Lazy products

In class on Tuesday, we learned about eager products. When eliminating eager products, we first reduced the components from left to right until both were values. Only then did we deem the pair to be a value. This is because we wanted to simultaneously use both components in the case elimination form, and in a call-by-value semantics, variables should only ever be instantiated with values.

With lazy products, a pair of expressions is always a value. Instead of a case elimination form that simultaneously extracts both components of an ordered pair, lazy products have left and right projections that extract the single corresponding component from the pair. Before we can project out of a term that is not an ordered pair, we must first reduce that term until it becomes one.

In this part of the assignment, you will explore the dynamics for lazy products and prove various properties about them. We explore lazy products in the context
of the simple language we have seen so far. Its syntax, statics, and dynamics are given in the appendices.

We first extend our syntax:

\[
\begin{align*}
\tau &::= \cdots \\
&| \tau_1 \& \tau_2 \quad \text{lazy product of } \tau_1 \text{ and } \tau_2 \\
\end{align*}
\]

\[
\begin{align*}
e &::= \cdots \\
&| \langle| e_1, e_2 |\rangle \quad \text{ordered pair of } e_1 \text{ and } e_2 \\
&| e \cdot l \quad \text{left projection} \\
&| e \cdot r \quad \text{right projection} \\
\end{align*}
\]

The introduction rule for lazy pairs is:

\[
\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \\
\Gamma \vdash \langle| e_1, e_2 |\rangle : \tau_1 \& \tau_2 \quad (I-\&) \\
\]

Its elimination rules are:

\[
\begin{align*}
\Gamma \vdash e : \tau_1 \& \tau_2 \\
\Gamma \vdash e \cdot l : \tau_1 \quad (E-\&-L) \\
\Gamma \vdash e \cdot r : \tau_2 \quad (E-\&-R) \\
\end{align*}
\]

Task 1 (10 points, WB). Give rules capturing the intended dynamics for lazy products. That is, give rules defining the judgments \(e \ val\) and \(e \mapsto e'\) for terms \(e\) of the form \(\langle| e_1, e_2 |\rangle\), \(e_0 \cdot l\), and \(e_0 \cdot r\). They should satisfy \(\langle| e_1, e_2 |\rangle \cdot l \mapsto^* e_1\) and \(\langle| e_1, e_2 |\rangle \cdot r \mapsto^* e_2\) and capture the intuitions we gave above.

Solution 1: Pairs are always values. We step under the projections until they reach a value, and which point we extract the appropriate component.

\[
\begin{align*}
\langle| e_1, e_2 |\rangle \ val \quad (&-\text{VAL}) \\
\end{align*}
\]

\[
\begin{align*}
e \mapsto e' \\
\hline
e \cdot l \mapsto e' \cdot l \quad (\text{STEP-PROJL-1}) \\
\hline
\end{align*}
\]

\[
\begin{align*}
e \mapsto e' \\
\hline
e \cdot r \mapsto e' \cdot r \quad (\text{STEP-PROJR-1}) \\
\end{align*}
\]

\[
\begin{align*}
\langle| e_1, e_2 |\rangle \cdot l \mapsto e_1 \quad (\text{STEP-PROJL-2}) \\
\hline
\langle| e_1, e_2 |\rangle \cdot r \mapsto e_2 \quad (\text{STEP-PROJR-2}) \\
\end{align*}
\]

\begin{itemize}
\item Task 2 (10 points, WB). State and prove the canonical forms theorem for lazy pairs. The statement should be of the following form: “if \(\vdash e : \tau_1 \& \tau_2\) and \(e \ val\), then \(e\) is a term of the form(s) \ldots”.
\end{itemize}
Solution 2: The statement is: if $\cdot \vdash e : \tau_1 \& \tau_2$ and $e \text{ val}$, then $e$ is a term of the form $\langle e_1, e_2 \rangle$. The proof is by induction the judgment $e \text{ val}$.

