Token-passing semantics with and without rewriting

Koko Muroya  
(RIMS, Kyoto University & University of Birmingham)

Steven W. T. Cheung  
Dan R. Ghica  
(University of Birmingham)
Token-passing semantics 

*without* rewriting
Token-passing semantics without rewriting

execution models, provided by Girard’s GoI, of functional programming

- call-by-name \( \lambda \)-calculus [Danos&Regnier ‘99] [Mackie ‘95]
- call-by-value \( \lambda \)-calculus [Fernandez&Mackie ‘02]
- and more: PCF, effects, concurrency...
Token-passing semantics *without* rewriting

execution models, provided by Girard’s GoI, of functional programming

- call-by-name \( \lambda \)-calculus [Danos&Regnier ‘99] [Mackie ‘95]
- call-by-value \( \lambda \)-calculus [Fernandez&Mackie ‘02]
- and more: PCF, effects, concurrency...

<table>
<thead>
<tr>
<th>program</th>
<th>diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation</td>
<td>token passing</td>
</tr>
</tbody>
</table>
Token-passing semantics *without* rewriting

<table>
<thead>
<tr>
<th>program</th>
<th>diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation</td>
<td>token passing</td>
</tr>
<tr>
<td>result</td>
<td>token data/position</td>
</tr>
</tbody>
</table>

live demo:

Jamping Abstract Machine for call-by-name λ-calculus [DR99]

https://koko-m.github.io/GoI-Visualiser/
Token-passing semantics *without* rewriting

<table>
<thead>
<tr>
<th>program</th>
<th>( \lambda \text{-graph + !-boxes} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation</td>
<td><em>token passing + jumping</em></td>
</tr>
<tr>
<td>result</td>
<td><em>token data/position</em></td>
</tr>
</tbody>
</table>

live demo:

Jamping Abstract Machine for call-by-name \( \lambda \)-calculus [DR99]

[https://koko-m.github.io/GoI-Visualiser/](https://koko-m.github.io/GoI-Visualiser/)
Token-passing semantics *without* rewriting

<table>
<thead>
<tr>
<th>program</th>
<th>diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation</td>
<td><em>token passing</em></td>
</tr>
<tr>
<td>result</td>
<td><em>token data/position</em></td>
</tr>
</tbody>
</table>

**execution models of functional programming**

- compiler [Mackie ‘95]
- higher-order synthesis [Ghica ‘07]
Token-passing semantics
with rewriting
Token-passing semantics with rewriting

execution models of functional programming

- call-by-name & call-by-value λ-calculus [Sinot ‘05]
- call-by-need & fully lazy λ-calculus [Sinot ‘06]
- call-by-need & call-by-value λ-calculus [−&Ghica ‘17]

conventional small-step semantics, diagramatically (+ more)
Token-passing semantics with rewriting

execution models of functional programming

- call-by-name & call-by-value λ-calculus [Sinot ‘05]
- call-by-need & fully lazy λ-calculus [Sinot ‘06]
- call-by-need & call-by-value λ-calculus

![Diagram](image)

- evaluation
  - redex search
- reduction
- token passing
- diagram rewriting

inspired by virtual reduction [Danos & Regnier ‘93]
Token-passing semantics *with* rewriting

<table>
<thead>
<tr>
<th>program</th>
<th>diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation</td>
<td></td>
</tr>
<tr>
<td>redex search</td>
<td>token passing</td>
</tr>
<tr>
<td>reduction</td>
<td>diagram rewriting</td>
</tr>
<tr>
<td>result</td>
<td>diagram</td>
</tr>
</tbody>
</table>

live demo:

Dynamic GoI Machine for call-by-value \(\lambda\)-calculus [KG17]

[https://koko-m.github.io/GoI-Visualiser/](https://koko-m.github.io/GoI-Visualiser/)
Token-passing semantics *with* rewriting

<table>
<thead>
<tr>
<th>program</th>
<th>( \lambda\text{-graph} + !\text{-boxes} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation</td>
<td><em>redex search</em></td>
</tr>
<tr>
<td>reduction</td>
<td><em>diagram rewriting</em></td>
</tr>
<tr>
<td>result</td>
<td>( \lambda\text{-graph} + !\text{-boxes} )</td>
</tr>
</tbody>
</table>

