# Towards a Synchronous Game Semantics*

Mohamed N. Menaa
&
Dan Ghica

University of Birmingham

GaLoP
28 March 2009

* (Work in progress)

# Synchrony

## The Perfectly Synchronous Concurrency Model

Based on the *synchronous hypothesis*: concurrent processes can compute and communicate in *zero time* (on a level of abstraction).

## Synchronous Languages

Computation proceeds in a sequence of atomic macro-steps (rounds) within which micro-steps are considered *simultaneous*, cyclically:

1. read the inputs
2. compute
3. produce the outputs

# 1 – Game Semantics is Asynchronous

# Concurrent Game Semantics

## Game semantics of Concurrent Algol [GM07]

- Language constants interpreted by *saturated strategies*
  - record all sequential observations of parallel interactions.

## Definition

$\sigma : A$ is saturated iff

1. If $s_0.m_1.m_2.s_1 \in \sigma$ and $\lambda_A(m_1) = \lambda_A(m_2)$ then $s_0.m_2.m_1.s_1 \in \sigma$
2. If $s_0.p.o.s_1 \in \sigma$ and $s_0.o.p.s_1 \in P_A$ then $s_0.o.p.s_1 \in \sigma$

# Asynchrony in Game Semantics

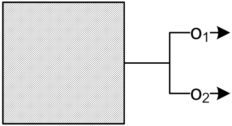
Saturated strategies capture the intuition that in a concurrent (asynchronous) setting, some of the ordering of events in a play is arbitrary:

- Arbitrary delays on communication channels.

$$m \parallel m' \rightsquigarrow m.m', m'.m$$

# True Concurrency

In some execution models (e.g. clocked digital hardware), concurrent events are truly simultaneous.

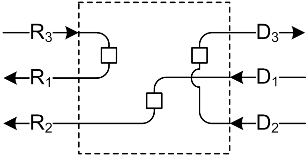



$$o_1 \parallel o_2 \rightsquigarrow \langle o_1, o_2 \rangle$$

# 2 – Synchronous Interpretations of Asynchronous Primitives

# I/O Simultaneity

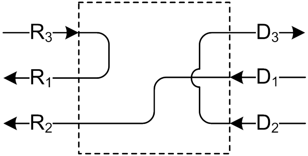
$$[\![ seq : com_1 \times com_2 \Rightarrow com_3 ]\!]$$




$$R_3 . R_1 . D_1 . R_2 . D_2 . D_3$$

# I/O Simultaneity

$$[\![ seq : com_1 \times com_2 \Rightarrow com_3 ]\!]$$




~~$R_3.R_1.D_1.R_2.D_2.D_3$~~

In a synchronous setting:

$\langle R_3, R_1 \rangle . \langle D_1, R_2 \rangle . \langle D_2, D_3 \rangle$

# Round Abstraction

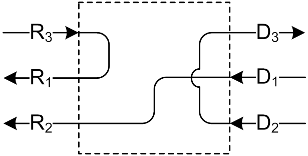
- Given an output variable $x$ on an asynchronous module $P$, `next` $x$ `for` $P$ is the module obtained by collapsing all computational steps occuring between two changes in $x$ into a single computational step [AH99].
- Use a variant where every output in a *round marker*, to systematically derive synchronous strategies for primitive that have an asynchronous definitions.

## Round generation

- if $s_1.o.p.s_2 \in \sigma$ then $s_1.\langle o, p\rangle.s_2 \in RA(\sigma)$
- if $s_1.p_1.p_2.s_2 \in \sigma$ then $s_1.\langle p_1, p_2\rangle.s_2 \in RA(\sigma)$

