \to \sigma' \text{ and } (A, \gamma') \in \mathsf{AVal}_\sigma(V), \\[1em]
(OA) & \langle \gamma, \xi, h\rangle \quad \xrightarrow{c(A)} \quad \langle K[A], c', \gamma, \xi, h\rangle \\
& \text{when } c : \sigma,\ A : \sigma \text{ and } \gamma(c) = K,\ \xi(c) = c' \\[1em]
(OQ) & \langle \gamma, \xi, h\rangle \quad \xrightarrow{f(A,c)} \quad \langle VA, c, \gamma, \xi, h\rangle \\
& \text{when } f : \sigma \to \sigma',\ A : \sigma,\ c : \sigma' \text{ and } \gamma(f) = V
\end{array}
$$

We consider the following program M:

$$\texttt{let x = ref 0 in } \lambda \texttt{f.x} := !\texttt{x} - 1; \texttt{f}(); \texttt{x} := !\texttt{x} + 1; !\texttt{x}$$

We consider the following program M:

$$\texttt{let x = ref 0 in } \lambda\texttt{f.x} := !\texttt{x} - 1; \texttt{f}(); \texttt{x} := !\texttt{x} + 1; !\texttt{x}$$

$$\langle M, c_0 \rangle \quad \xrightarrow{\tau} \quad \langle \overbrace{\lambda\texttt{f}.\ell := !\ell - 1; \texttt{f}(); \ell := !\ell + 1; !\ell}^{V}, c_0, [\ell \mapsto 0] \rangle$$

We consider the following program M:

$$\texttt{let x = ref 0 in } \lambda\texttt{f.x} :=!\texttt{x} - 1; \texttt{f}(); \texttt{x} :=!\texttt{x} + 1; !\texttt{x}$$

$$\begin{aligned} \langle M, c_0 \rangle &\xrightarrow{\tau} \langle \overbrace{\lambda \texttt{f}.\ell :=!\ell - 1; \texttt{f}(); \ell :=!\ell + 1; !\ell}^{V}, c_0, [\ell \mapsto 0] \rangle \\ &\xrightarrow{\bar{c}_0(g)} \langle \underbrace{[g \mapsto V]}_{\gamma}, [\ell \mapsto 0] \rangle \end{aligned}$$

We consider the following program M:

$$\texttt{let x = ref 0 in } \lambda \texttt{f.x} := !\texttt{x} - 1; \texttt{f}(); \texttt{x} := !\texttt{x} + 1; !\texttt{x}$$

$$
\begin{array}{lll}
\langle M, c_0 \rangle & \xrightarrow{\tau} & \langle \overbrace{\lambda \texttt{f}.\ell := !\ell - 1; \texttt{f}(); \ell := !\ell + 1; !\ell}^{V}, c_0, [\ell \mapsto 0] \rangle \\
& \xrightarrow{\bar{c}_0(g)} & \langle \underbrace{[g \mapsto V]}_{\gamma}, [\ell \mapsto 0] \rangle \\
& \xrightarrow{g(f, c_1)} & \langle V\ f, c_1, \gamma, [\ell \mapsto 0] \rangle
\end{array}
$$

We consider the following program M:

$$\texttt{let x = ref 0 in } \lambda \texttt{f.x} := !\texttt{x} - 1; \texttt{f}(); \texttt{x} := !\texttt{x} + 1; !\texttt{x}$$

$$\begin{array}{lll}
\langle M, c_0 \rangle & \xrightarrow{\tau} & \langle \overbrace{\lambda \texttt{f}.\ell := !\ell - 1; \texttt{f}(); \ell := !\ell + 1; !\ell}^{V}, c_0, [\ell \mapsto 0] \rangle \\
& \xrightarrow{\bar{c}_0(g)} & \langle \underbrace{[g \mapsto V]}_{\gamma}, [\ell \mapsto 0] \rangle \\
& \xrightarrow{g(f, c_1)} & \langle V \ f, c_1, \gamma, [\ell \mapsto 0] \rangle \\
& \xrightarrow{\tau} & \langle K[f()], c_1, \gamma, [\ell \mapsto -1] \rangle \quad \text{with } K = \bullet; \ell := !\ell + 1; !\ell
\end{array}$$

