work in progress

A Graph-Rewriting Perspective of the Beta-Law

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

<u>Koko Muroya</u>
(University of Birmingham
& RIMS, Kyoto University)

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

Equivalence of programs

syntactical equation

`t = u`

denotational equality

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

operational equivalence

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

Equivalence of programs

denotational equality

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

syntactical equation

`t = u`

graphically?

operational equivalence

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


call-by-value equational theory
[Plotkin ‘75]
contextual (operational) equivalence

call-by-value equational theory $\longrightarrow$ contextual (operational) equivalence

[Plotkin ‘75]

$[\![\_,\_]\!] : \{\text{function constants}\} \times \{\text{basic constants}\}$
$\longrightarrow \{\text{closed values}\}$

$$t ::= x \mid \lambda x.t \mid t\,t \mid c \mid f$$

$$v ::= x \mid \lambda x.t \mid c \mid f$$

$$\frac{}{\lambda x.M =_v \lambda y.M[y/x]}\,\alpha \qquad \frac{}{(\lambda x.t)\,v =_v t[v/x]}\,\beta \qquad \frac{}{f\,c =_v [\![f,c]\!]}\,\delta$$

$$\frac{t =_v u}{C[t] =_v C[u]}\,\mathrm{Cong} \qquad \frac{}{t =_v t}\,\mathrm{Refl} \qquad \frac{t =_v u}{u =_v t}\,\mathrm{Symm} \qquad \frac{t_1 =_v t_2 \quad t_2 =_v t_3}{t_1 =_v t_3}\,\mathrm{Tran}$$

call-by-value equational theory

[Plotkin '75]

contextual (operational) equivalence

$t \cong_v u \overset{\Delta}{\Leftrightarrow} \forall C$ s.t. $C[t]$ and $C[u]$ are closed,

$Eval_v(C[t])$ is defined $\Leftrightarrow$ $Eval_v(C[u])$ is defined

Moreover, $Eval_v(C[t]) = Eval_v(C[u])$

if $Eval_v(C[t])$ or $Eval_v(C[u])$ is

a basic constant

SECD machine

call-by-value equational theory $\xrightarrow[\text{[Plotkin '75]}]{\text{soundness}}$ contextual (operational) equivalence

$$t ::= x \mid \lambda x.t \mid t\,t \mid c \mid f$$

$$v ::= x \mid \lambda x.t \mid c \mid f$$

$$\frac{}{\lambda x.M =_v \lambda y.M[y/x]}\,\alpha \qquad \frac{}{(\lambda x.t)\,v =_v t[v/x]}\,\beta \qquad \frac{}{f\,c =_v [\![f, c]\!]}\,\delta$$

$$\frac{t =_v u}{C[t] =_v C[u]}\,\mathrm{Cong} \qquad \frac{}{t =_v t}\,\mathrm{Refl} \qquad \frac{t =_v u}{u =_v t}\,\mathrm{Symm} \qquad \frac{t_1 =_v t_2 \quad t_2 =_v t_3}{t_1 =_v t_3}\,\mathrm{Tran}$$

$t \cong_v u \overset{\Delta}{\Leftrightarrow} \forall C$ s.t. $C[t]$ and $C[u]$ are closed,

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

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

if $\mathrm{Eval}_v(C[t])$ or $\mathrm{Eval}_v(C[u])$ is a basic constant

SECD machine

call-by-value equational theory

graphically →

contextual (operational) equivalence

$$t ::= x \mid \lambda x.t \mid t\,t \mid c \mid f$$

$$v ::= x \mid \lambda x.t \mid c \mid f$$

$$\frac{}{\lambda x.M =_v \lambda y.M[y/x]}\,\alpha \qquad \frac{}{(\lambda x.t)\,v =_v t[v/x]}\,\beta \qquad \frac{}{f\,c =_v [\![f,c]\!]}\,\delta$$

$$\frac{t =_v u}{C[t] =_v C[u]}\,\mathrm{Cong} \qquad \frac{}{t =_v t}\,\mathrm{Refl} \qquad \frac{t =_v u}{u =_v t}\,\mathrm{Symm} \qquad \frac{t_1 =_v t_2 \quad t_2 =_v t_3}{t_1 =_v t_3}\,\mathrm{Tran}$$

$t \cong_v u \overset{\Delta}{\Leftrightarrow} \forall C$ s.t. $C[t]$ and $C[u]$ are closed,

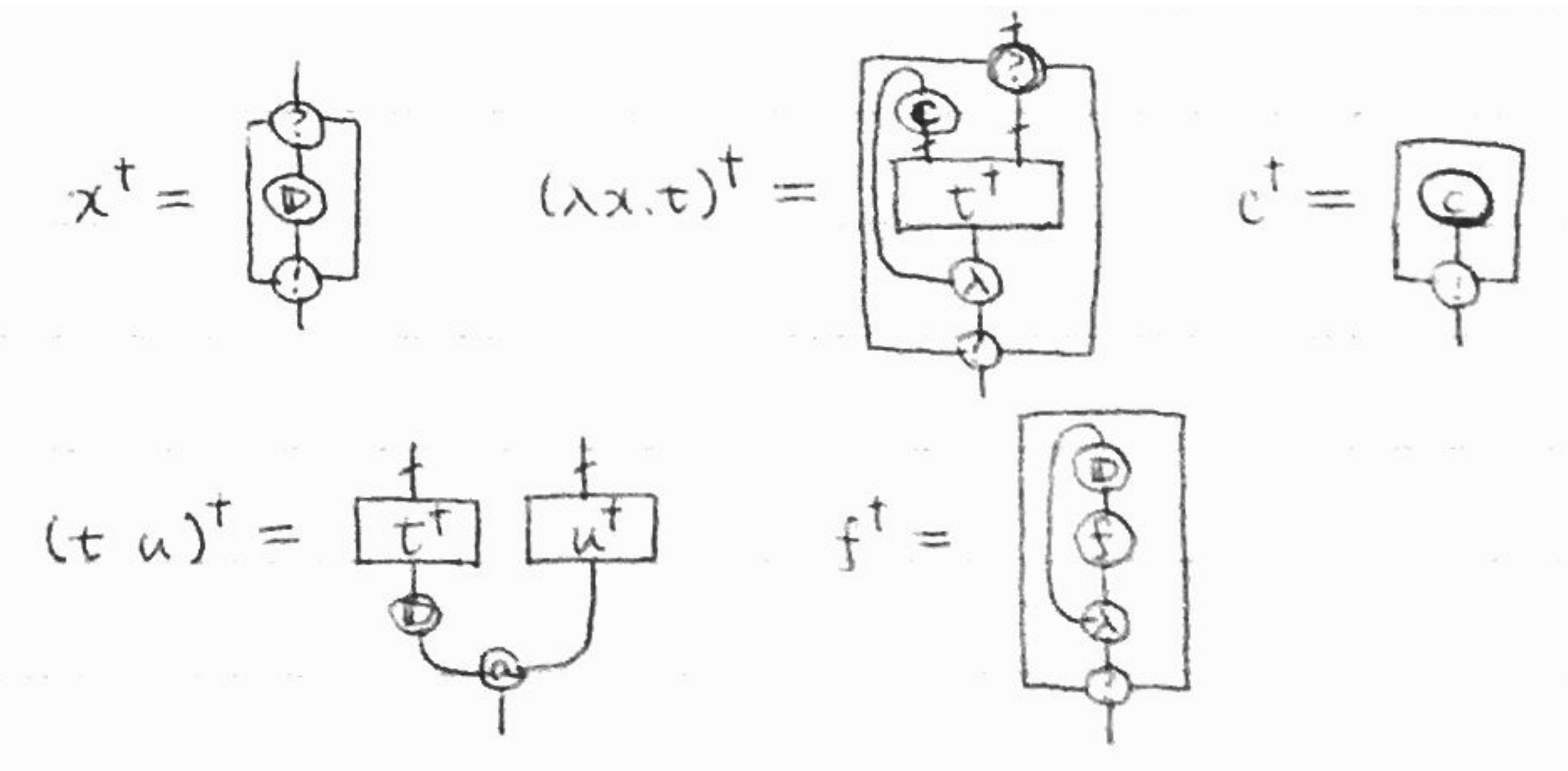
$\mathrm{Eval}_v(C[t])$ is defined $\Leftrightarrow$ $\mathrm{Eval}_v(C[u])$ is defined