# I/O Simultaneity

$$[\![ seq : com_1 \times com_2 \Rightarrow com_3 ]\!]$$



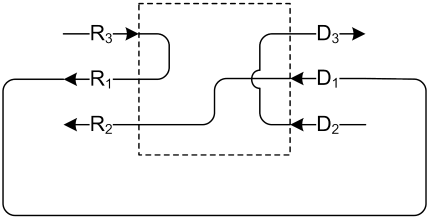

~~$R_3.R_1.D_1.R_2.D_2.D_3$~~

In a synchronous setting:

$$\langle R_3, R_1\rangle.\underbrace{\langle D_1, R_2\rangle}_{\text{round}}.\langle D_2, D_3\rangle$$

# O/I Simultaneity

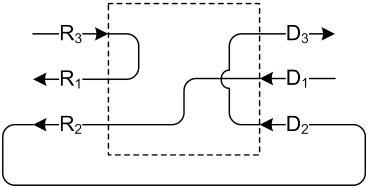
$$[\![ seq : com_1 \times com_2 \Rightarrow com_3 ]\!]$$




$$\langle R_3, R_1 \rangle . \langle D_1, R_2 \rangle . \langle D_2, D_3 \rangle$$
$$\langle R_3, R_1, D_1, R_2 \rangle . \langle D_2, D_3 \rangle$$

# O/I Simultaneity

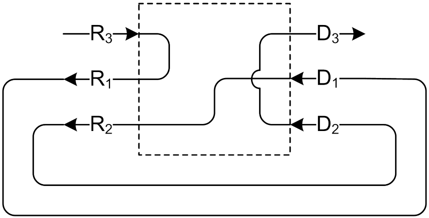
$$[\![ seq : com_1 \times com_2 \Rightarrow com_3 ]\!]$$




$$\langle R_3, R_1 \rangle . \langle D_1, R_2 \rangle . \langle D_2, D_3 \rangle$$
$$\langle R_3, R_1, D_1, R_2 \rangle . \langle D_2, D_3 \rangle$$
$$\langle R_3, R_1 \rangle . \langle D_1, R_2, D_2, D_3 \rangle$$

# O/I Simultaneity

$$[\![ seq : com_1 \times com_2 \Rightarrow com_3 ]\!]$$




$$\langle R_3, R_1\rangle.\langle D_1, R_2\rangle.\langle D_2, D_3\rangle$$
$$\langle R_3, R_1, D_1, R_2\rangle.\langle D_2, D_3\rangle$$
$$\langle R_3, R_1\rangle.\langle D_1, R_2, D_2, D_3\rangle$$
$$\langle R_3, R_1, D_1, R_2, D_2, D_3\rangle$$

# Round Abstraction

- Given an output variable $x$ on an asynchronous module $P$, `next` $x$ `for` $P$ is the module obtained by collapsing all computational steps occuring between two changes in $x$ into a single computational step [AH99].
- Use a similar concept to systematically derive synchronous strategies for primitive that have an asynchronous definitions

## Round generation

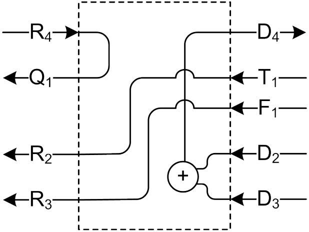
- if $s_1.o.p.s_2 \in \sigma$ then $s_1.\langle o, p\rangle.s_2 \in RA(\sigma)$
- if $s_1.p_1.p_2.s_2 \in \sigma$ then $s_1.\langle p_1, p_2\rangle.s_2 \in RA(\sigma)$

## Instant feedback

- if $s_1.p.o.s_2 \in RA(\sigma)$ then $s_1.\langle p, o\rangle.s_2 \in RA(\sigma)$
- if $s_1.o_1.o_2.s_2 \in RA(\sigma)$ then $s_1.\langle o_1, o_2\rangle.s_2 \in RA(\sigma)$