We consider the following program M:

$$\texttt{let x = ref 0 in } \lambda \texttt{f.x} := !\texttt{x} - 1; \texttt{f}(); \texttt{x} := !\texttt{x} + 1; !\texttt{x}$$

$$\begin{array}{lll}
\langle M, c_0 \rangle & \xrightarrow{\tau} & \langle \overbrace{\lambda \texttt{f}.\ell := !\ell - 1; \texttt{f}(); \ell := !\ell + 1; !\ell}^{V}, c_0, [\ell \mapsto 0] \rangle \\
 & \xrightarrow{\bar{c_0}(g)} & \langle \underbrace{[g \mapsto V]}_{\gamma}, [\ell \mapsto 0] \rangle \\
 & \xrightarrow{g(f, c_1)} & \langle V\ f, c_1, \gamma, [\ell \mapsto 0] \rangle \\
 & \xrightarrow{\tau} & \langle K[f()], c_1, \gamma, [\ell \mapsto -1] \rangle \quad \text{with } K = \bullet; \ell := !\ell + 1; !\ell \\
 & \xrightarrow{\bar{f}((), c_2)} & \langle \overbrace{\gamma \cdot [c_2 \mapsto K]}^{\gamma'}, \overbrace{[c_2 \mapsto c_1]}^{\xi}, [\ell \mapsto -1] \rangle
\end{array}$$

We consider the following program M:

$$\texttt{let x} \ = \ \texttt{ref 0 in } \lambda \texttt{f.x} := !\texttt{x} - 1; \texttt{f}(); \texttt{x} := !\texttt{x} + 1; !\texttt{x}$$

$$\begin{array}{lll}
\langle M, c_0 \rangle & \xrightarrow{\tau} & \langle \overbrace{\lambda \texttt{f}.\ell := !\ell - 1; \texttt{f}(); \ell := !\ell + 1; !\ell}^{V}, c_0, [\ell \mapsto 0] \rangle \\
& \xrightarrow{\bar{c_0}(g)} & \langle \underbrace{[g \mapsto V]}_{\gamma}, [\ell \mapsto 0] \rangle \\
& \xrightarrow{g(f, c_1)} & \langle V \ f, c_1, \gamma, [\ell \mapsto 0] \rangle \\
& \xrightarrow{\tau} & \langle K[f()], c_1, \gamma, [\ell \mapsto -1] \rangle \quad \text{with } K = \bullet; \ell := !\ell + 1; !\ell \\
& \xrightarrow{\bar{f}((), c_2)} & \langle \overbrace{\gamma \cdot [c_2 \mapsto K]}^{\gamma'}, \overbrace{[c_2 \mapsto c_1]}^{\xi}, [\ell \mapsto -1] \rangle \\
& & \text{here Opponent can either answer to } c_2 \\
& & \text{or interrogate } g \text{ again (nested call)} \\
& \xrightarrow{c_2(())} & \langle K[()], c_2, \gamma', \xi, [\ell \mapsto -1] \rangle \\
& \ldots &
\end{array}$$

c_2 can still be used afterwards, meaning that we can use the evaluation context K many times to increment ℓ.

Operational Game Semantics for HOS contexts

- Opponent cannot use call/cc anymore
  - but Player still can (*asymmetric situation*).
- Opponent behaviour is then restricted:
  - Opponent Answers must follow the **O-bracketing** discipline
  - enforced in the OA rule of the LTS by keeping track of the continuation structure for Opponent too.
- **Well-bracketing** when Player cannot use call/cc neither: the continuation structure used in the LTS is then a stack, getting back the model of [Laird 2007].

