Assignment 5
Affine Logic and SAX

15-836: Substructural Logics
Frank Pfenning

Due Friday, November 10, 2023
125 points

In this assignment we explore subtyping on message-passing programs.
You should hand in two files:

- `hw5.pdf` with the solutions to Problems 1 and 2.
- `hw5.sax` with the programs written in the SAX language as specified in Problem 3. This file is autograded.

The handout has the following structure:

- `src/`, the implementation of SAX.
- `starter/`, the starter code for this assignment, including a template for `hw5.sax`
- `hw5.pdf`, this assignment spec
- `*.{tex,sty}`, LaTeX source, macros, and style files

1 Affine Logic (35 pts)

Affine logic has recently received some interest because the type system of the programming language Rust is affine (even if it often described informally as linear). We obtain affine logic from linear logic by adding the rule of weakening, but not the rule of contraction. There is also an implicit formulation where we allow unused antecedents in the rules with no premises. So either we have the rule on the left, or we modify the rules id and $\mathbf{1}R$ as shown on the right while leaving the other rules unchanged.

explicit **implicit**

$$\frac{\Delta \overset{e}{\vdash} C}{\Delta, A \overset{e}{\vdash} C}\ \mathsf{weaken} \qquad\qquad \frac{}{\Delta \overset{i}{\vdash} \mathbf{1}}\ \mathbf{1}R_W \qquad \frac{}{\Delta, A \overset{i}{\vdash} A}\ \mathsf{id}_W$$

Task 1 (10 pts) Sketch the proof that the explicit and implicit version of affine logic are equivalent in the sense that $\Delta \overset{e}{\vdash} A$ if and only if $\Delta \overset{i}{\vdash} A$. Clearly explain the structure of your proof in each direction and show at least one interesting case for each direction.

Task 2 (15 pts) We can interpret affine logic in linear logic with a translation of affine propositions $(A)^w$ such that $\Delta \vdash A$ in affine logic if and only if $(\Delta)^w \vdash (A)^w$ in linear logic. Here, $(\Delta)^w$ means that the w-translation is applied to each proposition in Δ. Provide such a translation for the propositions

$$A, B \quad ::= \quad P \mid A \oplus B \mid A \otimes B \mid \mathbf{1} \mid A \multimap B \mid A \mathbin{\&} B$$

We have purposely omitted $\mathbf{0}$ and $\top$ (which should be neither affine nor linear propositions) since they might reduce the chance that the translation is computationally meaningful.

Task 3 (10 pts) Sketch the proof that the translation from the previous task is correct, where you are free to choose either the explicit or implicit presentation. Clearly explain the structure of your proof in each direction and show at least one interesting case for each direction.

2 Implementing Weakening (30 pts)

As many of you discovered in a previous homework, weakening of *purely positive* propositions is admissible in linear logic. In this problem you are asked to (abstractly) define a family of processes in SAX that implement weakening.

Task 4 (30 pts) Define a family of processes $\mathit{drop}_A\ (u : \mathbf{1})\ (x : A)$ for

$$A, B \quad ::= \quad t \mid A \otimes B \mid \mathbf{1} \mid \oplus\{\ell : A_\ell\}_{\ell \in L} \mid {\downarrow} S$$

where S is an arbitrary structural type while all other types are linear. Here, t stands for type names that are defined with equations $t = A$ (in a possibly mutually recursive way) in a global signature Σ.

Your solution should not use message sequences (which are derived from partial focusing), just messages $k(x')$, (y, x'), $(\,)$ and $\langle x' \rangle$ as needed.

Each process drop_A should of course be well-typed and terminating as long as the input along $x : A$ is finite. You do not need to prove either property.

3 Sets of Binary Numbers as Tries (60 pts)

In this problem we implement sets of binary numbers as binary tries in SAX. We enforce through typing that the binary numbers have no leading zeros because otherwise the trie would think of $\epsilon 100$ and $\epsilon 0100$ as two different numbers even though both have the value 4. It isn't particularly relevant here, but recall that our representation of binary numbers starts with the least significant bit.

Tries are not unique. For example,

```
'leaf()
```

and

```
'node('leaf(),'false(),'leaf())
```

both represent an empty trie. A trie just containing the number $\epsilon 1$ could be

```
'node('leaf(),'false(),'node('leaf(),'true(),'leaf()))
```

In your code, you do not need to be concerned with issues of efficiency in terms of time or space—simplicity is the key.

We have taken a liberty with the notation above in that these are not quite valid message sequences. The second would actually be displayed as

```
a0 -> 'node.(a1).(a2).'leaf.()
a1 -> 'leaf.()
a2 -> 'false.()
```

for channels `a0`, `a1`, and `a2`. If you run SAX with

```
./sax --subtyping -d hw5.sax
```

it will show observations not just as message sequences, but also in tree form. The switch `--subtyping` is necessary because of the use of subtypes to capture binary numbers without leading zeros.

We show below the starter code with brief specifications for each process.

```
type bool = +{'false : 1, 'true : 1}

type std = +{'b0 : pos, 'b1 : std, 'e : 1}
type pos = +{'b0 : pos, 'b1 : std        }

type trie = +{'node : trie * bool * trie, 'leaf : 1}

(* r = {} *)
proc empty (r : trie)

(* r = {x} *)
proc singleton (r : trie) (x : std)

(* r = s union t *)
proc union (r : trie) (s : trie) (t : trie)

(* r = s diff t = {x in s | x not in t} *)
proc diff (r : trie) (s : trie) (t : trie)

(* a set of numbers as an object *)
type set = &{'insert : std -o set,
             'delete : std -o set,
             'member : std -o bool * set,
             'dealloc : 1}

(* S represents the empty set *)
proc new_set (S : set)
```