Case $(\&-\text{VAL})$. Then $e$ is $\langle e_1, e_2 \rangle$ for some $e_1$ and $e_2$. We proceed by induction on the derivation $\cdot \vdash \langle e_1, e_2 \rangle : \tau_1 \& \tau_2$. The only rule with a conclusion of this form is $(I-\&)$, and by inversion we get $\cdot \vdash e_i : \tau_i$.

Case $(\rightarrow-\text{VAL})$. Then $e$ is $\lambda x.e'$ for some $e'$. There are no typing rules with conclusions of the form $\cdot \vdash \lambda x.e' : \tau_1 \& \tau_2$, so this case is impossible.

Case $(\text{VAL}/\text{L})$. Then $e$ is $l \cdot e'$ for some $e'$. There are no typing rules with conclusions of the form $\cdot \vdash l \cdot e' : \tau_1 \& \tau_2$, so this case is impossible.

Case $(\text{VAL}/\text{R})$. Symmetric to $(\text{VAL}/\text{L})$.

Case $(\text{PAIR}-\text{VAL})$. Then $e$ is $\langle e_1, e_2 \rangle$ for some $e_1$ and $e_2$. There are no typing rules with conclusions of the form $\cdot \vdash \langle e_1, e_2 \rangle : \tau_1 \& \tau_2$, so this case is impossible.

There are no other rules with conclusions of the form $e \text{ val}$, so this completes the proof. $\square$

Task 3 (10 points, WB). Show that the progress property still holds after the addition of lazy pairs. In particular, prove if $\cdot \vdash e : \tau$ by $(I-\&)$, $(E-\&-\text{L})$, or $(E-\&-\text{R})$, then either $e \text{ val}$ or there exists an $e'$ such that $e \mapsto e'$.

Solution 3: We proceed by induction on the judgment $\cdot \vdash e : \tau$. All cases but the following are unchanged.

Case $(I-\&)$. Then $e$ is $\langle e_1, e_2 \rangle$ for some $e_1$ and $e_2$. We have $e \text{ val}$ by $(\&-\text{VAL})$.

Case $(E-\&-\text{L})$. Then $\cdot \vdash e \cdot l : \tau_1$ for some $e$ such that $\cdot \vdash e : \tau_1 \& \tau_2$. By the induction hypothesis, either $e \text{ val}$ or $e \mapsto e'$. In the first case, $e \equiv \langle e_1, e_2 \rangle$ by Task 2. Then $e \cdot l \mapsto e_1$ by $(\text{STEP-PROJL}-2)$. In the second case, $e \cdot l \mapsto e' \cdot l$ by $(\text{STEP-PROJL}-1)$.

Case $(E-\&-\text{R})$. Symmetric to the previous case. $\square$

Task 4 (5 points, WB). Show that the preservation property still holds after the addition of lazy pairs. In particular, show that if $e \mapsto e'$ by one of the rules you gave in task 1 and $\cdot \vdash e : \tau$, then $\cdot \vdash e' : \tau$.

Solution 4: We proceed by induction on the judgment $e \mapsto e'$.

Case $(\text{STEP-PROJL}-1)$. Then $e \cdot l \mapsto e' \cdot l$ and $e \mapsto e'$. The only rule with a conclusion of the form $\cdot \vdash e \cdot l : \tau$ is $(E-\&-\text{L})$. By inversion, $\cdot \vdash e : \tau \& \tau_2$ for some $\tau_2$. By the induction hypothesis on $e \mapsto e'$, we have $\cdot \vdash e' : \tau \& \tau_2$. By $(E-\&-\text{L})$, we have $\cdot \vdash e' \cdot l : \tau$ as desired.

Case $(\text{STEP-PROJR}-1)$. Symmetric to the previous case.

Case $(\text{STEP-PROJL}-2)$. Then $\langle e_1, e_2 \rangle \cdot l \mapsto e_1$. The only rule with a conclusion of the form $\cdot \vdash \langle e_1, e_2 \rangle \cdot l : \tau$ is $(E-\&-\text{L})$. By inversion, $\cdot \vdash \langle e_1, e_2 \rangle : \tau \& \tau_2$ for some $\tau_2$. By inversion (rule $(I-\&)$), $\cdot \vdash e_1 : \tau$, as desired.
Case (STEP-PROJ-R-2). Symmetric to the previous case.