LTS for HOS

$$
\begin{array}{l|lcl}
(P\tau) & \langle M, c, \gamma, \xi, h \rangle & \xrightarrow{\tau} & \langle N, c', \gamma, \xi, h' \rangle \\
& \text{when } (M, c, h) \to (N, c', h') & & \\
(PA) & \langle V, c, \gamma, \xi, h \rangle & \xrightarrow{\bar{c}(A)} & \langle \gamma \cdot \gamma', \xi, h, c' \rangle \\
& \text{when } c : \sigma,\ (A, \gamma') \in \mathsf{AVal}_\sigma(V),\ \xi(c) = c' & & \\
(PQ) & \langle K[fV], c, \gamma, \xi, h \rangle & \xrightarrow{\bar{f}(A, c')} & \langle \gamma \cdot \gamma' \cdot [c' \mapsto K], \xi \cdot [c' \mapsto c], h, c' \rangle \\
& \text{when } f : \sigma \to \sigma',\ (A, \gamma') \in \mathsf{AVal}_\sigma(V),\ c' : \sigma' & & \\
(OA) & \langle \gamma, \xi, h, c'' \rangle & \xrightarrow{c(A)} & \langle K[A], c', \gamma, \xi, h \rangle \\
& \text{when } c = c'',\ c : \sigma,\ A : \sigma,\ \gamma(c) = K,\ \xi(c) = c' & & \\
(OQ) & \langle \gamma, \xi, h, c'' \rangle & \xrightarrow{f(A, c)} & \langle VA, c, \gamma, \xi \cdot [c \mapsto c''], h \rangle \\
& \text{when } f : \sigma \to \sigma',\ A : \sigma,\ c : \sigma',\ \gamma(f) = V & &
\end{array}
$$

$$M \triangleq \texttt{let x = ref 0 in } \lambda \texttt{f.x} := !\texttt{x} - 1; \texttt{f}(); \texttt{x} := !\texttt{x} + 1; !\texttt{x}$$

$$
\begin{array}{ll}
\langle M, c_0, \overbrace{[c_0 \mapsto \bot]}^{\xi_0} \rangle \xrightarrow{\tau} & \langle \overbrace{\lambda \texttt{f}.\ell := !\ell - 1; \texttt{f}(); \ell := !\ell + 1; !\ell}^{V}, c_0, \xi_0, [\ell \mapsto 0] \rangle \\
\xrightarrow{\bar{c_0}(g)} & \langle \underbrace{[g \mapsto V]}_{\gamma}, \xi_0, [\ell \mapsto 0], \bot \rangle \\
\xrightarrow{g(f, c_1)} & \langle V\ f, c_1, \overbrace{\xi_0 \cdot [c_1 \mapsto \bot]}^{\xi_1}, \gamma, [\ell \mapsto 0] \rangle \\
\xrightarrow{\tau} & \langle K[f()], c_1, \gamma, \xi_1, [\ell \mapsto -1] \rangle \text{ with } K = \bullet; \ell := !\ell + 1; !\ell \\
\xrightarrow{\bar{f}((), c_2)} & \langle \overbrace{\gamma \cdot [c_2 \mapsto K]}^{\gamma'}, \overbrace{\xi_1 \cdot [c_2 \mapsto c_1]}^{\xi_2}, [\ell \mapsto -1], c_2 \rangle \\
\xrightarrow{g(f', c_3)} & \langle V\ f', c_3, \gamma', \overbrace{\xi_2 \cdot [c_3 \mapsto c_2]}^{\xi_3}, [\ell \mapsto -1] \rangle \\
\xrightarrow{\tau} & \langle K[f'()], c_3, \gamma', \xi_3, [\ell \mapsto -2] \rangle \\
\xrightarrow{\bar{f'}((), c_4)} & \langle \overbrace{\gamma \cdot [c_4 \mapsto K]}^{\gamma'}, \overbrace{\xi_3 \cdot [c_4 \mapsto c_3]}^{\xi_4}, [\ell \mapsto -2], c_4 \rangle
\end{array}
$$

Then Opponent answers should be first on c_4, then on c_2.