**live demo:**

Dynamic GoI Machine for call-by-value \( \lambda \)-calculus [KG17]

[https://koko-m.github.io/GoI-Visualiser/](https://koko-m.github.io/GoI-Visualiser/)
Token-passing semantics *with* rewriting

<table>
<thead>
<tr>
<th>program</th>
<th>diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation</td>
<td></td>
</tr>
<tr>
<td><em>redex search</em></td>
<td><em>token passing</em></td>
</tr>
<tr>
<td><em>reduction</em></td>
<td><em>diagram rewriting</em></td>
</tr>
<tr>
<td>result</td>
<td><em>diagram</em></td>
</tr>
</tbody>
</table>

execution models of functional programming

- with robustness [S05] [S06]
- with time/space efficiency [MG17]
Token-passing semantics with rewriting

<table>
<thead>
<tr>
<th>program</th>
<th>diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation</td>
<td>redex search</td>
</tr>
<tr>
<td></td>
<td>reduction</td>
</tr>
<tr>
<td>result</td>
<td>diagram</td>
</tr>
</tbody>
</table>

execution models of functional programs

- with robustness [S05] [S06]
- with time/space efficiency [MG17]

whenever possible
## Token-passing semantics with rewriting

<table>
<thead>
<tr>
<th>program</th>
<th>diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation</td>
<td>redex search</td>
</tr>
<tr>
<td>reduction</td>
<td>diagram rewriting</td>
</tr>
<tr>
<td>result</td>
<td>diagram</td>
</tr>
</tbody>
</table>

**Execution models of functional programs**

- with robustness [S05] [S06]
- with time/space efficiency [MG17]
Token-passing semantics with and without rewriting
token-passing semantics *without* rewriting

... space efficiency

[Diagram: Plus sign]

token-passing semantics *with* rewriting

... time efficiency

[Diagram: Downward arrow]

selective rewriting

non-trivial space/time balancing?
token-passing semantics *without* rewriting

... space efficiency

+  

token-passing semantics *with* rewriting

... time efficiency

selective rewriting

non-trivial space/time balancing?

another perspective
token-passing semantics *without* rewriting

... result given by the token

+  

token-passing semantics *with* rewriting

... result given by a diagram

selective rewriting

programming with dual result

... given by the token *and* a diagram?
token-passing semantics without rewriting

... result given by the token

+ 

selective rewriting

programming with “computation graphs”

... result being value with computation graph
Programming with “computation graphs”

<table>
<thead>
<tr>
<th>result</th>
<th>value with computation graph</th>
</tr>
</thead>
</table>

- construction
- manipulation
Programming with “computation graphs”

TensorFlow, Google’s machine-learning library

```
W = tf.Variable(...)  
b = tf.Variable(...)  
y = W * x_data + b

x_data = ...   
y_data = ...   
sess = tf.Session()   
sess.run(init)   
sess.run(train)

x_data = ...   
sess = tf.Session()   
sess.run(init)   
y_initial_values = sess.run(y)
```

**construction**
- machine-learning model with parameters

**manipulation**
- model training
  - (imperative parameter update)

**value extraction**
- output prediction
Programming with "computation graphs"

TensorFlow, Google’s machine-learning library

\[ W = \text{tf.Variable}(...) \]
\[ b = \text{tf.Variable}(...) \]
\[ y = W \ast x_{\text{data}} + b \]

\[ x_{\text{data}} = \ldots \]
\[ y_{\text{data}} = \ldots \]
\[ \text{sess} = \text{tf.Session()} \]
\[ \text{sess.run(init)} \]
\[ \text{sess.run(train)} \]

\[ x_{\text{data}} = \ldots \]
\[ \text{sess} = \text{tf.Session()} \]
\[ \text{sess.run(init)} \]
\[ y\_\text{initial\_values} = \text{sess.run(y)} \]
Programming with “computation graphs”

TensorFlow, Google’s machine-learning library

```python
W = tf.Variable(...)  
b = tf.Variable(...)  
y = W * x_data + b

x_data = ...  
y_data = ...  
sess = tf.Session()  
sess.run(init)  
sess.run(train)

x_data = ...  
sess = tf.Session()  
sess.run(init)  
y_initial_values = sess.run(y)
```
Programming with “computation graphs”