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

if $\mathrm{Eval}_v(C[t])$ or $\mathrm{Eval}_v(C[u])$ is a basic constant

graph-rewriting machine

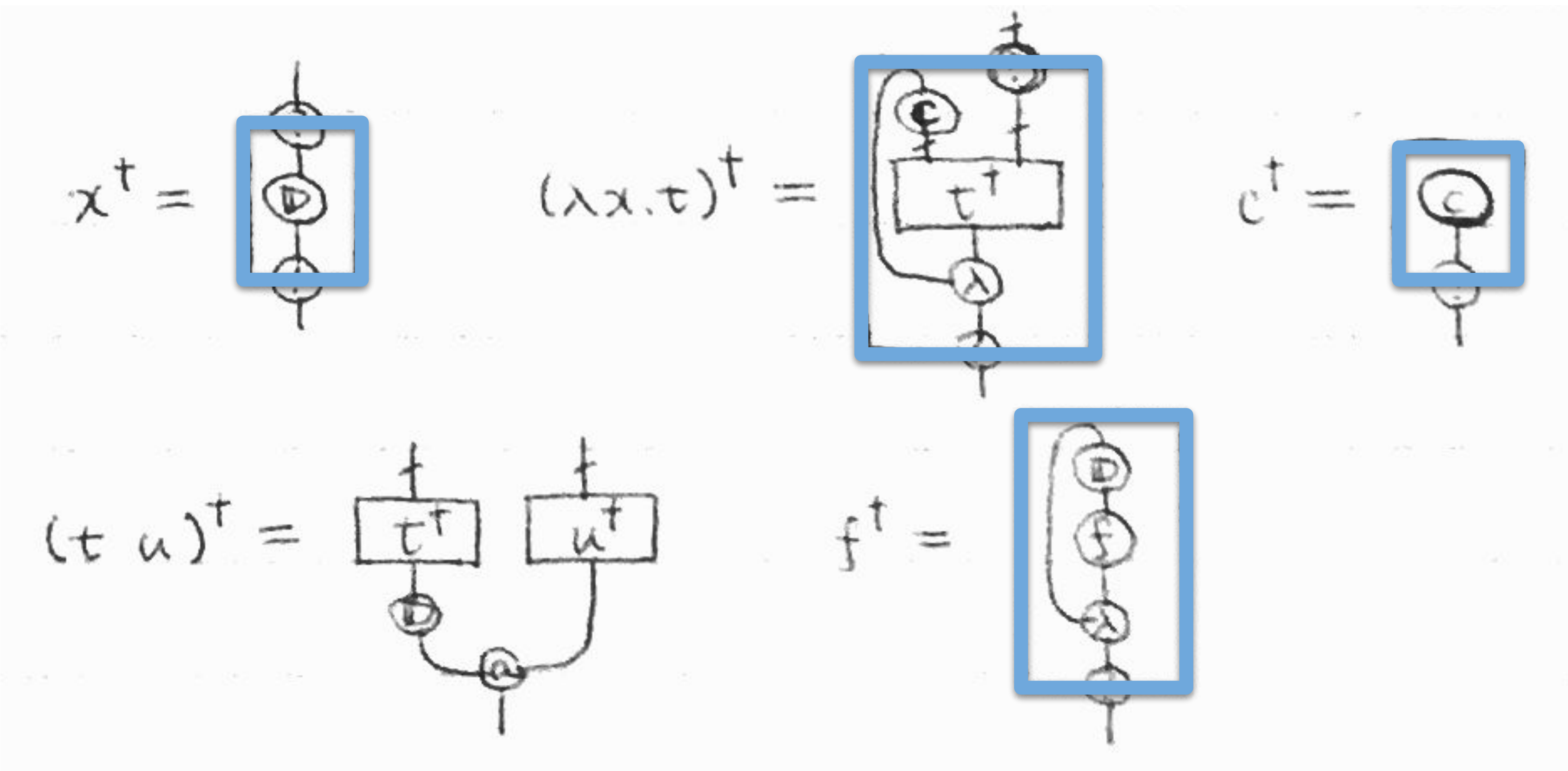
call-by-value graph-equational theory → graphically → contextual (operational) equivalence



call-by-value graph-equational theory

graphically

contextual (operational) equivalence

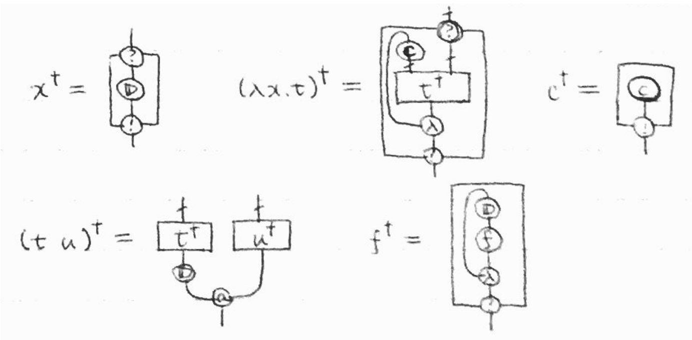


all and only values are duplicable

call-by-value graph-equational theory

graphically →

contextual (operational) equivalence



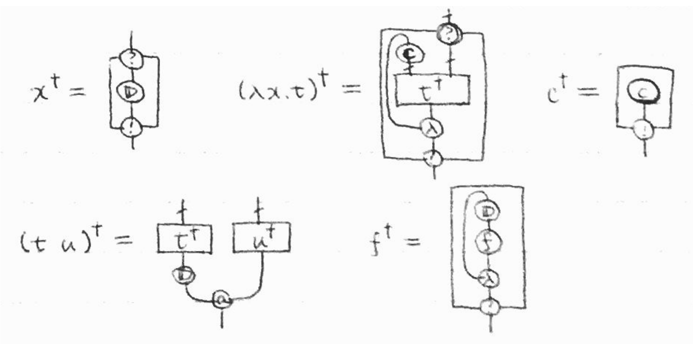
$$\frac{}{\lambda x.M =_v \lambda y.M[y/x]}\ \alpha \qquad \frac{}{(\lambda x.t)\,v =_v t[v/x]}\ \beta \qquad \frac{}{f\,c =_v [\![f, c]\!]}\ \delta$$

