A Graph-Rewriting Perspective of the Beta-Law

Dan R. Ghica
Todd Waugh Ambridge
(University of Birmingham)

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

work in progress

S-REPLS 9 (Univ. Sussex), 25 May 2018
Equivalence of programs

t = u

Syntactical equation

Do \( t \) and \( u \) denote the same (mathematical) object?

Denotational equality

Given any “closing” context \( C \), do evaluations of \( C[t] \) and \( C[u] \) yield the same value?

Operational equivalence
Equivalence of programs

- **Syntactical equation**
  - $t = u$

- **Operational equivalence**
  - Graphically?

- **Denotational equality**
  - Do $t$ and $u$ denote the same (mathematical) object?

Given *any* “closing” context $C$, do evaluations of $C[t]$ and $C[u]$ yield the same value?
call-by-value equational theory \[\text{[Plotkin '75]}\] contextual (operational) equivalence
call-by-value equational theory

contextual (operational) equivalence

\[ \text{Plotkin '75} \]

\[
\begin{align*}
\text{t} &:= x \mid \text{Ax} \cdot \text{t} \mid \text{tt} \mid \text{clf} \\
\text{v} &:= x \mid \text{Ax} \cdot \text{t} \mid \text{clf}
\end{align*}
\]

\[
\begin{align*}
\text{Ax} \cdot \text{M} = \text{v} \Rightarrow \text{Ax} \cdot \text{M}[y/x] &\quad \alpha \\
(\text{Ax} \cdot \text{t}) \text{v} = \text{v} \cdot \text{t}[v/x] &\quad \beta \\
f \cdot c = \text{v} \cdot [f, c] &\quad \gamma
\end{align*}
\]

\[
\begin{align*}
t = \text{v} \cdot \text{u} &\Rightarrow \text{Cnf} \\
\text{c[t]} = \text{v} \cdot \text{c[u]} &\quad \text{Refl} \\
t = \text{v} \cdot \text{t} &\quad \text{Symm} \\
t_1 = t_2 &\quad t_2 = v \cdot t_3 &\quad \text{Trans}
\end{align*}
\]
$t \equiv^v u \iff \forall c \text{ s.t. } \text{cl}[t] \text{ and } \text{cl}[u] \text{ are closed,}
\begin{align*}
\text{Eval}^v(\text{cl}[t]) \text{ is defined } & \iff \text{Eval}^v(\text{cl}[u]) \text{ is defined} \\
\text{Moreover, } \text{Eval}^v(\text{cl}[t]) &= \text{Eval}^v(\text{cl}[u]) \\
\text{if } \text{Eval}^v(\text{cl}[t]) \text{ or } \text{Eval}^v(\text{cl}[u]) \text{ is a basic constant}
\end{align*}
call-by-value equational theory

soundness

[Plotkin '75]

contextual (operational) equivalence

\[
\begin{align*}
  t &::= x | \lambda x.t | t t | c | f \\
  v &::= x | \lambda x.t | c | f \\
  \lambda x.M &\equiv v \lambda y.M[y/x] \\
  (\lambda x.t)v &\equiv t[v/x] \\
  f c &\equiv \llbracket f, c \rrbracket
\end{align*}
\]

Congruence Rules:

- \( t =_v u \)
  - \( c[t] =_v c[u] \)

Reflection Rules:

- \( t =_v t \)
  - \( u =_v t \)
  - \( t =_v u \)

Symmetry Rules:

- \( t =_v t \)

Transitivity Rules:

- \( t =_v t \)

SECD machine

SECD machine

\[
t =_v u \iff \forall c \text{ s.t. } c[t] \text{ and } c[u] \text{ are closed, }
\]

\[
\text{Eval}_v(c[t]) \text{ is defined } \iff \text{Eval}_v(c[u]) \text{ is defined}
\]

