

Examen du DEA d'informatique

Sémantique et programmation

Typage et Programmation

10 décembre 1997

Durée 2h30

Exceptions et types concrets

Les parties 1, 2 et 3 sont indépendantes. La partie 4 ne dépend que de la partie 1. On pourra admettre les résultats des parties 1, 2, 3 et 4 dans la partie 5. Il y a des questions faciles dans chacune des parties, et la plupart des réponses sont courtes. Les questions plus longues ou plus difficiles sont marquées par *.

1 Exceptions On considère un langage L qui étend le noyau ML avec une seule sorte d'exception.

$$a ::= x \mid a \mid v \mid \text{let } x = v \text{ in } a \mid \text{raise } a \mid \text{try } a \text{ with } v \quad v ::= \lambda x. a$$

Les contextes d'évaluations sont définis par :

$$\begin{array}{ll} E ::= \{ \} \mid E \ a \mid v \ E \mid \text{raise } E \mid \text{try } E \text{ with } v & \text{Contextes d'évaluation} \\ E' ::= \{ \} \mid E' \ a \mid v \ E' \mid \text{raise } E' & \text{Contextes applicatifs} \end{array}$$

La réduction étend celle de ML avec les règles suivantes

$$\text{try } v_0 \text{ with } v \longrightarrow v_0 \text{ (SUCCESS)} \quad \text{try } E' \{ \text{raise } v_0 \} \text{ with } v \longrightarrow v \ v_0 \text{ (JUMP)}$$

1.1 Montrer sur un exemple que remplacer E' par E ne donnerait pas la sémantique attendue.

On suppose donné un type τ_0 sans variable. On donne les deux nouvelles règles de typage :

$$\frac{A \vdash a : \tau_0}{A \vdash \text{raise } a : \tau} \text{ (RAISE)} \quad \frac{A \vdash a : \tau \quad A \vdash v : \tau_0 \rightarrow \tau}{A \vdash \text{try } a \text{ with } v : \tau} \text{ (TRY)}$$

1.2 Donner un exemple de programme bien typé qui n'est pas une valeur.

1.3 Quelle est la forme des programmes bien typés irréductibles.

1.4 On note $\xrightarrow{X}$ la relation $\longrightarrow$ étendue avec les règles suivantes :

$$\begin{array}{ll} \text{raise } (\text{raise } v) \longrightarrow \text{raise } v & \text{(RAISE)} \\ (\text{raise } v) \ a \longrightarrow \text{raise } v & \text{(LEFT)} \\ v \ (\text{raise } v') \longrightarrow \text{raise } v' & \text{(RIGHT)} \end{array}$$

Vérifier que si $E'_1 \{ \text{raise } v \} \xrightarrow{X} a$, alors a est de la forme $E'_2 \{ \text{raise } v \}$.

1.5 En déduire que si $a \xrightarrow{X}_* v$ alors $a \longrightarrow_* v$.

***1.6** En présence des règles RAISE, LEFT, RIGHT, peut-on remplacer la règle JUMP par la règle plus faible

$$\text{try raise } v_0 \text{ with } v \longrightarrow v v_0 \quad (\text{FAILURE})$$

sans changer l'ensemble des résultats possibles (justifier) ?

Dans la suite on prendra pour sémantique des exceptions, les règles RAISE, LEFT, RIGHT et FAILURE au lieu de la règle JUMP,

2 Types concrets On considère maintenant un autre langage L' qui étend le noyau ML avec un type concret à deux constructeurs qui pourrait être défini en ML par la déclaration $\text{type } \langle \alpha_1 \mid \alpha_2 \rangle = \text{Val of } \alpha_1 \mid \text{Exc of } \alpha_2$. Les expressions de L' sont

$$\begin{aligned} a ::= & x \mid v \mid a a \mid \text{let } x = v \text{ in } a \mid \text{Val } a \mid \text{Exc } a \mid \text{nil} \\ & \mid \text{match } a \text{ with Val } x \rightarrow a \text{ else } v \mid \text{match } a \text{ with Exc } x \rightarrow a \text{ else } v \\ v ::= & \lambda x. a \mid \text{Val } v \mid \text{Exc } v \mid \text{nil} \end{aligned}$$