$$\frac{t =_v u}{C[t] =_v C[u]}\ \mathrm{Cong} \qquad \frac{}{t =_v t}\ \mathrm{Refl} \qquad \frac{t =_v u}{u =_v t}\ \mathrm{Symm} \qquad \frac{t_1 =_v t_2 \quad t_2 =_v t_3}{t_1 =_v t_3}\ \mathrm{Tran}$$

call-by-value graph-equational theory

graphically →

contextual (operational) equivalence



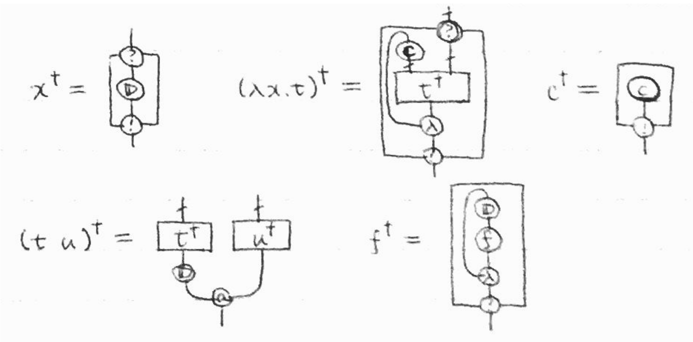
alpha-law: trivial
beta-law: refined
(cf. explicit substitution)

$$\frac{}{\lambda x.M =_v \lambda y.M[y/x]}\,\alpha \qquad \frac{}{(\lambda x.t)\,v =_v t[v/x]}\,\beta \qquad \frac{}{f\,c =_v [\![f, c]\!]}\,\delta$$

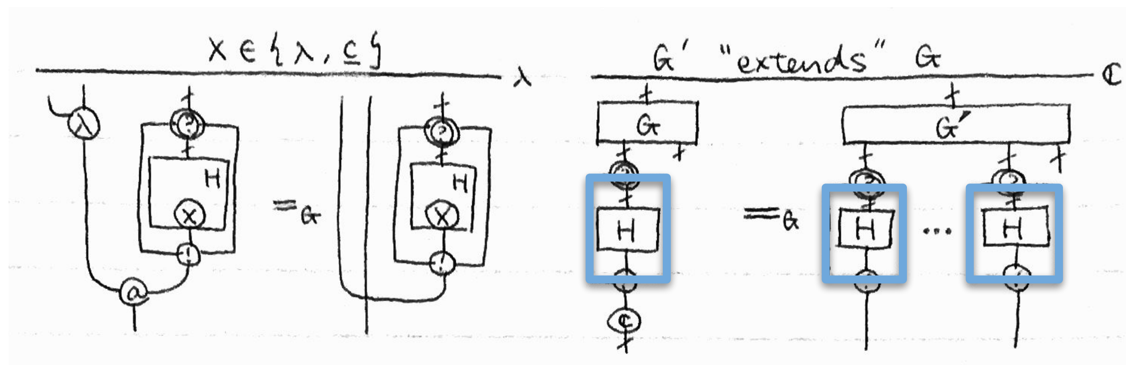
$$\frac{t =_v u}{C[t] =_v C[u]}\,\mathrm{Cong} \qquad \frac{}{t =_v t}\,\mathrm{Refl} \qquad \frac{t =_v u}{u =_v t}\,\mathrm{Symm} \qquad \frac{t_1 =_v t_2 \quad t_2 =_v t_3}{t_1 =_v t_3}\,\mathrm{Tran}$$

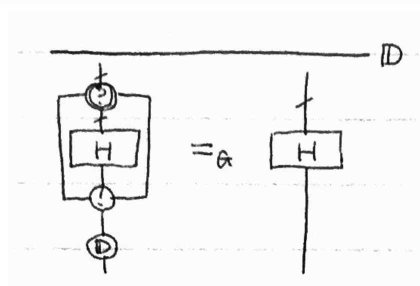
call-by-value graph-equational theory → contextual (operational) equivalence

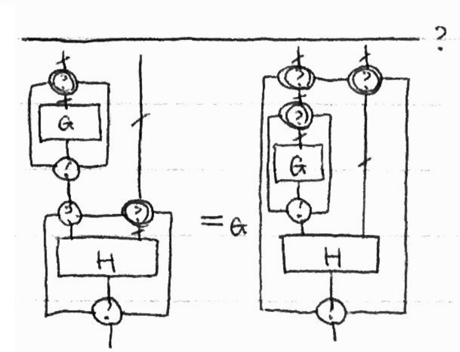
graphically



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





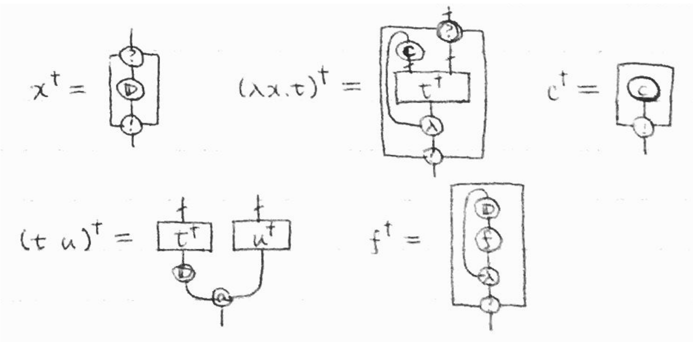


Muroya (U. B'ham. & RIMS, Kyoto U.)