Operational Game Semantics for GOSC contexts

- Opponent cannot store functional values nor continuations produced by call/cc
  - but Player still can (*asymmetric situation*).
- Both Opponent Questions and Answers are restricted: **O-visibility**
  - to control the functional names used in Opponent Questions and the continuation names used in Opponent Answers to be in the *scope*, called the O-view;
  - enforced in the Opponent rules of the LTS by keeping track of the O-views at each interaction points.

Modularity

Operational Game Semantics for GOS contexts

- Opponent cannot store functional values and cannot use call/cc.
- Combine both O-bracketing and O-visibility
- On Opponent Answer, O-bracketing implies O-visibility.

Modularity

Operational Game Semantics for GOS contexts

- Opponent cannot store functional values and cannot use call/cc.
- Combine both O-bracketing and O-visibility
- On Opponent Answer, O-bracketing implies O-visibility.

More generally, a common LTS for the four fragments:

- enforcing O-bracketing and O-visibility is based on the *trace already played*;
- uniform treatment by incorporating in configurations a notion of *history of available names used*;
- condition on Opponent moves depending if we want to capture HOSC, HOS, GOSC, or GOS contexts.

Theorem (Full Abstraction)

For each fragment GOS,GOSC,HOS,HOSC, the set of traces, generated by the corresponding LTS, for two programs M_1, M_2, are equal iff M_1, M_2 are contextually equivalent.

- For HOS and GOS: error observation rather than termination in the definition of contextual equivalence
  - to avoid the restriction to *complete* traces.

- Proofs of these full-abstraction results use ciu-equivalence and definability theorems.

Theorem (Full Abstraction)

For each fragment GOS,GOSC,HOS,HOSC, the set of traces, generated by the corresponding LTS, for two programs M_1, M_2, are equal iff M_1, M_2 are contextually equivalent.

- For HOS and GOS: error observation rather than termination in the definition of contextual equivalence
  - to avoid the restriction to *complete* traces.

- Proofs of these full-abstraction results use ciu-equivalence and definability theorems.

- Asymmetric case: M_1, M_2 are in HOSC, while contexts are taken either in HOS, GOS or GOSC;
  - Player is more powerful than Opponent;
  - soundness wrt ciu-equivalence only...
  - equivalence is not a congruence anymore !

Can we prove contextual equivalence of two programs using these fully-abstract models ?

Prove trace equality of the LTS corresponding to the two programs using bisimulations.

- Eager normal-form bisimulations for $\sim$HOSC [Støvring & Lassen 2007]
- extended to HOS in [Biernacki, Lenglet & Polesiuk, 2019]
- main difficulty: the dynamic environment γ of LTS configurations keeps growing

1. Operational Semantics

2. Operational Game Semantics

3. Kripke Normal-Form Bisimulations

We would like to relate each corresponding components of the environments independently

Problem: this is **unsound** in presence of references

`let c = ref0 in`
`let inc () = c :=` $!c + 1$ `in`
`let get () = !c in`
$\langle$`inc, get`$\rangle$
	`let c = ref0 in`
	`let dec () = c :=` $!c - 1$ `in`
	`let get () = !c in`
	$\langle$`dec, get`$\rangle$

$\rightsquigarrow$ Related to the unsoundness of applicative bisimulations in a similar setting [Koutavas,Levy, Sumii 2015] ?

A Solution:

Use worlds w as memory invariants to specify the heap resources shared by all the components of the two programs.

$\rightsquigarrow$ as introduced with Kripke Logical Relations [Pitts & Stark 1998; Dreyer, Neis, Birkedal 2010]

Adaptation to Bisimulations over Operational Game Semantics LTS:

Kripke Normal-Form Bisimulations

Kripke reasoning

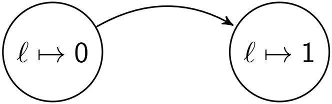
Definition

A world transition system (WTS) $\mathcal{A}$ is a triple $(\mathrm{Worlds}, \sqsubseteq_{\mathsf{OQ}}, \sqsubseteq_{\mathsf{OA}}, \mathcal{I})$, where:

- Worlds is a set of states,
- $\sqsubseteq_{\mathsf{OQ}}, \sqsubseteq_{\mathsf{OA}}$ are binary reflexive relations on Worlds,
- $\mathcal{I} : \mathrm{Worlds} \to \mathcal{P}(\mathrm{Heap} \times \mathrm{Heap})$ is the *invariant assignment*.