Il y a 4 nouvelles règles de réductions sont

$$\begin{aligned} \text{match Val } v_0 \text{ with Val } x \rightarrow a \text{ else } v &\longrightarrow a\{v_0/x\} & (\text{VAL-VAL}) \\ \text{match Exc } v_0 \text{ with Val } x \rightarrow a \text{ else } v &\longrightarrow v (\text{Exc } v_0) & (\text{EXC-VAL}) \end{aligned}$$

et les 2 règles symétriques EXC-EXC et VAL-EXC.

2.1 Décrire l'ensemble des contextes d'évaluation.

La constante `nil` ne peut jamais être réduite. Elle est utilisée pour “fermer” la construction `match`. L'expression ML `match  $a_0$  with Val  $x_1 \rightarrow a_1 \mid \text{Exc } x_2 \rightarrow a_2$` peut se voir comme une abréviation pour

$$\text{match } a_0 \text{ with Val } x_1 \rightarrow a_1 \text{ else } \lambda z. (\text{match } z \text{ with Exc } x_2 \rightarrow a_2 \text{ else nil})$$

On prendra pour expressions de types $\tau ::= \alpha \mid \tau \rightarrow \tau \mid \langle \tau \mid \tau \rangle$ et on donne les règles de typage suivantes :

$$\frac{A \vdash a : \tau_1}{A \vdash \text{Val } a : \langle \tau_1 \mid \tau_2 \rangle} (\text{VAL}) \qquad \frac{A \vdash a : \tau_2}{A \vdash \text{Exc } a : \langle \tau_1 \mid \tau_2 \rangle} (\text{EXC})$$

2.2 Donner la règle de typage de la construction `match  $a_0$  with Val  $x_1 \rightarrow a_1 \mid \text{Exc } x_2 \rightarrow a_2$` .

2.3 À quelle expression ML correspond l'expression `match  $a_0$  with Val  $x \rightarrow a$  else nil` ? Comment cette expression est-elle typée en ML. Quel problème pose son évaluation ? En déduire que l'on ne peut pas simultanément typer `match  $a_0$  with Val  $x \rightarrow a$  else  $v$` et `nil` tout en préservant la correction de l'évaluation.

3 Typage avec des variantes polymorphes On reprend le typage mais en considérant le type concret comme une variante polymorphe. :

$$\tau ::= \alpha \mid \tau \rightarrow \tau \mid \langle \rho \mid \rho \rangle \qquad \rho ::= \alpha \mid \text{Pre } \tau \mid \text{Abs}$$

3.1 Proposer de nouvelles règles de typage pour les expressions **Val** a , **Exc** a et **match** a with **Val** $x \rightarrow a$ **else** v .

3.2 On donne le type $\forall \alpha. \langle \text{Abs} \mid \text{Abs} \rangle \rightarrow \alpha$ à **nil**. Que faut-il vérifier pour que la correction soit préservée? (énoncer la propriété sans la démontrer)

3.3 Vérifier que l'expression a égale à **match** $(\lambda x. \text{Val } x) (\lambda y. y)$ with **Val** $z \rightarrow z$ **else** **nil** est bien typée. En admettant la correction du typage vis à vis de l'évaluation, que peut-on en déduire sur le programme $(\lambda x. \text{Val } x) (\lambda y. y)$?

3.4 Montrer que si v est de type $\langle \tau \mid \text{Abs} \rangle$ alors v n'est pas de la forme **Exc** v . Retrouver le résultat précédent sur a uniquement à partir du typage de a . Généraliser.