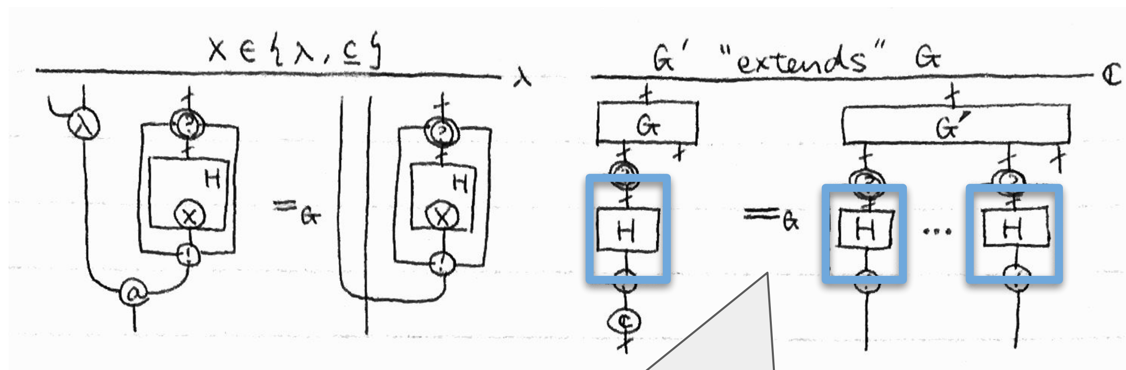
call-by-value graph-equational theory

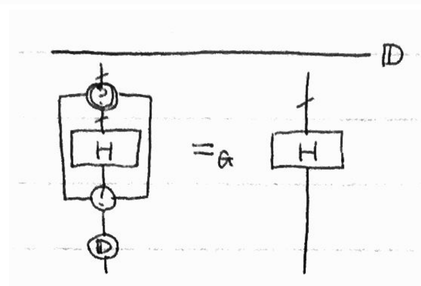
graphically

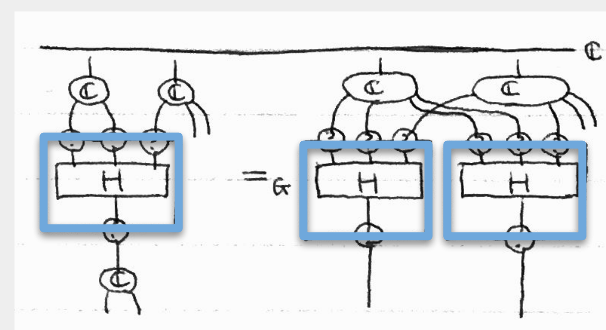
contextual (operational) equivalence



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



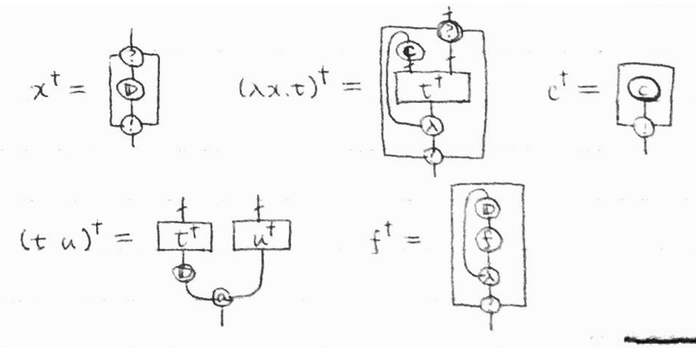




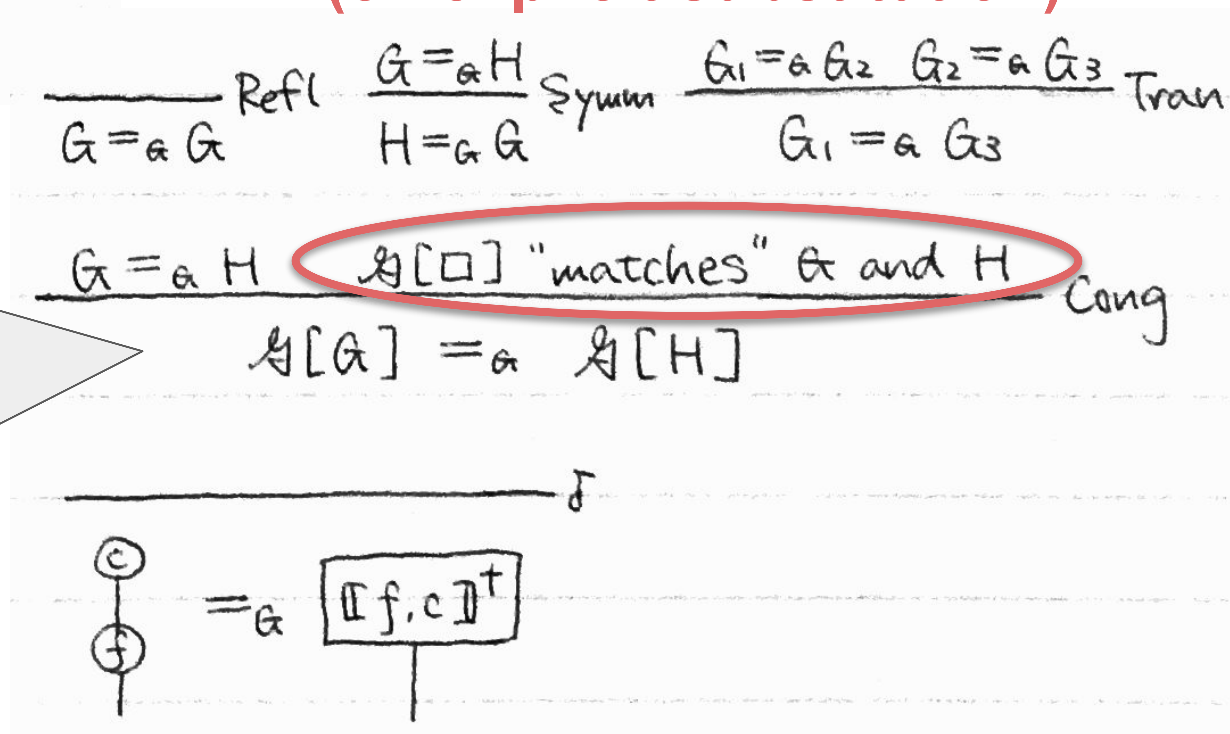
call-by-value graph-equational theory

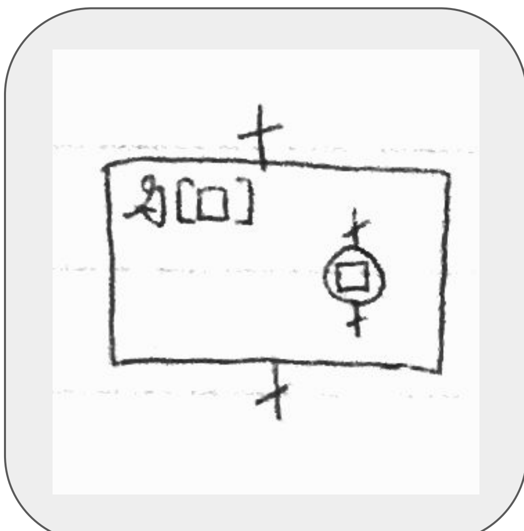
graphically →

contextual (operational) equivalence



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

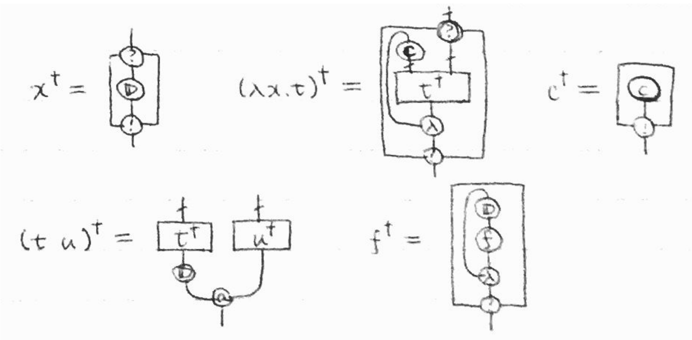




call-by-value graph-equational theory

graphically

contextual (operational) equivalence



alpha-law: trivial

beta-law: refined (cf. explicit substitution)

$t \simeq_v u \overset{\Delta}{\Leftrightarrow} \forall C$ s.t. $C[t]$ and $C[u]$ are closed,

$\mathrm{Evalv}(C[t])$ is defined $\Leftrightarrow$ $\mathrm{Evalv}(C[u])$ is defined

Moreover, $\mathrm{Evalv}(C[t]) = \mathrm{Evalv}(C[u])$

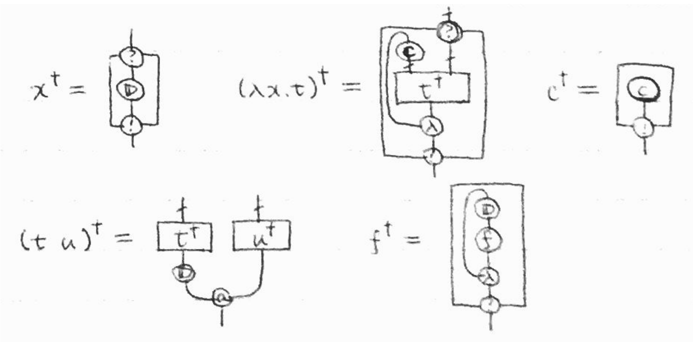
if $\mathrm{Evalv}(C[t])$ or $\mathrm{Evalv}(C[u])$ is a basic constant