For example:

- Relational invariants: $\mathcal{I}(w) = \{(h_1, h_2) \mid h_1(\ell_1) = -h_2(\ell_2)\}$
- Transition systems of invariants:



Decomposing configurations

- Partial configurations D, D': remove the shared resources
  - heap h;
  - available name history components used to restrict Opponent behaviour.

- Product of partial configurations $D \otimes D'$:
  - concatenate the dynamic environments γ, γ' of D, D';
  - at most one Player (i.e. active) configuration among D, D';
  - Opponent names can be shared between D, D'.

- Prime configurations:
  - partial configurations that are irreducible w.r.t. $\otimes$;
  - either Player (active) configuration with empty dynamic environment γ;
  - or Opponent (passive) configurations with singleton γ.

Decomposing configurations

- Partial configurations D, D': remove the shared resources
  - heap h;
  - available name history components used to restrict Opponent behaviour.

- Product of partial configurations $D \otimes D'$:
  - concatenate the dynamic environments γ, γ' of D, D';
  - at most one Player (i.e. active) configuration among D, D';
  - Opponent names can be shared between D, D'.

- Prime configurations:
  - partial configurations that are irreducible w.r.t. $\otimes$;
  - either Player (active) configuration with empty dynamic environment γ;
  - or Opponent (passive) configurations with singleton γ.

Normal-Form bisimulations/Bohm trees corresponds to LTS over prime configurations !

Kripke Normal-Form Bisimulations for HOSC

Kripke Normal-Form Bisimulations are relations over triples $R = (R_{\mathcal{V}}, R_{\mathcal{K}}, R_{\mathcal{E}})$ which are post-fixpoint of $(\mathcal{V}_{\mathcal{A}}, \mathcal{K}_{\mathcal{A}}, \mathcal{E}_{\mathcal{A}})$, i.e. $R \subseteq (\mathcal{V}_{\mathcal{A}}(R), \mathcal{K}_{\mathcal{A}}(R), \mathcal{E}_{\mathcal{A}}(R))$, where:

$$
\begin{array}{rcl}
\mathcal{V}_{\mathcal{A}}(R_{\mathcal{V}}, R_{\mathcal{K}}, R_{\mathcal{E}}) & \triangleq & \{\ (\beta, V, V, w) \mid V : \beta \ \wedge \ \ \beta \in \mathrm{Int}, \mathrm{Bool}, \mathrm{Unit}\} \cup \\
& & \quad \{\ (\sigma \to \sigma', V_1, V_2, w) \mid \forall w' \sqsupseteq^{*}_{\mathsf{OQ}} w.\ \forall A : \sigma.\ \forall c : \tau. \\
& & \qquad (\sigma', V_1 A, c, V_2 A, c, w') \in R_{\mathcal{E}}\ \} \\
\mathcal{K}_{\mathcal{A}}(R_{\mathcal{V}}, R_{\mathcal{K}}, R_{\mathcal{E}}) & \triangleq & \{\ (\sigma, \sigma', K_1, c_1, K_2, c_2, w) \mid \forall w' \sqsupseteq^{*}_{\mathsf{OA}} w.\ \forall A : \sigma. \\
& & \qquad (\sigma', K_1[A], c_1, K_2[A], c_2, w') \in R_{\mathcal{E}}\ \} \\
\mathcal{E}_{\mathcal{A}}(R_{\mathcal{V}}, R_{\mathcal{K}}, R_{\mathcal{E}}) & \triangleq & \{\ (\sigma, M_1, c_1, M_2, c_2, w) \mid \forall (h_1, h_2) \in \mathcal{I}(w). \\
& & \qquad\qquad P_{Div} \vee P_{PA} \vee P_{PQ}\ \}
\end{array}
$$

