Sémantique opérationnelle

Pierre-Louis Curien

(gdt Modélisation et Vérification, LIAFA)

2 avril 2001

Introduction

On expose, dans le cadre de divers langages fonctionnels ou extensions de langages fonctionnels, diverses notions:

- de **convergence** – comment spécifier les règles d'évaluation;
- d'**équivalences observationnelles** de programmes dérivées de cette notion de convergence – on fera apparaître des liens avec la notion de *bisimulation*, et on donnera des illustrations dans un langage fonctionnel qui permet de programmer des calculs de suites infinies;
- de **complète adéquation** – quête de modèles dénotationnels induisant exactement l'équivalence observationnelle; on commentera quelques succès dans ce domaine;
- de **machine abstraite**, spécification plus concrète des règles d'évaluation – on fera apparaître un lien avec le *calcul des séquents*. De même que dans le cochon on mange tout, dans une machine abstraite on type tout.

$\boldsymbol{\lambda}$-calcul sans types

Sémantique *à grands pas* (V abstraction):

$$\frac{M \Downarrow \lambda x.P \quad P[x \leftarrow N] \Downarrow V}{MN \Downarrow V} \qquad \frac{}{\lambda x.M \Downarrow \lambda x.M}$$

Sémantique *à petits pas* ($P\vec{R}$ abréviation de $PR_1 \ldots R_n$):

$$(\lambda x.M)N\vec{N} \rightarrow M[x \leftarrow N]\vec{N}$$

On remarque qu'un terme clos M est en forme normale pour $\rightarrow$ ssi $M = \lambda x.N$. On pose $M \downarrow V$ ssi $M \rightarrow^{\star} V$.

Proposition. $M \Downarrow V \Leftrightarrow M \downarrow V$

Préordre observationnel ($\boldsymbol{\lambda}$-calcul sans types)

Trois notions:

- préordre **contextuel** (syntaxe des contextes: $M ::= [\,] \parallel x \parallel MN \parallel \lambda x.M$):

  $$M \preceq_{cont} N \quad \text{ssi} \quad \forall C \;\; C[M] \Downarrow \;\Rightarrow\; C[N] \Downarrow$$

- préordre **applicatif**:

  $$M \preceq_{app} N \quad \text{ssi} \quad \forall \vec{P} \;\; M\vec{P} \Downarrow \;\Rightarrow\; N\vec{P} \Downarrow$$

- **similarité** (Abramsky), définie par plus grand point fixe: $M \preceq_{sim} N$ ssi

$$M \Downarrow \lambda x.P \Rightarrow \begin{cases} N \Downarrow \lambda x.Q \text{ et} \\ \forall R \;\; P[x \leftarrow R] \preceq_{sim} Q[x \leftarrow R] \end{cases}$$

Co-induction

Théorème. On a pour tous M, N:

$$M \preceq_{cont} N \;\Leftrightarrow\; M \preceq_{app} N \;\Leftrightarrow\; M \preceq_{sim} N$$

PREUVE. $(\preceq_{cont} \subseteq \preceq_{app})$. Prendre $C = [\,]\vec{P}$.

$(\preceq_{app} \subseteq \preceq_{cont})$. Cette implication porte le nom de *Lemme du Contexte*.

$(\preceq_{app} = \preceq_{sim})$. La similarité est essentiellement une reformulation du préordre applicatif, mais une reformulation très importante, car elle ouvre au *raisonnement co-inductif*:

Si F est monotone, alors F a un plus grand point fixe, noté $\nu X.F(X)$, et

$$\text{(co-induction)} \quad S \subseteq F(S) \;\Rightarrow\; S \subseteq \nu X.F(X)$$

Le langage PCF

On passe à un cadre typé, On suppose qu'il y a des types de base, et des types de fonctions:

$$A ::= \mathrm{bool} \parallel \mathrm{int} \parallel A \to A$$

On ajoute au langage les constantes correspondantes 0, 1,..., true, false, et d'autres: successeur, prédécesseur, test à zéro, conditionnelle, définition récursive de fonction.

On désigne un type de base par le nom générique κ. On désigne les constantes de type κ par le nom générique v.

La convergence (à petits pas) est maintenant définie seulement sur les termes (clos) *de type de base*. On a maintenant qu'un terme clos M de type de base est en forme normale pour $\to$ ssi $M = v$. On pose $M \Downarrow v$ ssi $M \to^\star v$.