SECD machine

call-by-value graph-equational theory

graphically

graph-contextual (operational) equivalence



alpha-law: trivial

beta-law: refined (cf. explicit substitution)

$G \cong_{a} H \overset{\Delta}{\Leftrightarrow} \forall \mathcal{C}[\square]$ s.t. $\boxed{\mathcal{C}[G]}$ and $\boxed{\mathcal{C}[H]}$,

$\mathrm{Eval}_a(\mathcal{C}[G])$ is defined $\Leftrightarrow$ $\mathrm{Eval}_a(\mathcal{C}[H])$ is defined

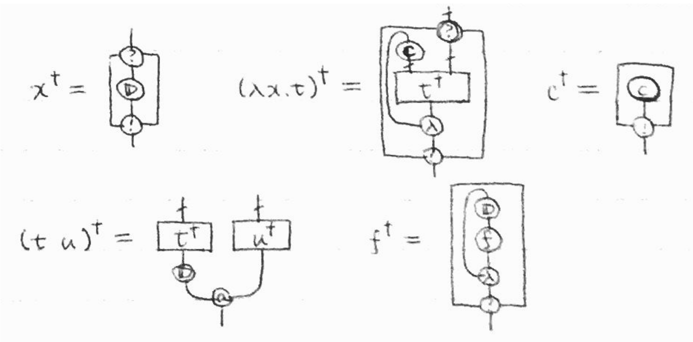
Moreover, if defined, $\mathrm{Eval}_a(\mathcal{C}[G]) = \mathrm{Eval}_a(\mathcal{C}[H]$

graph-rewriting machine

call-by-value graph-equational theory

graphically →

graph-contextual (operational) equivalence



alpha-law: trivial

beta-law: refined (cf. explicit substitution)

$G \cong_a H \Leftrightarrow \forall \mathcal{G}[\square]$ s.t. $\boxed{\mathcal{G}[G]}$ and $\boxed{\mathcal{G}[H]}$,

$Eval_a(\mathcal{G}[G])$ is defined $\Leftrightarrow$ $Eval_a(\mathcal{G}[H])$ is defined

Moreover, if defined, $Eval_a(\mathcal{G}[G]) = Eval_a(\mathcal{G}[H]$

dGoI machine

SECD machine

dGoI machine

t : program

$$\langle \square, \phi, t, \square \rangle \Downarrow \vdots$$

$$\langle \boxed{t^{\dagger}}, \square, \star : \square, \star : \square \rangle \downarrow \vdots$$

- stack of closures
- environment
- control string
- dump

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

SECD machine

dGoI machine

t : program

$$\langle \square, \phi, t, \square \rangle$$

$$\Downarrow$$

$$\vdots$$

$$\langle \langle u, E \rangle, \phi, \square, \square \rangle$$

$$\langle \boxed{t^{\dagger}}, \square, \star : \square, \star : \square \rangle$$