4 Simulation des exceptions avec des types concrets On définit la traduction suivante des expressions de L en expressions de L' (typo $\underline{x}$ corrigée):

$$\begin{aligned} \llbracket \text{let } x = v \text{ in } a \rrbracket &= \text{match } \llbracket v \rrbracket \text{ with Val } y \rightarrow \text{let } x = y \text{ in } \llbracket a \rrbracket \text{ else } \lambda z. z \\ \llbracket a_1 a_2 \rrbracket &= \\ &\quad \text{match } \llbracket a_1 \rrbracket \text{ with} \\ &\quad \quad \text{Val } x \rightarrow (\text{match } \llbracket a_2 \rrbracket \text{ with Val } y \rightarrow x y \text{ else } \lambda z. z) \\ &\quad \text{else } \lambda z. z \\ \llbracket \text{try } a_1 \text{ with } v \rrbracket &= \text{match } \llbracket a_1 \rrbracket \text{ with Exc } x \rightarrow \llbracket v \rrbracket \underline{x} \text{ else } \lambda z. z \\ \llbracket \text{raise } a \rrbracket &= \text{match } \llbracket a \rrbracket \text{ with Val } x \rightarrow \text{Exc } x \text{ else } \lambda z. z \\ \llbracket \lambda x. a \rrbracket &= \text{Val } (\lambda x. \llbracket a \rrbracket) & \llbracket x \rrbracket &= \text{Val } x \end{aligned}$$

4.1 Montrer que si $\llbracket a_1 \rrbracket \longrightarrow \text{Val } v_1$ et $\llbracket a_2 \rrbracket \longrightarrow \text{Val } v_2$ alors $\llbracket a_1 a_2 \rrbracket \longrightarrow v_1 v_2$

On considère la relation de réduction formée par l'application de la règle VAL-VAL dans un contexte d'évaluation E quelconque. Remarquer que cette réduction termine. On note a^V la forme normale de a par $\xrightarrow{V}$.

On étend la traduction aux contextes en ajoutant $\llbracket \{\} \rrbracket = \{\}$. On remarquera que $\llbracket E \rrbracket^V$ est un contexte d'évaluation ($\llbracket E \rrbracket$ n'en est pas nécessairement un). D'autre part $\llbracket E \{a\} \rrbracket^V$ est égal à $\llbracket E \rrbracket^V \{ \llbracket a \rrbracket^V \}$.

***4.2** On admet que si $a_1 \longrightarrow a_2$ est un β -rédex dans L , alors $\llbracket a_1 \rrbracket^V \longrightarrow \llbracket a_2 \rrbracket^V$ dans L' . Vérifier la propriété pour les autres rédex de L (i.e. RAISE, LEFT, RIGHT et FAILURE).

***4.3** En déduire que si $a_1 \longrightarrow a_2$ alors $\llbracket a_1 \rrbracket^V \longrightarrow \llbracket a_2 \rrbracket^V$.

5 Typage des exceptions

5.1 On considère le programme a égal à **try raise** v_0 **with** $\lambda z. z$. En supposant que v_0 est une valeur de type τ_0 , quel est le type de $\llbracket a \rrbracket$ (on ne demande pas la dérivation) ? En déduire que le programme a ne se bloque pas sur une exception.

Plus généralement, quelle condition sur le type de $\llbracket a \rrbracket$ pour que a ne se bloque pas sur une exception ?

5.2 Vérifier que la fonction $\lambda x. \text{raise } x$ ne soulève pas d'exception. Quel est son type ?

Vérifier que son utilisation peut soulever une exception. Comment peut-on lire cette information sur son type ?