Kripke Normal-Form Bisimulations for HOSC

$$P_{Div} \triangleq (M_1, c_1, h_1) \Uparrow \wedge (M_2, c_2, h_2) \Uparrow$$

$$\begin{aligned} P_{PA} \triangleq\ & \exists V_1, V_2, c.\ \exists w' \sqsupseteq_c w.\ \exists (h_1', h_2') \in \mathcal{I}(w').\ (\sigma, V_1, V_2, w') \in R_{\mathcal{V}} \wedge \\ & (M_1, c_1, h_1) \to^* (V_1, c, h_1') \wedge (M_2, c_2, h_2) \to^* (V_2, c, h_2') \end{aligned}$$

$$\begin{aligned} P_{PQ} \triangleq\ & \exists K_1, V_1, K_2, V_2.\ \exists c_1', c_2' : \tau.\ \exists \sigma_1, \sigma_2.\ \exists f : \sigma_1 \to \sigma_2.\ \exists w' \sqsupseteq_f (w). \\ & \exists (h_1', h_2') \in \mathcal{I}(w').\ (\sigma_1, V_1, V_2, w', \mathcal{H}) \in R_{\mathcal{V}} \\ & \quad \wedge (\sigma_2, \sigma, K_1, c_1', K_2, c_2', w', \mathcal{H}) \in R_{\mathcal{K}} \\ & \wedge (M_1, c_1, h_1) \to^* (K_1[fV_1], c_1', h_1') \wedge (M_2, c_2, h_2) \to^* (K_2[fV_2], c_2', h_2') \end{aligned}$$

Modular Kripke Normal-Form Bisimulations

Extending the definition to restricted Opponent (GOS,GOSC,HOS):

- by index the definition of KNFB with a world-history $\mathcal{H}$ that associates worlds to continuation and functional names;
- related to the available-name history of the uniform OGS LTS.

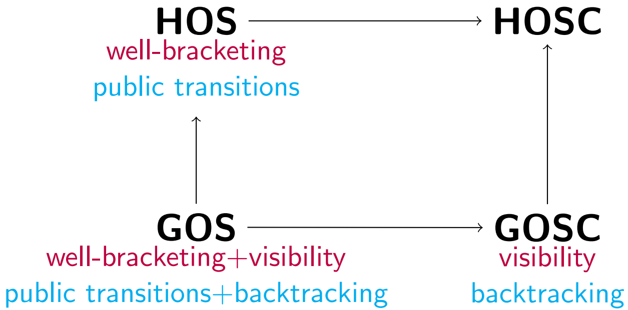
Theorem (Soundness & Completeness)

For each fragment $\mathsf{x} \in \{GOS, GOSC, HOS, HOSC\}$*, considering for two programs* M_1, M_2 *of these fragments, there exists a world transition system* $\mathcal{A}$ *and an* $\mathcal{A}, \mathsf{x}$*-Kripke normal-form bisimulation that contains* M_1, M_2 *iff* M_1, M_2 *are contextually equivalent in* x*.*

- Soundness: prime decomposition of the LTS.
- Completeness: transformation of the OGS LTS into a WTS $\mathcal{A}$.

Relating Semantic Squares

Relate semantic characterizations of effects
coming from game semantics and Kripke logical relations



- [Abramsky & McCusker 1997], [Laird 1997]
- [Dreyer, Neis & Birkedal 2010]

In the future:

- Construction from normal-form bisimulations to Kripke normal-form bisimulations as an up-to/abstraction technique ?
- Assymetric reasoning on effects between the program and contexts and fully-abstract interopability.
- Extension to parametric polymorphism, following [Lassen, Levy 2008], [Jaber, Tzevelekos 2016, 2018].
- Automation of contextual equivalence for these fragments, following [Jaber 2020].