Préordre observationnel (PCF)

- Préordre contextuel: $M \preceq_{cont} N$ ssi

$$\forall C \text{ de base } \quad C[M] \Downarrow \Rightarrow C[N] \Downarrow$$

- Préordre applicatif: $M \preceq_{app} N$ ssi

$\forall \vec{P}$ tel que $M\vec{P}, N\vec{P} : \kappa \; (M\vec{P} \Downarrow \; \Rightarrow \; N\vec{P} \Downarrow)$

- Similarité: $M \preceq_{sim} N$

$$\begin{cases} \forall R \;\; MR \preceq_{sim} NR & (M, N : A \to B) \\ M \Downarrow v \Rightarrow N \Downarrow v & (M, N : \kappa) \end{cases}$$

Théorème. On a pour tous M, N:

$$M \preceq_{cont} N \;\; \Leftrightarrow \;\; M \preceq_{app} N \;\; \Leftrightarrow \;\; M \preceq_{sim} N$$

Remarque. On a gardé les mêmes noms pour simplifier, mais ce ne sont plus les mêmes notions d'équivalence. Ainsi, si Ω est un programme divergent, on a:

- $\lambda x.\Omega x \not\preceq_{sim} \Omega$ avec la définition d'Abramsky (car $\lambda x.\Omega x \Downarrow$ alors que Ω diverge);
- $\lambda x.\Omega x \preceq_{sim} \Omega$ avec la définition ci-dessus. Ces deux termes, une fois appliqué à R, ne peuvent être distingués.

Listes infinies paresseuses (Gordon)

Pour avoir de vrais exemples concrets d'équivalence de programmes, on ajoute les listes (potentiellement infinies), ou suites:

$$A ::= \mathrm{bool} \parallel \mathrm{int} \parallel \mathrm{list}(A) \parallel A \to A$$

La convergence est définie sur les termes (clos) de type de base *et de type liste*:

- Un terme clos M de type de base est en forme normale pour $\to$ ssi $M = v$. On pose $M \Downarrow v$ ssi $M \to^\star v$ (comme plus haut).
- Un terme clos M de type $\mathrm{list}(A)$ est en forme normale, ou plutôt *canonique*, pour $\to$ ssi $M = (M_1 :: M_2)$. On pose

  $$M \Downarrow (M_1 :: M_2) \quad \text{ssi} \quad M \to^\star (M_1 :: M_2)$$

  Noter que M_1, et surtout M_2 sont non évalués; et que l'on n'observe pas $M \to^\star \mathrm{nil}$ $\hookrightarrow$ *listes paresseuses infinies*, e.g.

  $$\mathrm{pair}\ n = (2n) :: (\mathrm{pair}\ (n+1))$$

Préordre observationnel (PCF + suites infinies)

- Préordre contextuel: $M \preceq_{cont} N$ ssi

$$\forall\, C \text{ de base } \ (C[M]\Downarrow \Rightarrow C[N]\Downarrow)$$

- Préordre applicatif: comme plus haut, on peut se contenter de contextes d'une forme particulière – moralement, ce sont $[\,]R$, $hd[\,]$ et $tl[\,]$; mais le langage de Gordon n'a pas hd et tl, mais une opération scase telle que

$$\text{scase}(a :: l, b, c) \to cab$$

qui permet de récupérer des variantes *totales* de hd et tl en prenant respectivement $c = \lambda xy.x$ et $c = \lambda xy.y$ – détails omis.

- Similarité: $M \preceq_{sim} N$ ssi

$$\begin{cases} \forall\, R \ \ MR \preceq_{sim} NR & (M, N : A \to B) \\ M \Downarrow (M_1 :: M_2) \Rightarrow \begin{cases} N \Downarrow (N_1 :: N_2) \\ M_1 \preceq_{sim} N_1 \text{ et} \\ M_2 \preceq_{sim} N_2 \end{cases} & (M, N : \text{list}(A)) \\ M \Downarrow v \Rightarrow N \Downarrow v & (M, N : \kappa) \end{cases}$$