5.3 Montrer que si l'on retire la primitive `nil`, on peut se restreindre à des types de la forme $\langle \text{Pre } \tau_1 \mid \tau_2 \rangle$ (i.e. ré-écrire les nouvelles règles de typage VAL, EXC et MATCH)

On voudrait maintenant un système de typage primitif sur le langage L permettant de détecter les exceptions qui peuvent s'échapper. On se propose d'écrire des jugements de la forme $A \vdash a : \tau, \rho$ où τ est le type ordinaire, et ρ indique si une exception peut être soulevée ou pas. On écrira $\tau \xrightarrow{\rho} \tau$ au lieu de $\tau \rightarrow \langle \text{Pre } \tau \mid \rho \rangle$.

5.4 En maintenant l'équivalence entre $A \vdash a : \tau, \rho$ et $A \vdash \llbracket a \rrbracket : \langle \text{Pre } \tau \mid \rho \rangle$, proposer de nouvelles règles de typage pour le jugement $A \vdash a : \tau, \rho$ dans L (on omettra la règle Let).

5.5 Comment traiter le cas des exceptions multiples ?

Corrigé

1.1 On pourrait alors “sauter” au dessus d’une construction “try”. On aurait par exemple

$$\text{try raise } v_1 \text{ with } v_2 \longrightarrow \text{raise } v_1$$

puisque $\text{try } \{\} \text{ with } v_2$ serait un contexte d’évaluation correct.

1.2 $\text{raise } v$ (comme l’énoncé ne précisait pas explicitement irréductible, $(\lambda x.x)(\lambda x.x)$ est aussi une réponse correcte).

1.3 Les programmes bien typés irréductibles sont des valeurs ou des exceptions non rattrapées, i.e. de la forme $E'\{\text{raise } v\}$.

1.4 Considérons d’abord les réductions en tête. Il ne peut pas s’agir d’une règle β car $\text{raise } v$ n’est pas une valeur. Il ne peut pas s’agir d’une règle JUMP car E'_1 ne peut pas être de la forme $\text{try } E_2\{v_0\} \text{ with } v$. Chacune des règles RAISE, LEFT et RIGHT s’applique, et le réduit est toujours $\text{raise } v$. Il suffit donc de prendre $\{\}$ pour E'_2 .

Sinon, il s’agit d’une réduction de la forme $E'_2\{a_1\} \longrightarrow E'_2\{a_2\}$ où $a_1 \longrightarrow a_2$ est une réduction en tête. Comme nous l’avons vu ci-dessus a_1 se réduit forcément sur $\text{raise } v$. Donc $E'_2\{a_1\}$, i.e. $E'_1\{\text{raise } v\}$ se réduit sur $E'_2\{\text{raise } v\}$.

1.5 Un nouveau redex $a_1 \xrightarrow{X} a_2$ ne peut s’opérer que dans un contexte de la forme $E\{\text{try } E'_1\{\text{raise } v_0\} \text{ with } v\}$ et être suivi de réductions de la forme suivante

$$E\{\text{try } E'_1\{\text{raise } v_0\} \text{ with } v\} \xrightarrow{X} E\{\text{try } E'_2\{\text{raise } v_0\} \text{ with } v\} \xrightarrow{X} \dots$$

$$E\{\text{try } E'_n\{\text{raise } v_0\} \text{ with } v\} \longrightarrow E\{v \ v_0\}$$

On aurait donc pu directement dériver

$$E\{\text{try } E'_1\{\text{raise } v_0\} \text{ with } v\} \longrightarrow E\{v \ v_0\}$$

1.6 Considérons une application de la règle

$$\text{try } E'_1\{\text{raise } v\} \text{ with } v \longrightarrow v \ v_0 \quad (\text{JUMP})$$

Privons nous de la règle JUMP. La suite de réductions “interne”

$$E'_1\{\text{raise } v_0\} \longrightarrow E'_2\{\text{raise } v_0\} \longrightarrow \dots E'_n\{\text{raise } v_n\} \dots$$

termine, car la taille de E'_{k+1} est toujours inférieure à celle de E'_k . Or, la seule façon de terminer est que le contexte E' soit vide. On a donc $E'\{\text{raise } v\} \longrightarrow_* \text{raise } v$ et on peut en déduire

$$\text{try } E'_1\{\text{raise } v\} \text{ with } v \longrightarrow v \ v_0 \quad (\text{FAILURE})$$