$$\downarrow$$

$$\vdots$$

$$\langle \boxed{H}, \square, \kappa : \square, ! : \square \rangle$$

SECD machine

dGoI machine

t : program

$$\langle \square, \phi, t, \square \rangle$$

$$\Downarrow$$

$$\vdots$$

$$\langle \langle u, E \rangle, \phi, \square, \square \rangle$$

$$\mathrm{Eval}_V(t) := \mathrm{Subst}(u, E)$$

$$\langle \boxed{t^\dagger}, \square, \star : \square, \star : \square \rangle$$

$$\downarrow$$

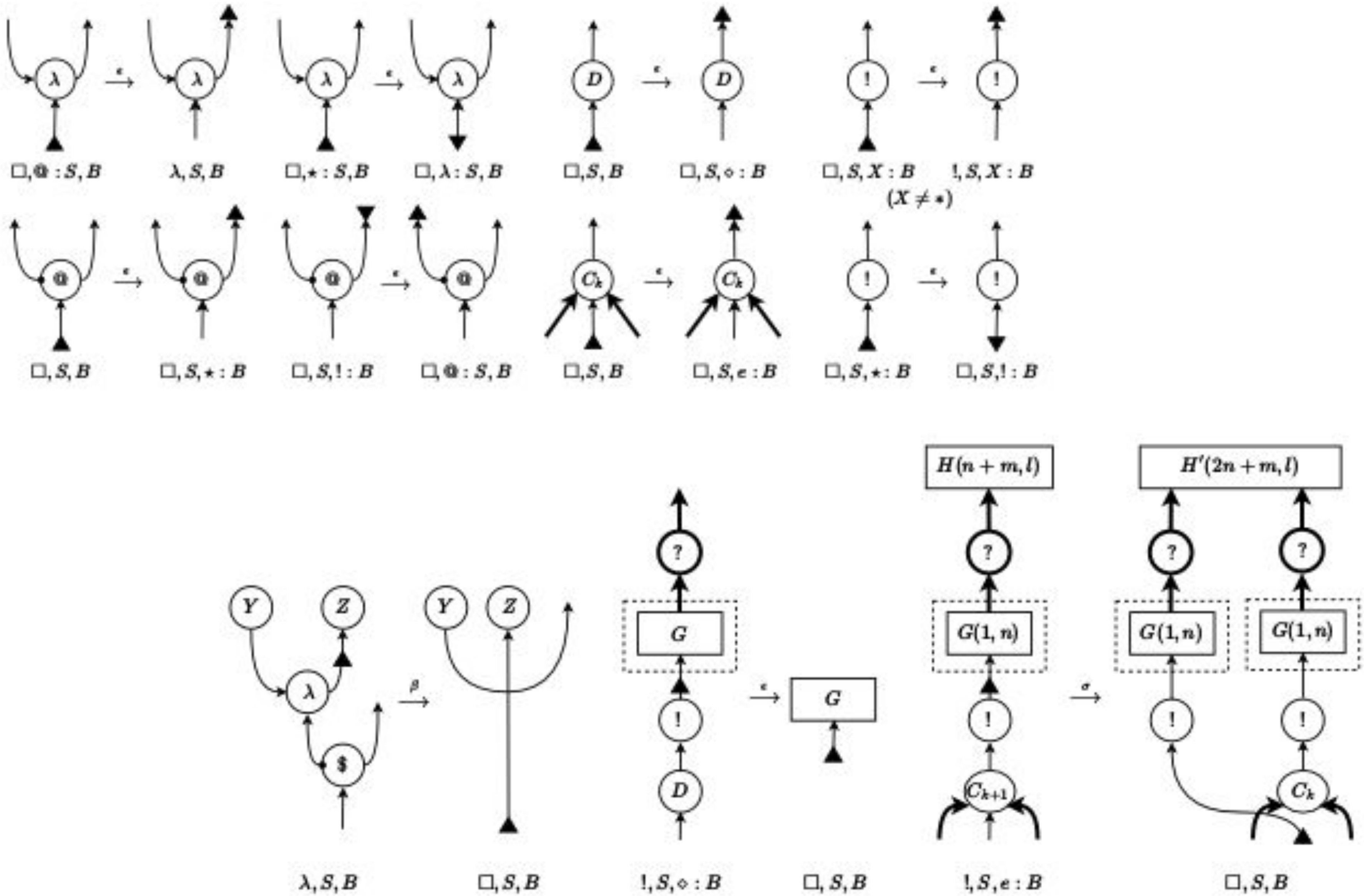
$$\vdots$$

$$\langle \boxed{H}, \square, \kappa : \square, ! : \square \rangle$$

$$\kappa ::= \lambda \mid c$$

$$\mathrm{Eval}_G(t) := \kappa$$

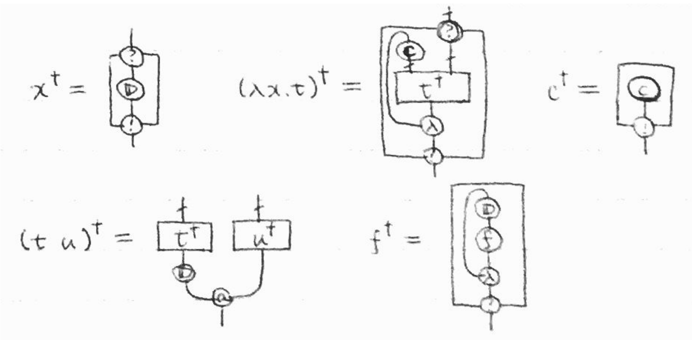
dGoI-machine transitions



call-by-value *graph*-equational theory

graphically →

graph-contextual (operational) equivalence



alpha-law: trivial

beta-law: refined (cf. explicit substitution)

$G \cong_{\mathfrak{a}} H \overset{\Delta}{\Leftrightarrow} {}^{\forall}\mathcal{C}[\square]$ s.t. $\boxed{\mathcal{C}[G]}$ and $\boxed{\mathcal{C}[H]}$,

$\mathrm{Eval}_{\mathfrak{a}}(\mathcal{C}[G])$ is defined $\Leftrightarrow$ $\mathrm{Eval}_{\mathfrak{a}}(\mathcal{C}[H])$ is defined

Moreover, if defined, $\mathrm{Eval}_{\mathfrak{a}}(\mathcal{C}[G]) = \mathrm{Eval}_{\mathfrak{a}}(\mathcal{C}[H]$

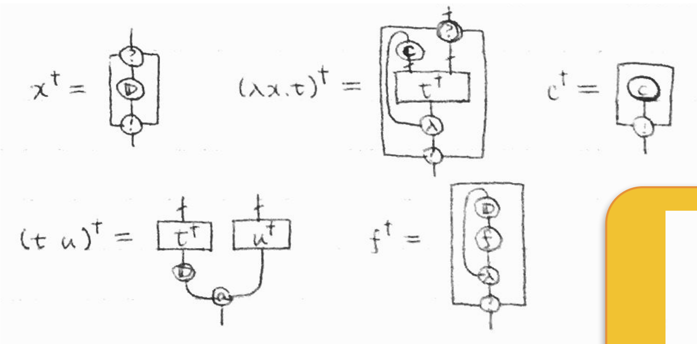
dGoI machine

call-by-value *graph*-equational theory

soundness

graphically

graph-contextual (operational) equivalence



alpha-law: trivial

beta-law: refined (cf. explicit substitution)

$$G =_{\mathfrak{G}} H \Longrightarrow G \cong_{\mathfrak{G}} H$$

$G \cong_{\mathfrak{G}} H \overset{\Delta}{\Leftrightarrow} {}^\forall \mathcal{C}[\square]$ s.t. $\boxed{\mathcal{C}[G]}$ and $\boxed{\mathcal{C}[H]}$,

$\underline{\mathrm{Eval}_{\mathfrak{G}}(\mathcal{C}[G])}$ is defined $\Leftrightarrow$ $\mathrm{Eval}_{\mathfrak{G}}(\mathcal{C}[H])$ is defined

Moreover, if defined, $\mathrm{Eval}_{\mathfrak{G}}(\mathcal{C}[G]) = \mathrm{Eval}_{\mathfrak{G}}(\mathcal{C}[H]$

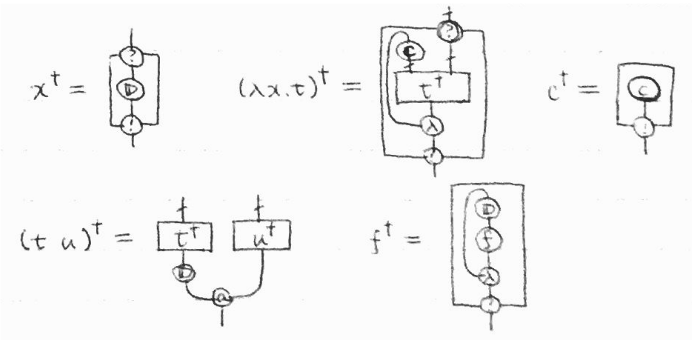
dGoI machine

call-by-value *graph*-equational theory

soundness

graphically

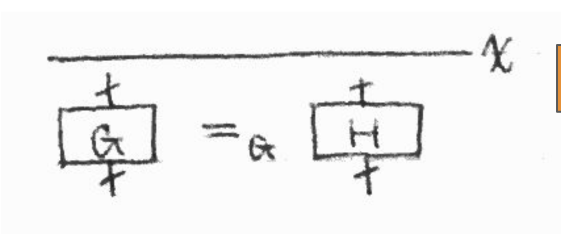
graph-contextual (operational) equivalence

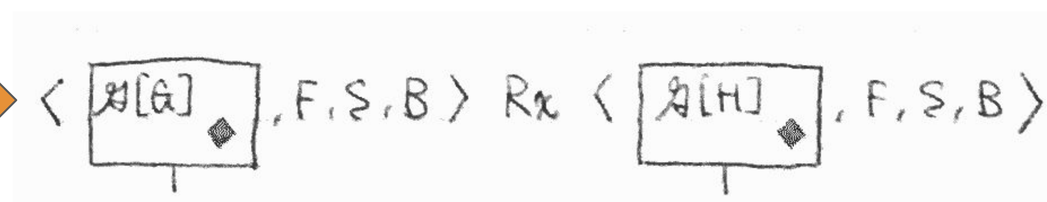


alpha-law: trivial

beta-law: refined (cf. explicit substitution)

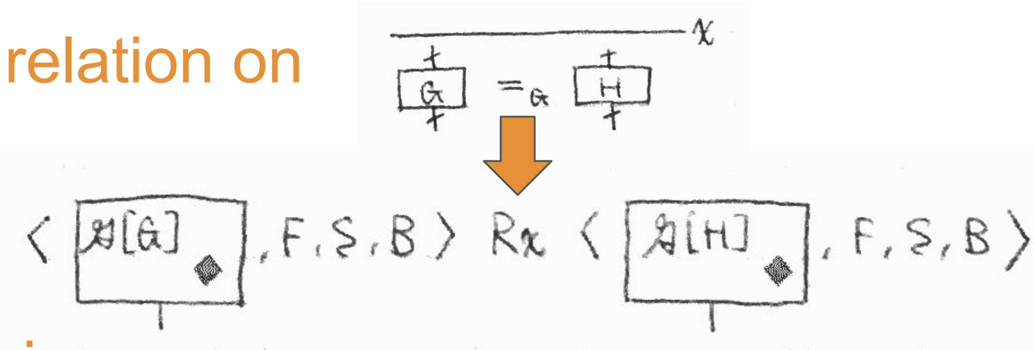
1. lift an axiom to a binary relation on (dGoI-machine) states





call-by-value *graph*-equational theory → **soundness** (graphically) → *graph*-contextual (operational) equivalence