# Strategy Derivation Through Round Abstraction

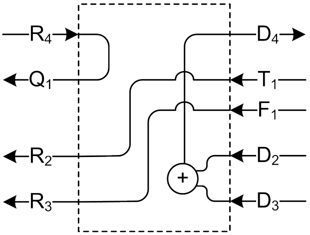
$$[\![ if : (exp_1 \times com_2 \times com_3) \rightarrow com_4 ]\!]$$




$$R4.Q1.T1.R2.D2.D4 \quad \xrightarrow{RA} \quad \begin{array}{c} \langle R4, Q1\rangle.\langle T1, R2\rangle.\langle D2, D4\rangle \\ \langle R4, Q1, T1, R2\rangle.\langle D2, D4\rangle \\ \langle R4, Q1\rangle.\langle T1, R2, D2, D4\rangle \\ \langle R4, Q1, T1, R2, D2, D4\rangle \end{array}$$

# Strategy Derivation Through Round Abstraction

$$[\![ if : (exp_1 \times com_2 \times com_3) \rightarrow com_4 ]\!]$$




$$R4.Q1.F1.R3.D3.D4 \quad \xrightarrow{RA} \quad \begin{array}{c} \langle R4, Q1\rangle.\langle F1, R3\rangle.\langle D3, D4\rangle \\ \langle R4, Q1, F1, R3\rangle.\langle D3, D4\rangle \\ \langle R4, Q1\rangle.\langle F1, R3, D3, D4\rangle \\ \langle R4, Q1, F1, R3, D3, D4\rangle \end{array}$$

# 3 – Synchronous Interpretations of Synchronous Primitives

# Synchronous Primitives

Strategies for synchronous primitives can be formulated.

# Synchronous Primitives

Strategies for synchronous primitives can be formulated.

## Esterel [BMR83]

Programs typically consist of several processes composed in parallel and synchronising using signals.

- Processes: sequential threads of execution.
- Signals: broadcast events of Boolean nature.

# Synchronous Primitives

Strategies for synchronous primitives can be formulated.

## Esterel [BMR83]

Programs typically consist of several processes composed in parallel and synchronising using signals.

- Processes: sequential threads of execution.
- Signals: broadcast events of Boolean nature.

## Some candidates (from Esterel)

- `pause`
- $p \parallel q$
- `emit` $S$
- `present` $S$ `then` $p$ `else` $q$ `end`
- `await` $S$
- `suspend` $p$ `when` $S$

# Synchronous Primitives

- ReactiveML [MP05] extends ML with such synchronous primitives by adding entities that are orthogonal to the type system.
  - Processes.
  - Signals.

# Synchronous Primitives

- ReactiveML [MP05] extends ML with such synchronous primitives by adding entities that are orthogonal to the type system.
  - ~~Processes~~ $\longrightarrow$ strategies.
  - ~~Signals~~ $\longrightarrow$ moves.

# Synchronous Primitives

- ReactiveML [MP05] extends ML with such synchronous primitives by adding entities that are orthogonal to the type system.
  - ~~Processes~~ → strategies.
  - ~~Signals~~ → moves.
- Use start and end of computation as *signals*.

# The Semantics of `await`

```
trap T in
  loop
    pause;
    present S then exit T else nothing end
  end
```

# The Semantics of `await`

```
trap T in
  loop
    pause;
    present S then exit T else nothing end
  end
```

- Variant: await the start of a command.
- A semantic version of a pointcut in Aspect-oriented Programming.

# The Semantics of `await`

```
trap T in
  loop
    pause;
    present S then exit T else nothing end
  end
```

- Variant: await the start of a command.
- A semantic version of a pointcut in Aspect-oriented Programming.

