Type-safety et gestion mémoire

Langages de bas niveau posent des problèmes particuliers

- Séparer allocation mémoire et initialisation
- Éviter la désallocation prématurées
- Éviter les fuites

Il y en a d'autres, bien sûr, e.g. les "tag bits", la *taille* en bits

CPS: un lambda calcul séquentiel

Beaucoup de langages type-safe de bas niveau utilisent CPS

CPS = Continuation Passing Style

- Tous les appels de fonction sont en position terminale
- Remplacer les `return` par des appels à des *continuations*

Rend explicite l'ordre d'évaluation et la pile

Aussi utilisable pour implémenter: exceptions, `call/cc`, ...

Appel de fonction = "jump with arguments"

¿¡Alloue la pile dans le tas!?

Exemple de style CPS: λ^K

$$
\begin{array}{lrcl}
\textit{(types)} & \tau & ::= & \alpha \mid \texttt{Int} \mid \langle \vec{\tau} \rangle \mid \forall[\vec{\alpha}].(\vec{\tau}) \to \bot \\
\textit{(computations)} & e & ::= & x[\vec{\tau}](\vec{v}) \mid \texttt{if}\,(v, e_1, e_2) \mid \texttt{halt}\; v \\
 & & & \mid \texttt{let}\; x = v\; \texttt{in}\; e \mid \texttt{let}\; x = \mathit{op}(\vec{v})\; \texttt{in}\; e \\
\textit{(primitives)} & \mathit{op} & ::= & + \mid - \mid \pi_i \mid \ldots \\
\textit{(values)} & v & ::= & x \mid i \mid \langle \vec{v} \rangle \mid \texttt{fix}\, x[\vec{\alpha}](\overrightarrow{x:\tau}).e
\end{array}
$$

Distinction entre *values* et *computations*

Typage des valeurs: $\Delta; \Gamma \vdash v : \tau$

Typage des calculs: $\Delta; \Gamma \vdash e$ wf

Allocation et initialisation

```
let x = malloc(N);
x.0 := v1;  x.1 := v2;
```

Quel est le type de x? Où/quand?

On veut éviter: `let x = malloc(N); print(x.0);`

strong update: une affectation qui modifie le type de l'objet modifié

```
let x = malloc(N); x.0 := "hi"; x.0 := 42;
```

Problème:

```
let x = malloc(N); let y = x;
x.0 := "hi"; y.0 := 42;
```

Allocation selon “From System-F to TAL”

$$
\begin{array}{lrcl}
\textit{(types)} & \tau & ::= & \ldots \mid \langle \tau^{\vec{\varphi}} \rangle \mid \forall[\vec{\alpha}].(\vec{\tau}) \to \bot \\
\textit{(initialisation)} & \varphi & ::= & 0 \mid 1 \\
\textit{(computations)} & e & ::= & \ldots \\
 & & & \mid \texttt{let}\ x = \texttt{malloc}[\vec{\tau}]\ \texttt{in}\ e \\
 & & & \mid \texttt{let}\ x = v_1[i] \leftarrow v_2\ \texttt{in}\ e \\
\textit{(values)} & v & ::= & \ldots\text{mais sans } \langle \vec{v} \rangle
\end{array}
$$

Les “*strong updates*” ne changent que l’état d’initialisation

Autres références pas mises à jour: sauvé par la monotonicité!