Moreover, \( \text{Eval}_v(c[t]) = \text{Eval}_v(c[u]) \)

if \( \text{Eval}_v(c[t]) \) or \( \text{Eval}_v(c[u]) \) is a basic constant
call-by-value equational theory

\[ t ::= x | \lambda x.t | tt | c \triangleright f \]

\[ v ::= x | \lambda x.t | c \triangleright f \]

\[ \lambda x.M =_v \lambda y.M[y/x] \]

\[ (\lambda x.t)v =_v t[v/x] \]

\[ f \cdot c =_v [f,c] \]

graphically

contextual (operational) equivalence

\[ t =_v u \quad \text{Cong} \]

\[ t =_v t \quad \text{Ref} \]

\[ t =_v u \quad \text{Symm} \]

\[ t_1 =_v t_2 \rightarrow t_2 =_v t_3 \quad \text{Trans} \]

graph-rewriting machine

\[ t =_v u \quad \triangleright c \quad \text{s.t. } \text{C[t]} \text{ and } \text{C[u]} \text{ are closed, } \]

\[ \text{Eval}_v(\text{C[t]}) \text{ is defined } \Leftrightarrow \text{Eval}_v(\text{C[u]}) \text{ is defined} \]

Moreover, \[ \text{Eval}_v(\text{C[t]}) = \text{Eval}_v(\text{C[u]}) \]

if \[ \text{Eval}_v(\text{C[t]}) \text{ or } \text{Eval}_v(\text{C[u]}) \] is a basic constant
call-by-value graph-equational theory graphically contextual (operational) equivalence
call-by-value graph-equational theory

graphically

contextual (operational) equivalence

all and only values are duplicable
call-by-value graph-equational theory

\[
x^t = \vphantom{\lambda} \quad (\lambda x.t)^t = \quad c^t = \\
(x^t =) \quad (\lambda x.t)^t = \quad c^t =
\]

\[
(t \ u)^t = \quad f^t =
\]

\[
\begin{align*}
\lambda x. M &= \nu \lambda y. M[y/x] & \alpha \\
(\lambda x.t) \nu &= \nu \ t[v/x] & \beta \\
f \ c &= \nu \ [f, c] & \delta
\end{align*}
\]

\[
\begin{align*}
t &= \nu \ u & \text{Cong} \\
C[t] &= \nu \ C[u] & \text{Refl} \\
t &= \nu \ t & \text{Symm} \\
\]

\[
\begin{align*}
t &= \nu \ u & \text{Cong} \\
u &= \nu \ t & \text{Refl} \\
t &= \nu \ t & \text{Symm} \\
t_1 &= \nu \ t_2 & t_2 = \nu \ t_3 & \text{Trans}
\end{align*}
\]

contextual (operational) equivalence
call-by-value
graph-equational
theory

contextual
(operational)
equivalence

graphically

alpha-law: trivial
beta-law: refined
(cf. explicit substitution)
call-by-value graph-equational theory

alpha-law: trivial
beta-law: refined
(cf. explicit substitution)
call-by-value graph-equational theory

contextual (operational) equivalence

graphically

alpha-law: trivial
beta-law: refined
(cf. explicit substitution)
call-by-value graph-equational theory

contextual (operational) equivalence

graphically

alpha-law: trivial
beta-law: refined
(cf. explicit substitution)
call-by-value graph-equational theory

contextual (operational) equivalence

graphically

alpha-law: trivial

beta-law: refined (cf. explicit substitution)

SECD machine

\[ t \equiv_v u \iff \exists C \text{ s.t. } C[t] \text{ and } C[u] \text{ are closed,} \]

\[ \text{Eval}_v(C[t]) \text{ is defined } \iff \text{Eval}_v(C[u]) \text{ is defined} \]

Moreover, \[ \text{Eval}_v(C[t]) = \text{Eval}_v(C[u]) \]

if \( \text{Eval}_v(C[t]) \) or \( \text{Eval}_v(C[u]) \) is a basic constant
call-by-value graph-equational theory

graphically

graph-contextual (operational) equivalence

alpha-law: trivial
beta-law: refined (cf. explicit substitution)

graph-rewriting machine

G ≅_α H \iff \forall G[H] s.t. \begin{array}{l} A_1[G] \\ \hline \end{array} \quad \text{and} \quad \begin{array}{l} A_1[H] \\ \hline \end{array}