Un système de transition étiqueté

On pose:

$$\begin{array}{lll} M \overset{[\,]R}{\longmapsto} MR & (M : A \to B) \\ M \overset{hd}{\longmapsto} M_1 & (M : \text{list}(A), M \Downarrow (M_1 :: M_2) \\ M \overset{tl}{\longmapsto} M_2 & (M : \text{list}(A), M \Downarrow (M_1 :: M_2) \\ M \overset{v}{\longmapsto} \mathbf{0} & (M : \kappa, M \Downarrow v) \end{array}$$

Proposition. La relation $\preceq_{sim}$ est le plus grand point fixe de: $M \preceq_{sim} N$ ssi

$$\forall\, a \;\; (M \overset{a}{\longmapsto} M' \;\; \Rightarrow \;\; N \overset{a}{\longmapsto} N' \text{ et } M' \preceq_{sim} N')$$

Preuve. C'est une simple reformulation de la définition de $\preceq_{sim}$. CQFD

Exemples d'équivalence observationnelle (preuves par co-induction)

Soit

$$\begin{array}{l}\text{map } f\ (a_1 :: a_2 :: \ldots :: a_n :: \ldots) = \\ \qquad (f a_1) :: (f a_2) :: \ldots :: (f a_n) :: \ldots\end{array}$$

$$\begin{array}{l}\text{iterate } f\ (a_1 :: a_2 :: \ldots :: a_n :: \ldots) = \\ \qquad a_1 :: (f a_2) :: \ldots :: (f^{n-1} a_n) :: \ldots\end{array}$$

$$\begin{array}{l}\text{filter } f\ (a_1 :: \ldots :: a_n :: \ldots) = \\ \qquad a_{i_1} :: \ldots :: a_{i_n} :: \ldots \\ \qquad\quad (\{i_1, \ldots, i_n, \ldots\} = \{i \mid f(i) = \text{true}\})\end{array}$$

On pose $(=_{sim}) \ = \ (\preceq_{sim} \ \cap \geq_{sim})$. On a:

$$\begin{array}{lll}(\text{iterate } f\ (f x)) & =_{sim} & (\text{map } f\ (\text{iterate } f\ x)) \\ (\text{filter } f) \circ (\text{map } g) & =_{sim} & (\text{map } g) \circ (\text{filter } (f \circ g))\end{array}$$

Computational Adequacy

On a vu plusieurs langages, et pour chacun une notion de convergence $\Downarrow$. On souhaite qualifier une *sémantique dénotationnelle* par rapport à cette notion de convergence.

$$(CA) \quad M \Downarrow \quad \Leftrightarrow \quad [\![M]\!] \neq \bot$$

Cette propriété est vérifiée pour la plupart des notions de convergence et de modèle:

- L'implication $\Rightarrow$ est la conséquence de la validité des règles de la sémantique opérationnelle dans le modèle, et de ce que si $M \Downarrow$, alors $[\![M]\!] \neq \bot$.
- L'implication $\Leftarrow$ est difficile, et se montre par une technique très répandue pour le λ-calcul, la *réalisabilité*.

Adéquation

Pour un modèle satisfaisant (CA), on souhaite comparer l'(in)égalité induite par la sémantique opérationnelle et celle induite par la sémantique dénotationnelle:

$$M \preceq_{cont} N \quad \text{versus} \quad [\![M]\!] \leq [\![N]\!]$$

L'implication

$$M \preceq_{cont} N \quad \Leftarrow \quad [\![M]\!] \leq [\![N]\!]$$

est appelée **adéquation** et est conséquence de (CA).

Preuve. (CA) permet de reformuler $\preceq_{cont}$ en termes de sémantique dénotationnelle seulement: $M \preceq_{cont} N$ ssi

$$\forall\, C \;\; ([\![C[M]]\!] \neq \bot \Rightarrow [\![C[N]]\!] \neq \bot)$$

La compositionnalité de la sémantique dénotationnelle garantit que

$$[\![M]\!] \leq [\![N]\!] \Rightarrow [\![C[M]]\!] \leq [\![C[N]]\!]$$

A fortiori, si $[\![C[M]]\!] \neq \bot$, alors $[\![C[N]]\!] \neq \bot$.

Adéquation complète

L'autre implication

$$M \preceq_{cont} N \quad \Rightarrow \quad [\![M]\!] \leq [\![N]\!]$$

est difficile. Lorsqu'elle est vérifée, on dit que le modèle est *complètement adéquat*.