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



2. show the binary relation is a "*U-simulation*"

R is a <u>U-simulation</u> $\overset{\Delta}{\Leftrightarrow}$ (1) if $\sigma_1 \mathrel{R} \sigma_2$ and $\sigma_1 \to \sigma_1'$,

(i) $\exists \sigma_2'.\ \sigma_2 \to \sigma_2'$ and $\sigma_1' \mathrel{R^*} \sigma_2'$

OR

(ii) $\exists \sigma_1''.\ \sigma_1 \to \sigma_1' \to^* \sigma_1''$ and $\sigma_1'' \mathrel{R^*} \sigma_2$

(2) if $\sigma_1 \mathrel{R} \sigma_2$ and $\sigma_1 \not\to$,

(i) $\sigma_2 \not\to$

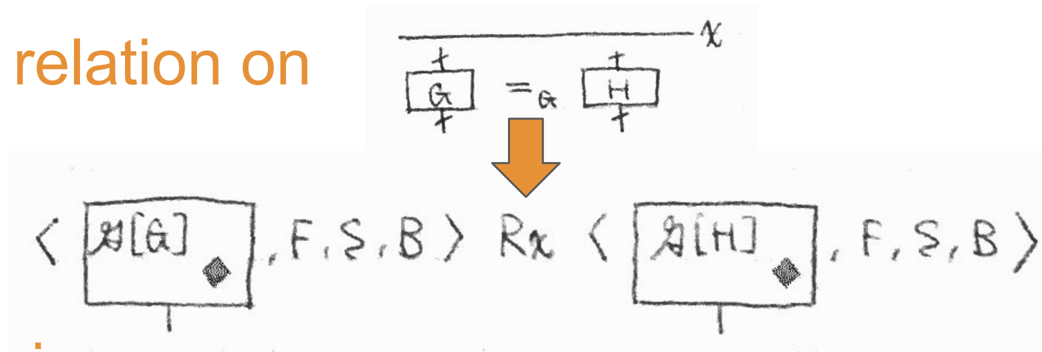
AND

(ii) σ_1 is final $\Leftrightarrow$ σ_2 is final

<u>Prop.</u> R_α is a U-simulation $\Longrightarrow$ $G \simeq_\alpha H$

call-by-value *graph*-equational theory → (soundness, graphically) → *graph*-contextual (operational) equivalence

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



2. show the binary relation is a "*U-simulation*"

R is a U-simulation $\overset{\Delta}{\Leftrightarrow}$ (1) if $\sigma_1 R \sigma_2$ and $\sigma_1 \to \sigma_1'$,

simulation

(i) $\exists \sigma_2'.\ \sigma_2 \to \sigma_2'$ and $\sigma_1' R^* \sigma_2'$

OR

(ii) $\exists \sigma_1''.\ \sigma_1 \to \sigma_1' \to^* \sigma_1''$ and $\sigma_1'' R^* \sigma_2$

(2) if $\sigma_1 R \sigma_2$ and $\sigma_1 \not\to$,

(i) $\sigma_2 \not\to$

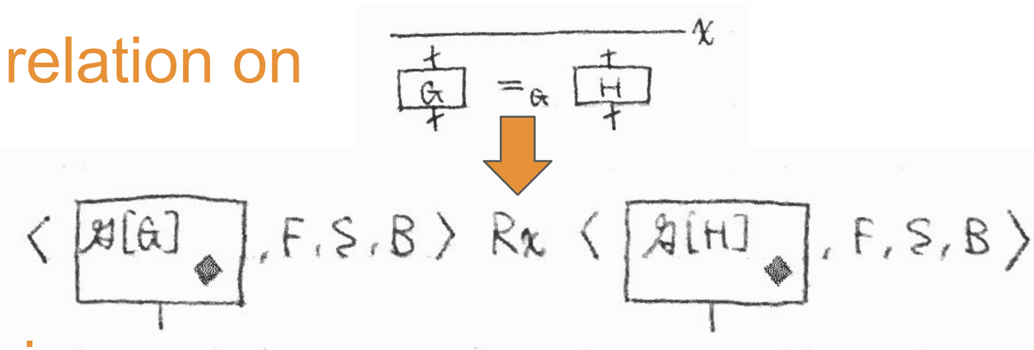
AND

(ii) σ_1 is final $\Leftrightarrow$ σ_2 is final

Prop. R_{α} is a U-simulation $\Rightarrow$ $G \simeq_{\alpha} H$

call-by-value *graph*-equational theory —soundness / graphically→ *graph*-contextual (operational) equivalence

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



2. show the binary relation is a "*U-simulation*"

R is a~~n~~ <u>U-simulation</u> $\overset{\Delta}{\Leftrightarrow}$ (1) if $\sigma_1 R \sigma_2$ and $\sigma_1 \to \sigma_1'$,

(i) $\exists \sigma_2'.\ \sigma_2 \to \sigma_2'$ and $\sigma_1' R^* \sigma_2'$

OR

(ii) $\exists \sigma_1''.\ \sigma_1 \to \sigma_1' \to^* \sigma_1''$ and $\sigma_1'' R^* \sigma_2$