Eval_α(A_1[G]) is defined \iff Eval_α(A_1[H]) is defined

Moreover, if defined, Eval_α(A_1[G]) = Eval_α(A_1[H])
Call-by-value graph-equational theory

Graphically

Graph-contextual (operational) equivalence

Alpha-law: trivial
Beta-law: refined (cf. explicit substitution)

\( x^t = \quad (\lambda x.t)^t = \quad c^t = \)

\( (t \ u)^t = \quad t^t \ u^t \quad f^t = \)

\( G \models_\alpha H \iff \forall G_1 \text{ s.t. } \overline{G_1[\alpha]} \text{ and } \overline{G_1[H]}, \)

\( \text{Eval}_\alpha(\overline{G_1[\alpha]}) \text{ is defined } \iff \text{Eval}_\alpha(\overline{G_1[H]}) \text{ is defined} \)

Moreover, if defined, \( \text{Eval}_\alpha(\overline{G_1[\alpha]}) = \text{Eval}_\alpha(\overline{G_1[H]}) \)

dGoI machine
- stack of closures
- environment
- control string
- dump

- graph
- evaluation control ("token")
- rewriting flag
- computation stack
- box stack
SECD machine

\[
\langle \square, \emptyset, t, \square \rangle \quad \langle t^+, \square, \ast: \square, \ast: \square \rangle
\]

\[
\langle \langle u, E \rangle, \emptyset, \square, \square \rangle \quad \langle H, \square, \kappa: \square, !: \square \rangle
\]
SECD machine

\[ \langle \text{program}, t \rangle \]

\[ \downarrow \]

\[ \langle \text{H}, \text{E}, \text{v}, \text{c} \rangle \]

\[ \text{Eval}_v(t) := \text{Subst}(u, \text{E}) \]

\[ \text{Eval}_\alpha(t) := \text{v} \]

\textbf{dGoI machine}

\[ \langle \text{t}^+, \text{E}, \text{v}, \text{c} \rangle \]

\[ \downarrow \]

\[ \langle \text{H}, \text{E}, \text{v}, \text{c} \rangle \]

\[ \text{v} ::= \lambda | c \]
dGoI-machine transitions
call-by-value\n\textit{graph}-equational\ntheory

\textbf{graph-contextual (operational) equivalence}

\textbf{graphically}

alpha-law: trivial
beta-law: refined (cf. explicit substitution)

\textbf{dGoI machine}

\begin{align*}
\Gamma \vdash_{\alpha} H & \iff \forall \forall [\emptyset] \text{ s.t. } \frac{\Delta[\emptyset]}{\Gamma} \text{ and } \frac{\Delta[H]}{\Gamma}, \\
\text{Eval}_a(\Delta[\emptyset]) \text{ is defined} & \iff \text{Eval}_a(\Delta[H]) \text{ is defined} \\
\text{Moreover, if defined, } \text{Eval}_a(\Delta[\emptyset]) & = \text{Eval}_a(\Delta[H])
\end{align*}
call-by-value

\textit{graph}-equational

theory

\begin{align*}
x^t &= \quad (\lambda x.t)^t &= \quad t^t = \\
(t \cdot u)^t &= \quad t^t \cdot t^t = \\
\end{align*}

\textbf{alpha-law: trivial}

\textbf{beta-law: refined (cf. explicit substitution)}

dGoI machine

soundness

\textit{graph}-contextual (operational) equivalence

\[ G =_{\alpha} H \quad \implies \quad G \cong_{\alpha} H \]

\[ G \cong_{\alpha} H \quad \iff \quad \forall \overline{A}[\emptyset] \text{ st. } \overline{A}[G] \text{ and } \overline{A}[H] , \]

\[ \text{Eval}_\alpha(\overline{A}[G]) \text{ is defined } \iff \text{Eval}_\alpha(\overline{A}[H]) \text{ is defined} \]