Une remarque. En logique, la complétude part du modèle et va vers la syntaxe: si A est valide, alors A est prouvable. Si la terminologie a un sens, on est donc conduit à penser à la sémantique opérationnelle comme au modèle et à la sémantique dénotationelle comme à la syntaxe. Et en effet, les modèles peuvent être présentés syntaxiquement:

$$[\![M]\!] \;=\; \begin{array}{l}\text{l'ensemble des types que } M \text{ peut}\\ \text{recevoir dans un système de types}\end{array}$$

C'est la théorie des *types avec intersections*. La "vraie" sémantique est donc celle induite par l'équivalence contextuelle.

Discussion

Ce qui est vraiment difficile/intéressant, ce n'est pas simplement de construire un modèle complètement adéquat. On peut le faire par force brute: prendre un modèle de termes, le quotienter par l'équivalence observationnelle. C'est d'ailleurs souvent par ce genre de force brute et souvent triviale que l'on prouve des théorèmes de complétude en logique.

C'est de construire ce modèle *par d'autres moyens*. C'est un peu vague. Pour préciser, on peut vouloir que cette construction offre des moyens nouveaux de prouver que deux termes sont équivalents ou non-équivalents. Il faut donc que l'on puisse faire des preuves dans le modèle dénotationnel. Ce n'est clairement pas le cas si le modèle est obtenu par "collapse extensionnel", qui est la version mathématique de $\preceq_{app}$.

Complète adéquation: le cas de PCF

Résultat négatif de Loader: *l'équivalence observationnelle n'est pas décidable sur les types finis* (construits à partir de bool et $\rightarrow$). Un quotient "aveugle" est donc inévitable.

Néanmoins, en 1994 (Abramsky-Jagadeesan-Malacaria, Hyland-Ong), les sémantiques de jeux ont permis de présenter le modèle de termes (avant quotient observationnel) d'une manière indépendante de la syntaxe: bijection (Hyland-Ong) entre

arbres de Böhm pour PCF et

$$\text{stratégies} \begin{cases} \textbf{innocentes} \text{ et} \\ \textbf{bien parenthésées} \end{cases}$$

(cf. exposé Danos). Si on y tient absolument, on peut dire que le modèle complètement adéquat est le quotient observationnel du modèle de jeux, mais ce qui est plus intéressant, c'est cette caractérisation des termes (arbres infinis, plus exactement) en forme normale $\hookrightarrow$ programme de classification.

Complète adéquation: IA

Idealized Algol: On ajoute comme types de base le type des commandes et celui des variables. Le langage comprend le séquencement de commandes, l'affectation $x ::= e$, la lecture deref x, la déclaration de variables locales.

La relaxation de l'innocence (en gardant le bon parenthésage) conduit à un modèle complètement adéquat de IA, *sans quotient* (Abramsky et McCusker). L'idée est de modéliser une variable locale par des coups des joueurs (lecture, écriture). La discipline de lecture de la dernière valeur écrite force bien la non-innocence: la valeur lue dépend crucialement de l'histoire.

Théorème [Abramsky-McCusker (1996)]. Le modèle des stratégies bien parenthésées est complètement adéquat pour IA.

PREUVE. Voir SPCF. Notons que la stratégie a dans cette preuve n'est pas en général innocente (même si $c_1\, v_1\, \ldots\, c_n\, v_n$ est une vue pour Joueur, ce n'en est pas une pour Opposant, qui est Joueur dans $A \to \kappa$). CQFD

Complète adéquation: SPCF (1/3)

La relaxation de l'autre condition (bon parenthésage) (en gardant l'innocence) permet de modéliser des opérateurs de contrôle, tels que *catch* et *throw* (Laird). Mais cela ne conduit pas directement à un modèle complètement adéquat: l'épouvantail du quotient observationnel est toujours là.

Mais une autre variante (non formulée en termes de jeux, mais déjà clairement en termes d'**interaction**), donne l'adéquation complète *sans quotient*: **algorithmes séquentiels** de Berry-Curien (1978).

La différence essentielle avec les sémantiques de jeux "anglaises" de 1994, c'est le traitement de l'exponentielle $!A$: les algorithmes séquentiels ne répètent pas les questions, et $[\![\mathrm{bool} \to \mathrm{bool}]\!]$ est de cardinal fini. (Notons que la répétition des questions est utilisée de façon essentielle pour modéliser le comportement d'une variable impérative.)