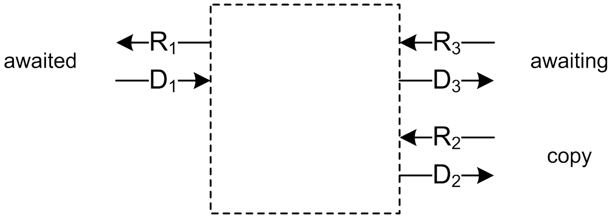
await: com $\Rightarrow$ com

$r^o$

$r^p$

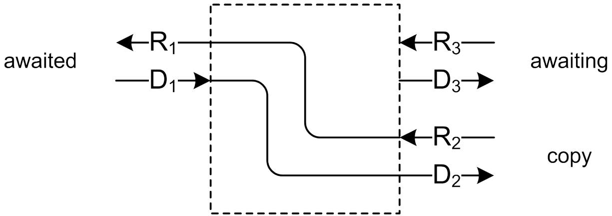
d

# The Semantics of `await`




await: $com_1 \Rightarrow com_2 \times com_3$

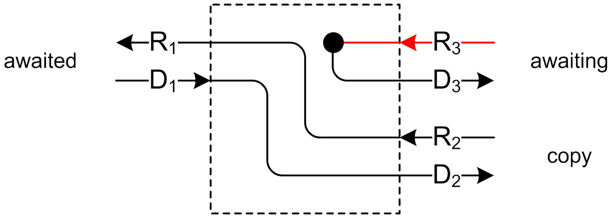
# The Semantics of `await`




| await: | $com_1$ | $\Rightarrow$ | $com_2$ | $\times$ | $com_3$ |
|---|---|---|---|---|---|
| | r | | r | | |
| | d | | d | | |

$$\langle R2, R1\rangle.\langle D1, R2\rangle$$
$$\langle R2, R1, D1, R2\rangle$$

# The Semantics of `await`



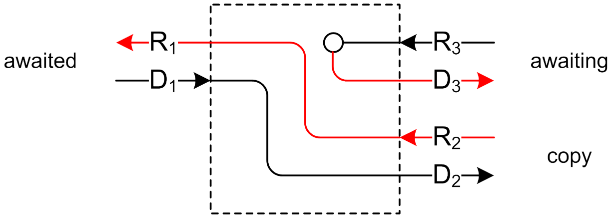

await: $com_1 \Rightarrow com_2 \times com_3$

| | | |
|---|---|---|
| | | r |
| r | r | d |
| d | d | |

$$\langle R2, R1\rangle.\langle D1, R2\rangle$$
$$\langle R2, R1, D1, R2\rangle$$

$$R3$$

# The Semantics of `await`



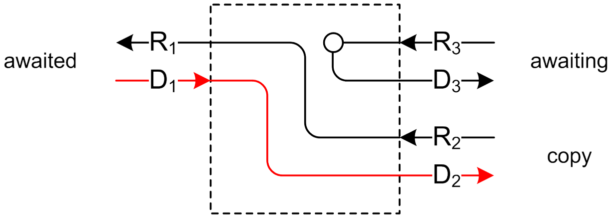

await: $com_1 \Rightarrow com_2 \times com_3$

| | | |
|---|---|---|
| | | r |
| r | r | d |
| d | d | |

$$\langle R2, R1\rangle.\langle D1, R2\rangle$$
$$\langle R2, R1, D1, R2\rangle$$

$$R3.\langle R2, R1, D3\rangle$$

# The Semantics of `await`



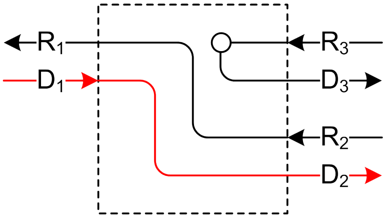

await: $com_1 \Rightarrow com_2 \times com_3$

| | | |
|---|---|---|
| | | r |
| r | r | d |
| d | d | |

$\langle R2, R1 \rangle . \langle D1, R2 \rangle$
$\langle R2, R1, D1, R2 \rangle$

$R3.\langle R2, R1, D3 \rangle . \langle D1, D2 \rangle$

# The Semantics of `await`




await: $com_1 \Rightarrow com_2 \times com_3$

| | $com_1$ | $com_2$ | $com_3$ |
|---|---|---|---|
| | | | r |
| | r | r | d |
| | d | d | |

$$\langle R2, R1\rangle.\langle D1, R2\rangle$$
$$\langle R2, R1, D1, R2\rangle$$

$$R3.\langle R2, R1, D3\rangle.\langle D1, D2\rangle$$

$$R3.\langle R2, R1, D3, D1, D2\rangle$$
$$\langle R3, R2, R1, D3\rangle.\langle D1, D2\rangle$$
$$\langle R3, R2, R1, D3, D1, D2\rangle$$

# 4 – Categorical Structure

# Synchronous Traces

Plays represented using *synchronous traces*.