2.1

$$E ::= \{\} \mid E \ a \mid v \ E \mid \text{Val } E \mid \text{Exc } E \mid \text{match } E \text{ with Val} \rightarrow a \text{ else } v \\ \mid \text{match } E \text{ with Exc} \rightarrow a \text{ else } v$$

2.2

$$\begin{array}{c}
 \text{(MATCH)} \\
 \frac{A \vdash a_0 : \langle \tau_1 \mid \tau_2 \rangle \quad A, x_1 : \tau_1 \vdash a_1 : \tau \quad A, x_2 : \tau_2 \vdash a_2 : \tau}{A \vdash \text{match } a_0 \text{ with Val } x \rightarrow a \mid \text{Exc } x_2 \rightarrow a_2 : \tau}
 \end{array}$$

2.3 L'expression ML correspondante est **match** a_0 **with** **Val** $x \rightarrow a$ ou encore **let** **Val** $x = a_0$ **in** a . Sa règle de typage est

$$\frac{A \vdash a_0 : \langle \tau_1 \mid \tau_2 \rangle \quad A, x : \tau_1 \vdash a : \tau}{A \vdash \text{match } a_0 \text{ with Val } x \rightarrow a : \tau}$$

Son évaluation peut se bloquer (ou lancer une exception "Match failure"). Si l'on pouvait typer simultanément **match** a_0 **with** **Val** $x \rightarrow a$ **else** v et **nil**, alors **nil** un type de la forme $\langle \tau_1 \mid \tau_2 \rangle \rightarrow \tau$. Mais alors on pourrait appliquer **nil** à une valeur de type τ , expression dont l'évaluation est bloquée (il n'y a pas de règle pour réduire **nil**).

3.1

$$\begin{array}{c}
 \text{(VAL)} \qquad \qquad \qquad \text{(EXC)} \\
 \frac{A \vdash a : \tau_1}{A \vdash \text{Val } a : \langle \text{Pre } \tau_1 \mid \tau_2 \rangle} \qquad \frac{A \vdash a : \tau_2}{A \vdash \text{Val } a : \langle \tau_1 \mid \text{Pre } \tau_2 \rangle} \\
 \\
 \text{(MATCH)} \\
 \frac{A \vdash a_0 : \langle \text{Pre } \tau_1 \mid \tau_2 \rangle \quad A, x : \tau_1 \vdash a : \tau \quad A \vdash v : \langle \tau' \mid \tau_2 \rangle \rightarrow \tau}{A \vdash \text{match } a_0 \text{ with Val } x \rightarrow a \text{ else } v : \tau}
 \end{array}$$

3.2 Il faut qu'il n'existe pas de valeur v de type $\langle \text{Abs} \mid \text{Abs} \rangle$.

3.3 On a la dérivation

$$\frac{\vdash (\lambda x. \text{Val } x) (\lambda y. y) : \langle \text{Pre } (\alpha \rightarrow \alpha) \mid \text{Abs} \rangle \quad z : (\alpha \rightarrow \alpha) \vdash z : (\alpha \rightarrow \alpha)}{\text{match } (\lambda x. \text{Val } x) (\lambda y. y) \text{ with Val } z \rightarrow z \text{ else nil} : \alpha \rightarrow \alpha}$$

L'expression a ne peut pas s'évaluer sur **Exc** v , sinon l'évaluation de a serait bloquée, ce qui ne peut arriver que si a est mal typé.

3.4 Une dérivation de typage de **Exc** v ne peut que se terminer par la règle **Exc**. Elle ne peut donc dériver qu'un type de la forme $\langle \tau_1 \mid \text{Pre } \tau_2 \rangle$ qui est incompatible avec un type de la forme $\langle \tau \mid \text{Abs} \rangle$.