simulation
...until the difference is reduced

(2) if $\sigma_1 R \sigma_2$ and $\sigma_1 \not\to$,

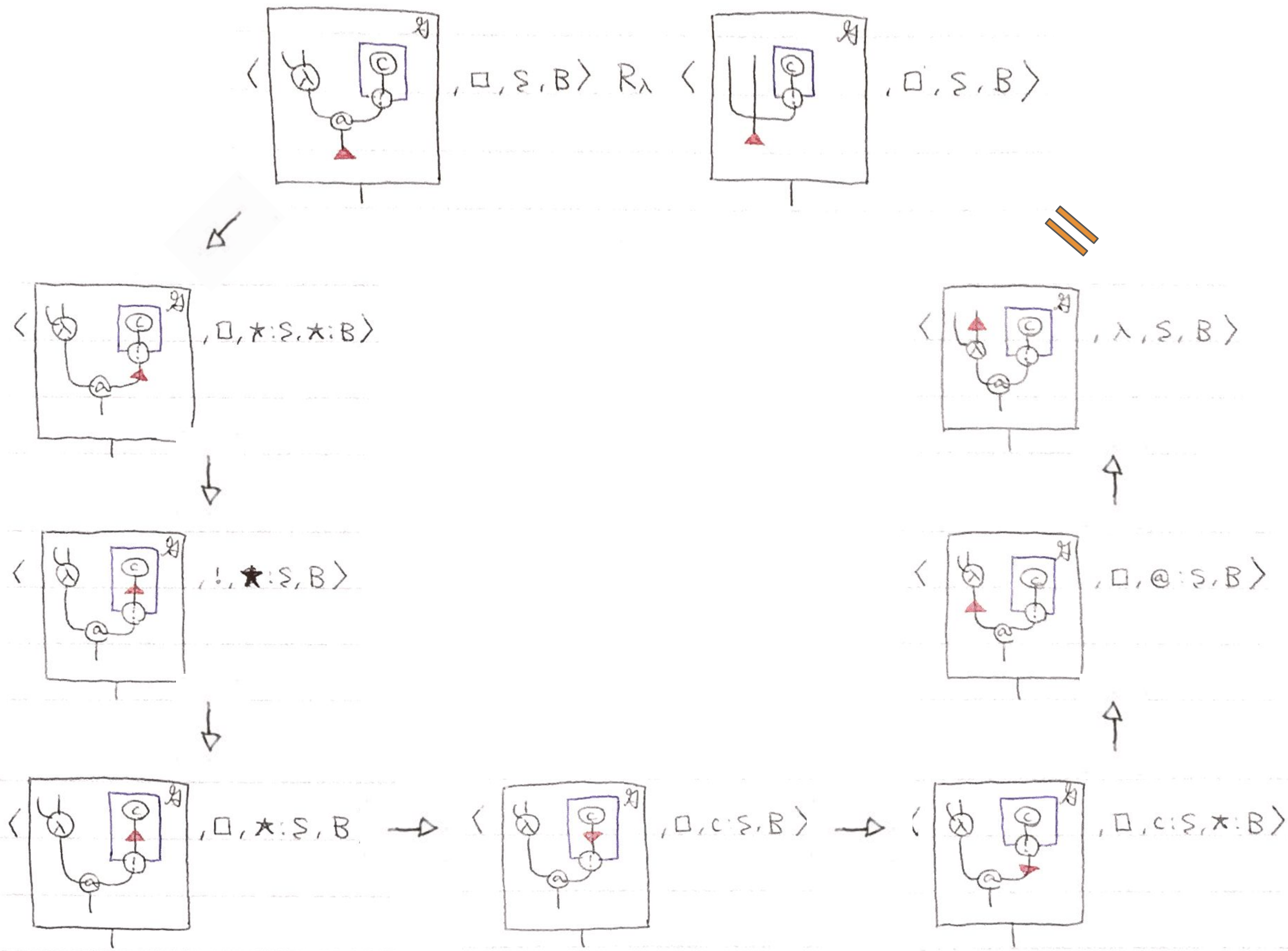
(i) $\sigma_2 \not\to$

AND

(ii) σ_1 is final $\Leftrightarrow$ σ_2 is final

<u>Prop.</u> R_α is a U-simulation $\Longrightarrow$ $G \cong_\alpha H$

“Until the difference is reduced”



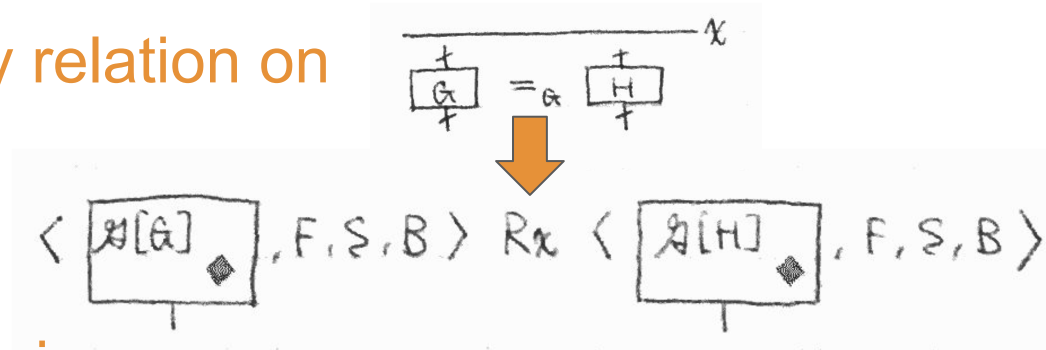
call-by-value *graph*-equational theory

soundness

graphically

graph-contextual (operational) equivalence

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



2. show the binary relation is a "*U-simulation*"

R is a $\cancel{\text{an}}$ <u>U-simulation</u> $\overset{\Delta}{\Leftrightarrow}$ (1) if $\sigma_1 \mathrel{R} \sigma_2$ and $\sigma_1 \to \sigma_1'$,

(i) $\exists \sigma_2'.\ \sigma_2 \to \sigma_2'$ and $\sigma_1' \mathrel{R^*} \sigma_2'$

OR

$\boxed{\text{(ii) } \exists \sigma_1''.\ \sigma_1 \to \sigma_1' \to^* \sigma_1'' \text{ and } \sigma_1'' \mathrel{R^*} \sigma_2}$

(2) if $\sigma_1 \mathrel{R} \sigma_2$ and $\sigma_1 \not\to$,

(i) $\sigma_2 \not\to$

AND

(ii) σ_1 is final $\Leftrightarrow$ σ_2 is final

simulation

...until the **difference** is reduced

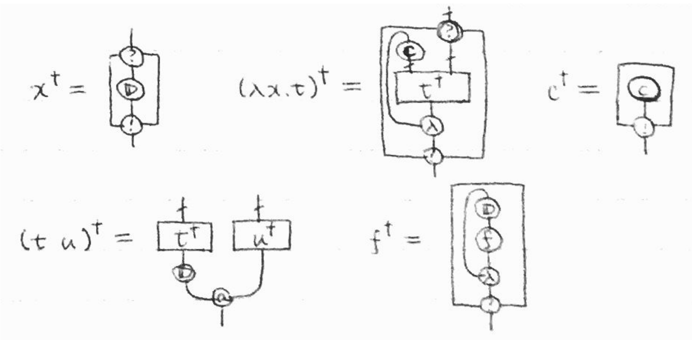
<u>Prop.</u> R_α is a U-simulation $\Longrightarrow$ $G \simeq_\alpha H$

call-by-value *graph*-equational theory

soundness

graphically

graph-contextual (operational) equivalence



alpha-law: trivial

beta-law: refined (cf. explicit substitution)

$$G =_{\mathfrak{G}} H \Longrightarrow G \cong_{\mathfrak{G}} H$$

modular proof using *U-simulations*

$G \cong_{\mathfrak{G}} H \Leftrightarrow \forall \mathcal{C}[\square]$ s.t. $\boxed{\mathcal{C}[G]}$ and $\boxed{\mathcal{C}[H]}$,

$\mathrm{Eval}_{\mathfrak{G}}(\mathcal{C}[G])$ is defined $\Leftrightarrow$ $\mathrm{Eval}_{\mathfrak{G}}(\mathcal{C}[H])$ is defined

Moreover, if defined, $\mathrm{Eval}_{\mathfrak{G}}(\mathcal{C}[G]) = \mathrm{Eval}_{\mathfrak{G}}(\mathcal{C}[H]$

dGoI machine

Equivalence of programs

denotational equality

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

syntactical equation

`t = u`

graphically?

operational equivalence

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

Equivalence of programs

syntactical equation

$t = u$

denotational equality

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

operational equivalence

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

graphically?

$$G =_{G} H \Longrightarrow G \cong_{G} H$$

modular proof of soundness using *U-simulations* so what?

Equivalence of programs

syntactical
equation

`t = u`

graphically?

operational
equivalence

$$G =_{\mathfrak{G}} H \implies G \simeq_{\mathfrak{G}} H$$

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

related proof techniques:
logical relations
applicative bisimulations
envirionmental bisimulations...

syntactical equation

`t = u`

graphically?

operational equivalence

$$G =_{G} H \Longrightarrow G \cong_{G} H$$

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

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

syntactical equation

`t = u`

graphically?

operational equivalence

$$G =_{\alpha} H \implies G \simeq_{\alpha} H$$

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

cost-sensitive equivalence?
(cf. [Schmidt-Schauss & Dallmeyer, WPTE '17]

syntactical equation

`t = u`

graphically?

operational equivalence

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

$$G =_{G} H \quad \Longrightarrow \quad G \cong_{G} H$$

modular proof of soundness using *U-simulations*