Complète adéquation: SPCF (2/3)

Le langage SPCF est PCF auquel on ajoute catch avec les règles (simplifiées ici):

$$\begin{array}{l}\mathrm{catch}(\lambda x_1 \ldots x_n . x_i M_1 \ldots M_m) = i - 1 \\ \mathrm{catch}(\lambda x_1 \ldots x_n . m) = n + m\end{array}$$

Théorème [Cartwright-Curien-Felleisen (1994)]. Le modèle des algorithmes séquentiels est complètement adéquat pour SPCF.

PREUVE. (1) Tout algorithme fini a est définissable, i.e. il existe M tel que $[\![M]\!] = a$.

Complète adéquation: SPCF (3/3)

(2) Si b et c sont deux algorithmes de type A tels que $b \not\leq c$, alors on peut trouver un algorithme de type $A \to \kappa$ tel que

$$a \cdot b = v \quad \text{et} \quad a \cdot c \neq v$$

($a \cdot b$ est l'application, définie par interaction entre a (contre-stratégie) et b (stratégie)). Si $c_1 v_1 \ldots c_n v_n$ est une position qui est dans b et non dans c, on peut définir a comme suit:

$$\begin{array}{lll} A & \to & \kappa \\ & & ? \\ c_1 & & \\ v_1 & & \\ \vdots & & \\ c_n & & \\ v_n & & \\ & & v \end{array}$$

Si $[\![M]\!] \not\leq [\![N]\!]$, alors par (2) il existe a tel que $a.[\![M]\!] = v$ et $a \cdot [\![N]\!] \neq v$. Par (1), il existe P tel que $a = [\![P]\!]$. On a donc $[\![PM]\!] = v$ et $[\![PN]\!] \neq v$. On conclut que $M \not\leq_{cont} N$, en prenant $C = P[]$. CQFD

Machines abstraites et séquents (1/8)

On revient au λ-calcul (typé). On veut spécifier une sémantique opérationnelle qui gère explicitement les appels récursifs implicites dans la spécification à grands pas de $\Downarrow$. La solution est connue, il faut une **pile**:

$$\begin{array}{lll} (NL) & \langle MN|S\rangle & \rightarrow \langle M|N :: S\rangle \\ (\rightarrow) & \langle \lambda x.M|N :: S\rangle & \rightarrow \langle M[x \leftarrow N]|S\rangle \end{array}$$

Les termes ont un type. On va aussi donner un type aux piles, et aux états $\langle M|S\rangle$. Dans un tel état, la pile représente un *contexte* (ou une *continuation*) qui attend le résultat de l'evaluation de M. D'où l'idée de jugements de la forme:

$$\Gamma \,|\, S : A \vdash \Delta$$

Ce qui importe, c'est le type du résultat que S attend *en entrée* pour continuer le calcul.

Machines abstraites et séquents (2/8)

Pour plus de symétrie, au lieu des jugements de typage habituels $\Gamma \vdash M : A$ pour les termes, on adopte

$$\Gamma \vdash M : A \,|\, \Delta$$

En effet, c'est ce typage qui convient pour des extensions du λ-calcul avec des opérateurs de contrôle (cf. plus haut). Logiquement, le passage d'une formule à droite de $\vdash$ à plusieurs formules à droite de $\vdash$ correspond à la levée de la restriction au raisonnement intuitionniste.

Machines abstraites et séquents (3/8)

Il reste à typer un état (*règle de coupure*):

$$\frac{\Gamma \vdash M : A \,|\, \Delta \qquad \Gamma \,|\, S : A \vdash \Delta}{\langle M|S\rangle : (\Gamma \vdash \Delta)}$$

La coupure, c'est la notion de *lemme*. Le calcul des séquents a été conçu autour de cette notion. Une preuve sans coupure est un formidable outil, qui sert notamment à prouver la cohérence logique d'un système formel. On montre d'abord que toute formule prouvable l'est par une preuve sans coupure. Les preuves sans coupure ont de remarquables propriétés, en particulier celle de la *sous-formule*: toute formule apparaissant dans la preuve doit être dans la conclusion. Aucun moyen alors de prouver un atome tel que $0 = 1$. D'où l'importance en logique de l'*élimination des coupures*.