## Definition

A trace $t \in U$, where $U$ is an arbitrary set of traces over a set of labels $L$, is a triple $\langle E, \preceq_E, \lambda : E \rightarrow L \rangle$ where

- $E$ is a set of events,
- $\preceq_E$ is a total preorder between events signifying *temporal precedence*. The equivalence relation $\approx_E$, which means the *simultaneous occurrence* of two events, is defined as:

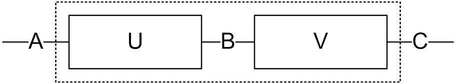
$$\forall a, b \in E \bullet a \preceq_E b \wedge b \preceq_E a \Leftrightarrow a \approx_E b$$

- $\lambda$ is a function mapping events to labels in a set $L$.

# Category

- Objects: sets of labels.
- Morphisms: sets of synchronous traces between sets of labels.

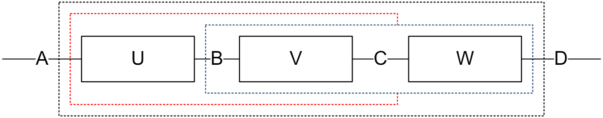
# Composition



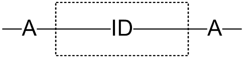

## Definition

$U : A \rightarrow B$ and $V : B \rightarrow C$ are two arbitrary sets of synchronous traces. Their composition is a set of traces $U; V : A \rightarrow C$ defined as:

$$\begin{aligned} U; V = \{ t' \in \Theta_{A+C} \mid \exists t \in \Theta_{A+B+C} \bullet \\ out^{A+B+C}_{A+B}(t) \in U \wedge \\ out^{A+B+C}_{B+C}(t) \in V \wedge \\ t' = out^{A+B+C}_{A+C}(t) \} \end{aligned}$$

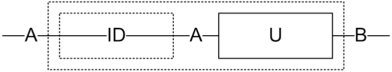


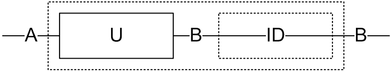
# Identity



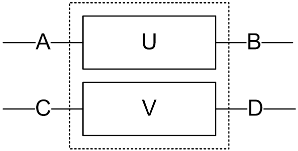

## Definition

$$
\begin{aligned}
ID_A = \{ \langle E, \preceq_E, \lambda : E \to A + A \rangle \mid \exists k \in \mathbb{N} \bullet E \overset{e}{\cong} \{1, 2, \ldots, 2k\}, \\
\forall i < 2k \bullet e(i) \preceq_E e(i+1) \wedge \\
(i \text{ is odd} \Rightarrow e(i) \approx_E e(i+1)) \wedge \\
(out_{A_1}^{A_1+A_2} \circ \lambda \circ e)(i) = (out_{A_2}^{A_1+A_2} \circ \lambda \circ e)(i+1) \}
\end{aligned}
$$



# Tensor



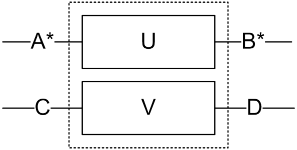

### Definition

A tensor is a bifunctor $\otimes : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ defined as

- On objects: $A \otimes B = A + B$.
- On morphisms: $U : A \to B$, $V : C \to D$

$$U \otimes V = \{t \in \Theta_{A+B+C+D} \mid out_{A+B}(t) \in U \wedge out_{C+D}(t) \in V\}$$

# Arrow



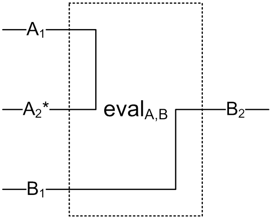

## Definition

The arrow is a functor $\Rightarrow: \mathcal{S}^{op} \times \mathcal{S} \to \mathcal{S}$ with the same definitions as $\otimes$.
In a polarised setting, its definitions are:

- On objects: $A \Rightarrow B = B + A^*$
- On morphisms: $U \Rightarrow V = V \otimes U^*$

where * reverses the I/O polarities of labels.