Task 5 (5 points, WB). Write down terms $\xi$ and $\xi^{-1}$ such that

$$
\cdot \vdash \xi : \tau \otimes \sigma \rightarrow \tau \& \sigma \\
\cdot \vdash \xi^{-1} : \tau \& \sigma \rightarrow \tau \otimes \sigma
$$

and for all values $v$ of type $\tau \otimes \sigma$, $\xi^{-1}(\xi(v)) \rightarrow^* v$ (you do not need to prove this).

Do these terms witness the isomorphism? If so, state this. If not, briefly explain why.

**Hint.** Consider the dynamics of eager and lazy products in the presence of non-termination.

**Solution 5:** Let $\xi : \tau \otimes \sigma \rightarrow \tau \& \sigma$ and $\xi^{-1} : \tau \& \sigma \rightarrow \tau \otimes \sigma$ be given by

$$
\xi = \lambda x. \text{case } x \{ (l, r) \Rightarrow \langle| l, r |\rangle \} \\
\xi^{-1} = \lambda x. (x \cdot l, x \cdot r)
$$

By the canonical forms theorem in lecture 7, all values $v$ of type $\tau \otimes \sigma$ are of the form $\langle v_1, v_2 \rangle$ with $v_1$ a value of type $\tau$ and $v_2$ a value of type $\sigma$. Then for all such $\langle v_1, v_2 \rangle$,

$$
\xi^{-1}(\xi(\langle v_1, v_2 \rangle)) \equiv (\lambda x. (x \cdot l, x \cdot r))(\lambda x. \text{case } x \{ (l, r) \Rightarrow \langle| l, r |\rangle \})(v_1, v_2) \\
\rightarrow (\lambda x. (x \cdot l, x \cdot r))(\text{case } \langle v_1, v_2 \rangle \{ (l, r) \Rightarrow \langle| l, r |\rangle \}) \\
\rightarrow (\lambda x. (x \cdot l, x \cdot r))(\langle| v_1, v_2 |\rangle \{ l, r \}) \\
\equiv (\lambda x. (x \cdot l, x \cdot r))\langle| v_1, v_2 |\rangle \\
\rightarrow (\langle| v_1, v_2 |\rangle \cdot l, \langle| v_1, v_2 |\rangle \cdot r) \\
\rightarrow \langle| v_1, v_2 |\rangle.
$$

These mappings fail to establish an isomorphism for several reasons. First, $\langle| e_1, e_2 |\rangle \equiv l$ even if $e_1$ diverges (does not reduce to a value). Then $\xi(\xi^{-1}(\langle| e_1, e_2 |\rangle))$ will diverge because $\langle| e, v \rangle \cdot l, \langle| e_1, e_2 \rangle \cdot r \rangle$ diverges and

$$
\xi(\xi^{-1}(\langle| e_1, e_2 |\rangle)) \equiv (\lambda x. \text{case } x \{ (l, r) \Rightarrow \langle| l, r |\rangle \})(\lambda x. (x \cdot l, x \cdot r))\langle| e_1, e_2 |\rangle \\
\rightarrow (\lambda x. \text{case } x \{ (l, r) \Rightarrow \langle| l, r |\rangle \})\langle| e_1, e_2 |\rangle \cdot l, \langle| e_1, e_2 \rangle \cdot r \rangle.
$$

Assume next that $e_1$ and $e_2$ converge to $v_1$ and $v_2$, respectively. In this case, $\xi(\xi^{-1}(\langle| e_1, e_2 |\rangle)) \rightarrow^* \langle| v_1, v_2 |\rangle$, and it is possible that $\langle| e_1, e_2 \rangle \neq \langle| v_1, v_2 \rangle$.