TensorFlow, Google’s machine-learning library

\[ W = \text{tf.Variable}(...) \]
\[ b = \text{tf.Variable}(...) \]
\[ y = W \times x\_data + b \]

\[ x\_data = ... \]
\[ y\_data = ... \]
\[ \text{sess} = \text{tf.Session}() \]
\[ \text{sess.run} \text{(init)} \]
\[ \text{sess.run} \text{(train)} \]

\[ x\_data = ... \]
\[ \text{sess} = \text{tf.Session}() \]
\[ \text{sess.run} \text{(init)} \]
\[ \text{y\_initial\_values} = \text{sess.run}(y) \]
Programming with “computation graphs”

TensorFlow, Google’s machine-learning library

\[
W = \text{tf.Variable}(...) \\
b = \text{tf.Variable}(...) \\
y = W \times x\_data + b
\]

\[
x\_data = ... \\
y\_data = ... \\
sess = \text{tf.Session()} \\
sess.run(init) \\
sess.run(train)
\]

\[
x\_data = ... \\
sess = \text{tf.Session()} \\
sess.run(init) \\
y\_initial\_values = sess.run(y)
\]
Programming with “computation graphs”

Self-Adjusting Computation [Acar ‘05] (Incremental, an OCaml library)

```ocaml
let x = Inc.Var.create 1 in
let y = Inc.map2 (Inc.Var.watch x) (Inc.Var.watch x) ~f: (+) in
let z = Inc.Var.create 2 in
let w = Inc.map2 (Inc.Var.watch y) (Inc.Var.watch z) ~f: (+) in
let w_obs = Inc.observe w in
Inc.Var.set x 3;
Inc.stabilize ();

print_int (Inc.Observer.value_exn w_obs)
```

construction
acyclic dependency graph with “modifiables/cells”

manipulation
change propagation
value extraction
Programming with “computation graphs”

Self-Adjusting Computation [Acar ‘05] (Incremental, an OCaml library)

```
let x = Inc.Var.create 1 in
let y = Inc.map2
  (Inc.Var.watch x) (Inc.Var.watch z) ~f:(+) in
let z = Inc.Var.create 2 in
let w = Inc.map2
  (Inc.Var.watch y) (Inc.Var.watch z) ~f:(+) in
let w_obs = Inc.observe w in
Inc.Var.set x 3;
Inc.stabilize ();

print_int (Inc.Observer.value_exn w_obs)
```

“spreadsheet”

“modifables”
“cells”
Programming with “computation graphs”

Self-Adjusting Computation [Acar ‘05] (Incremental, an OCaml library)

```
let x = Inc.Var.create 1 in
let y = Inc.map2 (Inc.Var.watch x) (Inc.Var.watch x) ~f:( + ) in
let z = Inc.Var.create 2 in
let w = Inc.map2 (Inc.Var.watch y) (Inc.Var.watch z) ~f:( + ) in
let w_obs = Inc.observe w in
Inc.Var.set x 3;
Inc.stabilize ();

print_int (Inc.Observer.value_exn w_obs)
```
Programming with “computation graphs”

Self-Adjusting Computation [Acar ‘05] (Incremental, an OCaml library)

let x = Inc.Var.create 1 in
let y = Inc.map2 (Inc.Var.watch x) (Inc.Var.watch x) ~f:( + ) in
let z = Inc.Var.create 2 in
let w = Inc.map2 (Inc.Var.watch y) (Inc.Var.watch z) ~f:( + ) in
let w_obs = Inc.observe w in
Inc.Var.set x 3;
Inc.stabilize ();
print_int (Inc.Observer.value_exn w_obs)

Muroya (RIMS, Kyoto U. & U. B’ham.)
Programming with “computation graphs”

Self-Adjusting Computation [Acar ‘05] 

(Incremental, an OCaml library)

```
let x = Inc.Var.create 1 in
let y = Inc.map2
  (Inc.Var.watch x)
  (Inc.Var.watch x) ~f:( + ) in
let z = Inc.Var.create 2 in
let w = Inc.map2
  (Inc.Var.watch y)
  (Inc.Var.watch z) ~f:( + ) in
let w_obs = Inc.observe w in
Inc.Var.set x 3;
Inc.stabilize ();

print_int
  (Inc.Observer.value_exn w_obs)
```

observation

value extraction
Programming with “computation graphs”