# Evaluation




## Definition

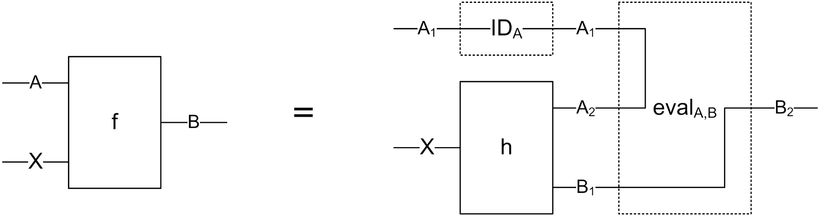
Eval is a morphism $eval_{A,B} : A \otimes (A \Rightarrow B) \rightarrow B$ that satisfies the following universal property: for every morphism $f : A \otimes X \rightarrow B$ in $\mathcal{S}$ there exists a unique morphism $h : X \rightarrow A \Rightarrow B$ such that $f = eval_{A,B} \circ (ID_A \otimes h)$. It is defined as:

$$eval_{A,B} = \{t \in \Theta_{A_1+A_2+B_1+B_2} \mid out_{A_1+A_2}(t) \in ID_{A_1+A_2} \wedge out_{B_1+B_2}(t) \in ID_{B_1+B_2}\}$$

# Evaluation - Universal Property

$\forall f : A \otimes X \rightarrow B$, $!\exists h : X \rightarrow A \Rightarrow B$ such that:

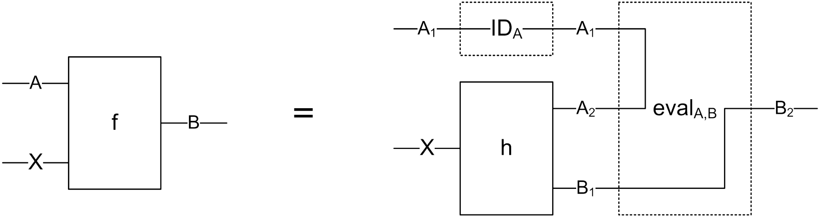
$$f = eval_{A,B} \circ (id_A \otimes h)$$



# Evaluation - Universal Property

$\forall f : A \otimes X \rightarrow B, !\exists h : X \rightarrow A \Rightarrow B$ such that:

$$f = eval_{A,B} \circ (id_A \otimes h)$$




(Compact) Closed monoidal category

# Outlook

- Closed monoidal category provides the right structural properties.
- Extend it with Cartesian product.
- Definability as a test for the choice of primitives.

# Outlook

- Closed monoidal category provides the right structural properties.
- Extend it with Cartesian product.
- Definability as a test for the choice of primitives.

THANKS!

# References


- Alur, R & Henzinger, T. A. (1999), "Reactive Modules", *Formal Methods in System Design*, 15, pp. 7–48.
- Berry, G., Moisan, S. & Rigault, J-P. (1983), "Esterel: Towards a Synchronous and Semantically Sound High-Level Language for Real-Time Applications", *Proc. IEEE Real-Time Systems Symposium*, pp. 30–40.
- Ghica, D. R. & Murawski, A. S. (2008), "Angelic Semantics of Fine-Grained Concurrency", *Annals of Pure and Applied Logic*, 151(2-3), pp. 89–114.
- Mandel, L. & Pouzet, M. (2005), "ReactiveML, a Reactive Extension to ML", *Proc. Principles and Practice of Declarative Programming*, pp. 82–93.