The assigned tasked also asked you to speculate on how we could modify the definition of mutual inverse so that $\xi$ and $\xi^{-1}$ became mutual inverses. This portion of the task was misguided because $\tau \otimes \sigma$ and $\tau \& \sigma$ are fundamentally
different types and should not be isomorphic. Indeed, it is laziness of $\&$ that permits us to encode infinite streams, something impossible under the eagerness of $\otimes$. A better formulation would have been “characterize the values $v$ for which $\xi(\xi^{-1}(v)) \mapsto v$”. This portion of the task was ignored during grading.

A billion dollar mistake

A frequent source of bugs programs is attempting to dereference null pointers. Languages subject to this class of errors feature a type $\pi$ of “pointers” with a distinguished pointer null called the “null pointer”. The intention is that the null pointer represent a lack of value, while all other pointers can be “dereferenced” to produce a value. Any attempt to dereference a null pointer is necessarily an error. This class of errors is so rampant that Hoare, the inventor of the null pointer, has called null pointers his “billion dollar mistake” [2].

The crux of the problem is a mis-identification of the type option type $\pi$ opt with the type $\pi$. The option type $\tau$ opt represents optional values of type $\tau$, where some $e$ captures the presence of a value $e$ of type $\tau$ and none represents the lack of value. We can test for the presence of a value using the construction “case $e_1 \{ \text{some } x \Rightarrow e_2 | \text{none } \Rightarrow e_3 \}””. This checks if $e_1$ is none, in which it produces $e_3$, and if $e_1$ is some $v_1$, then we produce $[v_1/x]e_2$. This can be encoded using sum types as follows:

$$\tau \text{ opt } = \tau + 1$$
$$\text{some } e = l \cdot e$$
$$\text{none } = r \cdot \langle \rangle$$

$$\text{case } e_1 \{ \text{some } x \Rightarrow e_2 | \text{none } \Rightarrow e_3 \} = \text{case } e_1 \{ l \cdot x \Rightarrow e_2 | r \cdot \_ \Rightarrow e_3 \}$$

The key point is that the type system stops us from directly using an optional value $e$ of type $\tau$ opt in a context expecting a value of type $\tau$: the type system forces us to account for the fact that $e$ could be none. In contrast, $\pi$ treats the null pointer null as a genuine pointer, allowing it to be used in any context expecting a genuine pointer.

The affected languages try to work around this problem by providing a function isnull : $\pi \rightarrow \text{bool}$ that produces true if it is applied to null, and otherwise produces false. To avoid accidentally dereferencing a pointer, code is then littered with checks of the form:

$$\text{if (isnull } e) \text{ then } \ldots \text{handle error} \ldots \text{ else } \ldots \text{do something} \ldots$$

Unfortunately, it is very easy to forget such a check and attempt to dereference the potentially null value $e$. Implicitly, this work-around attempts to simulate
the type \( \tau \text{opt} \) using \( \text{bool} \otimes \tau \), where a value of type \( \tau \) is tagged with a boolean that signals whether or not the value is actually present. The reader is referred to [1, pp. 92f.] for more details.

**Task 6** (10 points, WB, [1, Ex. 11.2]). Informally show that we cannot identify \( \text{bool} \otimes \tau \) and \( \tau \text{opt} \) for arbitrary \( \tau \). Do so by attempting to give an implementation of \( \tau \text{opt} \) in terms of \( \text{bool} \otimes \tau \) by sensibly completing the following chart:

\[
\begin{align*}
\text{none} &= ? \\
\text{some } e &= ? \\
\text{case } e_1 \{ \text{some } x \Rightarrow e_2 \mid \text{none} \Rightarrow e_3 \} &= ?
\end{align*}
\]

Where do you get stuck if you do not additionally assume the existence of a “null” value \( \text{null}_\tau \) for each type \( \tau \)? Argue that even by artificially assuming the existence of such values, you cannot complete the chart in a manner that captures the semantics of \( \tau \text{opt} \) in their absence. (Please be brief: your answer must contain at most eight lines of prose.)

For your convenience, you may assume the existence of an \( \text{if } b \text{ then } e_1 \text{ else } e_2 \) construct as defined in lecture 6.