Probabilistic Programming

[program]

construction
stochastic model + observations

[run-time system]

manipulation
stochastic inference
value extraction
posterior distribution
Programming with “computation graphs”

TensorFlow

imperative parameter update on machine-learning model

Self-Adjusting Computation

change propagation on acyclic dependency graph

Probabilistic Programming

inference on stochastic model
token-passing semantics without rewriting

... result given by the token

+ selective rewriting

token-passing semantics with rewriting

... result given by a diagram

programming with “computation graphs”

... result being value with computation graph
## Token-passing semantics *with & without* rewriting

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Program</strong></td>
<td><strong>Diagram</strong></td>
</tr>
<tr>
<td><strong>Redex search</strong></td>
<td><strong>Token passing</strong></td>
</tr>
<tr>
<td><strong>Reduction</strong></td>
<td><strong>Diagram rewriting</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Computation graphs</th>
<th><strong>Selective diagram rewriting</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Construction</strong></td>
<td><strong>Diagram rewriting</strong></td>
</tr>
<tr>
<td><strong>Manipulation</strong></td>
<td><strong>Diagram rewriting</strong></td>
</tr>
<tr>
<td><strong>Value extraction</strong></td>
<td><strong>Token data</strong></td>
</tr>
</tbody>
</table>

- Muroya (RIMS, Kyoto U. & U. B’ham.)
Programming with “computation graphs”

<table>
<thead>
<tr>
<th>result</th>
<th>value with computation graph</th>
</tr>
</thead>
</table>

Idealised TensorFlow [−, Cheung&Ghica ‘18]

- functional parameter update on machine-learning model

Synchronous Self-Adjusting Computation

- change propagation on cyclic dependency graph

- construction
- manipulation
Synchronous Self-Adjusting Computation

live demo: https://cwtsteven.github.io/GoI-SAC-Visualiser/

\[
\begin{align*}
\text{x} & = \{1\} \\
\text{y} & = \text{x} + \text{x} \\
\text{z} & = \{2\} \\
\text{w} & = \text{y} + \text{z} \\
\text{link x to 3;}
\end{align*}
\]

\[
\begin{align*}
\_ & = \text{step ()} \\
\_ & = \text{step ()} \\
\text{w};
\end{align*}
\]

**construction**
- cyclic dependency graph with “modifiables/cells”

**manipulation**
- cell-wise change propagation

multiple independent tokens without rewriting
Synchronous Self-Adjusting Computation

live demo: [https://cwtsteven.github.io/GoI-SAC-Visualiser/](https://cwtsteven.github.io/GoI-SAC-Visualiser/)

(*) alternating signal *)

\[ x = \{\text{true}\} \]
\[
\text{link } x \text{ to } \sim x; \\
_ = \text{step} () \\
_ = \text{step} ()
\]

construction
cyclic dependency graph
with “modifiabes/cells”

manipulation
cell-wise change propagation

multiple independent tokens
without rewriting
token-passing semantics \textit{without} rewriting

... result given by the token

\begin{itemize}
  \item \textbf{selective rewriting}
\end{itemize}

programming with \textit{“computation graphs”}

... result being value with computation graph

given by the token and a diagram with \textit{“cells”}

Muroya (RIMS, Kyoto U. & U. B’ham.)
## Token-passing semantics with & without rewriting

<table>
<thead>
<tr>
<th>Program</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation</td>
<td></td>
</tr>
<tr>
<td>redex search</td>
<td>token passing</td>
</tr>
<tr>
<td>reduction</td>
<td>diagram rewriting</td>
</tr>
<tr>
<td>Computation graphs</td>
<td></td>
</tr>
<tr>
<td>construction</td>
<td>selective diagram rewriting</td>
</tr>
<tr>
<td>manipulation</td>
<td>diagram rewriting</td>
</tr>
<tr>
<td>value extraction</td>
<td>token data</td>
</tr>
</tbody>
</table>
Directions

- sit back and lay the foundation?
  - “cells”, special constants
    - shared but never duplicated
    - blocking rewrite

- more applications?
  - differentiating (higher-order) computation graphs
  - digesting meta-level stochastic inference