Moreover, if defined, \[ \text{Eval}_\alpha(\overline{A}[G]) = \text{Eval}_\alpha(\overline{A}[H]) \]
call-by-value
graph-equational
theory

soundness
graphically

graph-contextual
(operational)
equivalence

alpha-law: trivial
beta-law: refined (cf. explicit substitution)

1. lift an axiom to a binary relation on
(dGol-machine) states
1. Lift an axiom to a binary relation on (dGoI-machine) states

2. Show the binary relation is a "U-simulation"

Prop. \( R_x \) is a U-simulation \( \Rightarrow G \cong \alpha H \)
1. Lift an axiom to a binary relation on (dGoI-machine) states.

2. Show the binary relation is a "U-simulation".
1. Lift an axiom to a binary relation on (dGoI-machine) states

2. Show the binary relation is a “U-simulation”

Simulation

...until the difference is reduced
“Until the difference is reduced”
1. Lift an **axiom** to a binary relation on (dGoI-machine) states

2. Show the binary relation is a “**U-simulation**”

...until the **difference** is reduced
call-by-value

graph-equational theory

soundness

graphically

graph-contextual (operational) equivalence

alpha-law: trivial
beta-law: refined (cf. explicit substitution)

modular proof using U-simulations

dGol machine
Equivalence of programs

**syntactical equation**

\[ t = u \] graphically?

**operational equivalence**

Given any “closing” context \( C \), do evaluations of \( C[t] \) and \( C[u] \) yield the same value?

**denotational equality**

Do \( t \) and \( u \) denote the same (mathematical) object?
Equivalence of programs

- **Syntactical equation**: $t = u$
- **Operational equivalence**: Graphically?
- **Denotational equality**: Do $t$ and $u$ denote the same (mathematical) object?

Given any “closing” context $C$, do evaluations of $C[t]$ and $C[u]$ yield the same value?

**Modular proof of soundness using U-simulations**

$$G =_a H \implies G \simeq_a H$$
so what?
Equivalence of programs

Given any “closing” context \( C \), do evaluations of \( C[t] \) and \( C[u] \) yield the same value?

modular proof of soundness using \( U \)-simulations

syntactical equation

operational equivalence

\[ \begin{align*}
G =_{\alpha} H & \implies G =_{\alpha} H \\
\text{t = u} & \text{graphically?}
\end{align*} \]
Equivalence of programs

\[ t = u \]

\[ G = \alpha H \implies G \equiv \alpha H \]

**syntactical equation**

**graphically?**

**related proof techniques:**

- logical relations
- applicative bisimulations
- envirionmental bisimulations...

---

**operational equivalence**

**modular proof of soundness using \( U\)-simulations**

Given *any* “closing” context \( C \), do evaluations of \( C[t] \) and \( C[u] \) yield the same value?
Equivalence of programs

Given any "closing" context $C$, do evaluations of $C[t]$ and $C[u]$ yield the same value?

**modular** proof of soundness using $U$-simulations

$G = _\alpha H \Rightarrow G = _\alpha H$

semantical criteria of primitive operations (function constants) to preserve beta-law?

syntactical equation

graphically?
Equivalence of programs

Given any “closing” context $C$, do evaluations of $C[t]$ and $C[u]$ yield the same value?

Equivalence of programs

Given any “closing” context $C$, do evaluations of $C[t]$ and $C[u]$ yield the same value?

Given any “closing” context $C$, do evaluations of $C[t]$ and $C[u]$ yield the same value?

**Modular proof of soundness using $U$-simulations**

$G = a \Rightarrow H$ implies $G \cong a \Rightarrow H$

Cost-sensitive equivalence? (cf. [Schmidt-Schauss & Dallmeyer, WPTE ’17])

Syntactical equation graphically?