**Solution 6:** Without the existence of \( \text{null}_\tau \), we get stuck when we try to implement \( \text{none} \): we have no a priori means of conjuring a closed value of type \( \tau \) for arbitrary \( \tau \) (consider \( \tau = 0 \)), but we need such a value for the second component of the pair. Using \( \text{null}_\tau \), we could sensibly attempt to complete the chart as:

\[
\begin{align*}
\text{none} &= \langle \text{false}, \text{null}_\tau \rangle \\
\text{some } e &= \langle \text{true}, e \rangle \\
\text{case } e_1 \{ \text{some } x \Rightarrow e_2 \mid \text{none} \Rightarrow e_3 \} &= \text{case } e_1 \{ \langle b, v \rangle \Rightarrow \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{if } b \ \text{then } (\lambda x.e_2)v \ \text{else } e_3 \}.
\end{align*}
\]

We could alternatively leverage the variable convention and use a binding trick:

\[
\begin{align*}
\text{case } e_1 \{ \text{some } x \Rightarrow e_2 \mid \text{none} \Rightarrow e_3 \} &= \text{case } e_1 \{ \langle b, x \rangle \Rightarrow \\
&\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{if } b \ \text{then } e_2 \ \text{else } e_3 \}.
\end{align*}
\]

This fails to capture the semantics of \( \tau \text{opt} \) because we might attempt to \text{case} on \( \langle \text{true}, \text{null}_\tau \rangle \). This pair has no corresponding genuine option type expression and the \text{case} expression steps to the illegal term \([\text{null}_\tau/x]e_2\).
Gaining types

Task 7 (5 points, WB). Find terms $e$ and $e'$ and types $\tau$ and $\tau'$ such that $\cdot \vdash e : \tau$, $e \mapsto e'$, and $\cdot \vdash e' : \tau'$, but not $\cdot \vdash e : \tau'$.

Solution 7: There were many creative solutions. Here are a few.

Consider

$$e = \text{case } l \cdot \langle \rangle \{ l \cdot x \Rightarrow l \cdot \langle \rangle \mid r \cdot x \Rightarrow r \cdot \langle \rangle \}.$$  

Then $\cdot \vdash e : 1 + 1$ has a unique type, but $e \mapsto l \cdot \langle \rangle$ and $\cdot \vdash l \cdot \langle \rangle : 1 + \tau$ for all $\tau$.

Consider

$$e = (\lambda f. \text{case } f \langle \rangle \{ \langle \rangle \Rightarrow f \}) (\lambda x.x).$$  

Then $\cdot \vdash e : 1 \to 1$ has a unique type, but $e \mapsto \text{case } (\lambda x.x) \langle \rangle \{ \langle \rangle \Rightarrow \lambda x.x \}$, which has type $\tau \to \tau$ for all $\tau$.

Consider

$$e = \text{case } (r \cdot \langle \rangle) \{ l \cdot \_ \Rightarrow \lambda f. \lambda x.f x \mid r \cdot \_ \Rightarrow \lambda x.x \}.$$  

Then $\cdot \vdash e : (\alpha \to \alpha) \to (\alpha \to \alpha)$ but not $\cdot \vdash e : \alpha \to \alpha$. However, $e \mapsto \lambda x.x$, which has type $\tau \to \tau$ for all $\tau$.

Is zero a zero?

Task 8 (5 points, WB). Is the nullary sum type $0$ the zero for $\otimes$? That is, prove or disprove that for all $\tau$,

$$\tau \otimes 0 \cong 0.$$  

In the affirmative case, we only ask that you write down the two terms witnessing the isomorphism. In the negative case, you must provide a counter-example and explain why it is a counter-example.

Solution 8: The isomorphism is witnessed by the following terms:

$$\phi = \lambda x. \text{case } x \{ \langle z \rangle \Rightarrow z \},$$  

$$\phi^{-1} = \lambda x. \text{case } x \{ \}. $$  

There are no values of type $0$, nor of type $\tau \otimes 0$ by the canonical forms theorem for $\otimes$, so these two functions are vacuously mutual inverses.

Task 9 (0 points). How long did you spend on this assignment? Please list the questions that you discussed with classmates using the whiteboard policy.
References