L'expression a a pour type $\langle \text{Pre } (\alpha \rightarrow \alpha) \mid \text{Abs} \rangle$ donc son évaluation sur une valeur a le même type et ne peut pas être de la forme **Exc** v . Cela se généralise à toute expression a telle que $\vdash a : \langle \tau_1 \mid \text{Abs} \rangle$.

4.1 Par définition, $\llbracket a_1 \ a_2 \rrbracket$ vaut **match** $\llbracket a_1 \rrbracket$ **with** **Val** $\rightarrow a$ **else** $\lambda z. z$ où a vaut $\lambda x. (\text{match } \llbracket a_1 \rrbracket \text{ with Val} \rightarrow \lambda y. x \text{ else } \lambda z. z)$. Le tout se réduit en **match** v_1 **with** **Val** $\rightarrow a$ **else** $\lambda z. z$, puis $a \ v_1$, et successivement (**match** $\llbracket a_2 \rrbracket$ **with** **Val** $\rightarrow \lambda y. v_1 \ y$ **else** $\lambda z. z$), puis (**match** v_2 **with** **Val** $\rightarrow \lambda y. v_1 \ y$ **else** $\lambda z. z$) et finalement $(\lambda y. v_1 \ y) v_2$, puis $v_1 \ v_2$.

4.2 Une valeur v dans L est une fonction $\lambda x. a$. On note $\langle\langle v \rangle\rangle$ pour $\lambda x. \llbracket a \rrbracket$. On a donc $\llbracket v \rrbracket$ égal à **Val** $\langle\langle v \rangle\rangle$.

Cas raise (**raise** v) $\rightarrow$ **raise** v : $\llbracket \text{raise } v \rrbracket^V$ vaut **Exc** $\langle\langle v \rangle\rangle$. $\llbracket a_1 \rrbracket$ vaut **match** **Exc** $\langle\langle v \rangle\rangle$ **with** **Val** $\rightarrow \lambda x. \text{Exc } x$ **else** $\lambda z. z$ et se réduit en **Exc** $\langle\langle v \rangle\rangle$.

Cas $(\text{raise } v) a \longrightarrow \text{raise } v$: $\llbracket a_1 \rrbracket^V$ vaut `match Exc  $\langle\langle v \rangle\rangle$  with Val  $\rightarrow \lambda x. \llbracket a \rrbracket$  else  $\lambda z. z$` et se réduit sur `Exc  $\langle\langle v \rangle\rangle$` .

Cas $v_0(\text{raise } v) \longrightarrow \text{raise } v$: $\llbracket a_1 \rrbracket^V$ vaut `match Exc  $\langle\langle v \rangle\rangle$  with Val  $\rightarrow \lambda y. v_0 y$  else  $\lambda z. z$` et se réduit sur $\llbracket a_1 \rrbracket^V$.

Cas `try raise  $v_0$  with  $v$` $\longrightarrow v v_0$: $\llbracket a_1 \rrbracket^V$ vaut `match Exc  $\langle\langle v \rangle\rangle_0$  with Exc  $y \rightarrow \llbracket v \rrbracket y$  else  $\lambda z. z$` et se réduit sur $\langle\langle v \rangle\rangle \langle\langle v \rangle\rangle_0$ (on trouverait $\llbracket v \rrbracket$ si on l'on n'avait pas ajouté y), qui est aussi égal à $\llbracket v v_0 \rrbracket$.