Machines abstraites et séquents (4/8)

Typons $N :: S$ (empilement). Puisque $\langle MN|S\rangle$ est un état, on doit avoir:

$$\Gamma \vdash M : A \to B \,|\, \Delta \quad \Gamma \vdash N : A \,|\, \Delta \quad \Gamma \,|\, S : B \vdash \Delta$$

De même, puisque $\langle M|N :: S\rangle$ est un état, on doit avoir:

$$\Gamma \,|\, (N :: S) : A \to B \vdash \Delta$$

On vient de synthétiser la règle suivante:

$$\frac{\Gamma \vdash N : A \,|\, \Delta \qquad \Gamma \,|\, S : B \vdash \Delta}{\Gamma \,|\, (N :: S) : A \to B \vdash \Delta}$$

C'est la règle d'*introduction à gauche* de l'implication en calcul des séquents:

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta}$$

Machines abstraites et séquents (5/8)

- Le calul des séquents n'a que des règles d'introduction, à droite et à gauche, pour les connecteurs logiques.
- La déduction naturelle, qui correspond exactement au λ-calcul typé, a des règles d'introduction (à droite) et d'élimination (à droite).

On peut donc voir la règle (NL) comme une règle de *transfert* de la déduction naturelle vers le calcul des séquents.

Machines abstraites et séquents (6/8)

Règle ($\rightarrow$). Logiquement, en oubliant les termes, elle transforme une preuve se terminant ainsi (où l'on retrouve en haut, de gauche à droite, M, N et S, respectivement):

$$\dfrac{\dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \rightarrow B, \Delta} \qquad \dfrac{\Gamma \vdash A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \rightarrow B \vdash \Delta}}{\Gamma \vdash \Delta}$$

en une preuve se terminant ainsi (les feuilles correspondant toujours à M, N et S, de gauche à droite):

$$\dfrac{\dfrac{\Gamma, A \vdash B, \Delta \quad \Gamma \vdash A, \Delta}{\Gamma \vdash B, \Delta} \qquad \Gamma, B \vdash \Delta}{\Gamma \vdash \Delta}$$

On a transformé une coupure sur une formule complexe ($A \rightarrow B$) en deux coupures sur ses sous-formules immédiates. Cela remonte au grand logicien Gentzen.

Machines abstraites et séquents (7/8)

Deux remarques:

- Si je fais un zoom sur

$$\frac{\overbrace{\Gamma, A \vdash B, \Delta}^{M} \quad \overbrace{\Gamma \vdash A, \Delta}^{[x \leftarrow N]}}{\Gamma \vdash B, \Delta}$$

  et si je relis cette inférence en termes de syntaxe, ce n'est pas la substitution effective de N dans M à chaque occurrence de x qui est décrite ici, mais une opération nouvelle, dite de **substitution explicite**. En effet, dans une implantation, on ne va pas effectuer ces substitutions partout $\hookrightarrow$ tables pour les valeurs des variables.

- L'ordre entre ces deux coupures est un détail sans importance, on peut aussi bien faire une "multi-coupure":

$$\frac{\overbrace{\Gamma, A \vdash B, \Delta}^{M} \quad \overbrace{\Gamma \vdash A, \Delta}^{[x \leftarrow N]} \quad \overbrace{\Gamma, B \vdash \Delta}^{S}}{\Gamma \vdash \Delta}$$

Machines abstraites et séquents (8/8)

Ces deux remarques suggèrent le passage à une machine abstraite à trois composantes, avec des états de la forme

$$\langle M[\rho] | S \rangle$$

M est le **code**
ρ l'**environnement**
S la **pile**

Un couple $M[\rho]$ est appelé une *fermeture* dans la communauté des langages fonctionnels.

Et on arrive à la **machine de Krivine** (1985):

$$\begin{array}{lcl}
\langle\, (MN)[\rho] \mid S \,\rangle & \to & \langle\, M[\rho] \mid N[\rho] :: S \,\rangle \\
\langle\, (\lambda x.M)[\rho] \mid N[\rho'] :: S \,\rangle & \to & \langle\, M[\rho[x \leftarrow N[\rho']]] \mid S \,\rangle \\
\langle\, x[\rho] \mid S \,\rangle & \to & \langle\, \rho(x) \mid S \,\rangle
\end{array}$$