4.3 Comme $\llbracket E\{a\} \rrbracket$ est égal à $\llbracket E \rrbracket \{\llbracket a \rrbracket\}$ et $\llbracket E \rrbracket$ est un contexte d'évaluation, à une réduction dans L de la forme $E\{a_1\} \longrightarrow E\{a_2\}$, correspond une évaluation

$$\llbracket E \rrbracket^V \{\llbracket a_1 \rrbracket^V\} \longrightarrow \llbracket E \rrbracket^V \{\llbracket a_2 \rrbracket^V\} \quad \text{i.e.} \quad \llbracket E\{a_1\} \rrbracket^V \longrightarrow \llbracket E\{a_2\} \rrbracket^V$$

5.1 $\llbracket a \rrbracket$ a le type $\langle \text{Val } \tau_0 \mid \tau \rangle$ pour tout type τ , en particulier pour τ égal à **Abs** ; $\llbracket a \rrbracket$ ne peut donc pas s'évaluer vers une valeur de la forme **Exc**. Par conséquent, le programme a ne bloque pas sur une exception, sinon il on aurait $a \longrightarrow \text{raise } v$ et donc $\llbracket a \rrbracket^V \longrightarrow \text{Exc } \langle\langle v \rangle\rangle$.

De façon générale, si $\llbracket a \rrbracket$ a un type de la forme $\langle \tau \mid \text{Abs} \rangle$, alors a ne peut pas se bloquer sur une exception.

5.2 L'expression $\llbracket \lambda x. \text{raise } x \rrbracket$ vaut **Val** ($\lambda x. \llbracket \text{raise } x \rrbracket$) c'est donc une valeur. Son type est $\langle \alpha \rightarrow \langle \alpha' \mid \text{Pre } \alpha \rangle \mid \alpha'' \rangle$.

Appliquée à une expression v elle produit une exception. Cela se voit parce que le type du résultat de cette application est $\langle \alpha' \mid \text{Pre } \alpha \rangle$ et n'est donc pas de la forme $\langle \tau' \mid \text{Abs} \rangle$.

5.3

$$\begin{array}{c} \text{(VAL)} \\ \frac{A \vdash a : \tau_1}{A \vdash \text{Val } a : \langle \text{Pre } \tau_1 \mid \tau_2 \rangle} \end{array} \quad \begin{array}{c} \text{(EXC)} \\ \frac{A \vdash a : \tau_2}{A \vdash \text{Val } a : \langle \text{Pre } \tau_1 \mid \text{Pre } \tau_2 \rangle} \end{array}$$

$$\begin{array}{c} \text{(MATCH)} \\ \frac{A \vdash a_0 : \langle \text{Pre } \tau_1 \mid \tau_2 \rangle \quad A, x : \tau_1 \vdash a : \tau \quad A \vdash v : \langle \text{Pre } \tau' \mid \tau_2 \rangle \rightarrow \tau}{A \vdash \text{match } a_0 \text{ with Val } x \rightarrow a \text{ else } v : \tau} \end{array}$$

5.4

$$\begin{array}{c} \frac{x : \tau \in A}{A \vdash x : \tau, \rho} \quad \frac{A \vdash a_1 : \tau_2 \xrightarrow{\rho} \tau_1, \rho \quad A \vdash a_2 : \tau_2, \rho}{A \vdash a_1 a_2 : \tau_1, \rho} \quad \frac{A, x : \tau_1 \vdash a : \tau_2, \rho}{A \vdash a : \tau_1 \xrightarrow{\rho} \tau_2} \\ \\ \frac{A, x : \sigma \vdash a : \tau, \rho}{A \vdash a : \tau_1 \xrightarrow{\rho} \tau_2} \quad \frac{A \vdash a : \tau, \text{Pre } \tau}{A \vdash \text{raise } a : \tau', \text{Pre } \tau} \quad \frac{A \vdash a : \tau, \text{Pre } \tau' \quad A \vdash v : \tau' \xrightarrow{\rho} \tau}{A \vdash \text{try } a \text{ with } v : \tau, \rho} \end{array}$$

5.5 Les exceptions multiples sont équivalentes à une exception simple dont la valeur est un type somme. On pourra en particulier choisir un type variante, comme nous l'avons fait dans le langage L' . L'extension ne pose donc pas de problème.