Régions: Calculus of Capabilities

$$
\begin{array}{lrcl}
\textit{(types)} & \tau & ::= & \ldots \mid \langle \vec{\tau} \rangle^{r} \mid \forall[\vec{\alpha}]\{C\}(\vec{\tau}) \to \bot \\
\textit{(computations)} & e & ::= & \ldots \\
 & & & \mid \texttt{let}\ r = \texttt{newrgn}\ \texttt{in}\ e \\
 & & & \mid \texttt{freergn}\ r; e \\
 & & & \mid \texttt{let}\ x = \langle \vec{v} \rangle\ \texttt{at}\ r\ \texttt{in}\ e \\
\textit{(primitives)} & \mathit{op} & ::= & + \mid - \mid \pi_i \mid \ldots \\
\textit{(values)} & v & ::= & x \mid i \\
\textit{(prog)} & p & ::= & \texttt{let}\ \overrightarrow{x[\vec{\alpha}]\{C\}(\overrightarrow{x:\tau}) = e}\ \texttt{in}\ v[\vec{\tau}](\vec{v})
\end{array}
$$

Règles de typage des régions

Jugement: $\Delta; \Psi; \Gamma \vdash e$ wf

$$\frac{\Delta; \Psi, r; \Gamma \vdash e \text{ wf}}{\Delta; \Psi; \Gamma \vdash \texttt{let}\ r = \texttt{newrgn}\ \texttt{in}\ e \text{ wf}}$$

$$\frac{\Psi = \Psi_1, r, \Psi_2 \quad \Delta; \Psi_1, \Psi_2; \Gamma \vdash e \text{ wf}}{\Delta; \Psi; \Gamma \vdash \texttt{freergn}\ r; e \text{ wf}}$$

$$\frac{\Delta; \Gamma \vdash v : \langle \vec{\tau} \rangle^r \quad r \in \Psi \quad \Delta; \Psi; \Gamma, x : \tau_i \vdash e \text{ wf}}{\Delta; \Psi; \Gamma \vdash \texttt{let}\ x = v[i]\ \texttt{in}\ e \text{ wf}}$$

Alias types

$$
\begin{array}{lrcl}
\textit{(types)} & \tau & ::= & t \mid \texttt{Int} \mid \texttt{ptr}\,\ell \mid \forall[\vec{\alpha}]\{\Psi\}(\vec{\tau}) \to \bot \\
\textit{(computations)} & e & ::= & \ldots \\
 & & & \mid \texttt{let}\,\ell, x = \texttt{alloc}\,n\,\texttt{in}\,e \\
 & & & \mid \texttt{free}\,v; e \\
 & & & \mid v_1.i := v_2; e \\
\textit{(heaps)} & \Psi & ::= & \bullet \mid \Psi, \ell \mapsto \langle \vec{\tau} \rangle \\
\textit{(values)} & v & ::= & x \mid i \\
\textit{(prog)} & p & ::= & \texttt{let}\,\overrightarrow{x[\vec{\alpha}]\{\Psi\}(\overrightarrow{x:\tau}) = e}\,\texttt{in}\,v[\vec{\tau}](\vec{v})
\end{array}
$$

Règles de typage des alias types

Jugement: $\Delta; \Psi; \Gamma \vdash e$ wf

$$\frac{\Delta, \ell{:}\mathtt{Ptr}; \Psi, \ell \mapsto \langle \overrightarrow{\mathtt{Int}} \rangle; \Gamma, x{:}\mathtt{ptr}\ \ell \vdash e \text{ wf}}{\Delta; \Psi; \Gamma \vdash \mathtt{let}\ \ell, x = \mathtt{alloc}\ n\ \mathtt{in}\ e \text{ wf}}$$

$$\frac{\Delta; \Gamma \vdash v : \mathtt{ptr}\ \ell \quad \Psi = \Psi_1, \ell \mapsto \langle ... \rangle, \Psi_2 \quad \Delta; \Psi_1, \Psi_2; \Gamma \vdash e \text{ wf}}{\Delta; \Psi; \Gamma \vdash \mathtt{free}\ v; e \text{ wf}}$$

$$\frac{\begin{array}{c}\Delta; \Gamma \vdash v_1 : \mathtt{ptr}\ \ell \quad \Psi = \Psi_1, \ell \mapsto \langle \tau_0, .., \tau_i, .., \tau_n \rangle, \Psi_2 \\ \Delta; \Gamma \vdash v_2 : \tau \quad \Delta; \Psi_1, \ell \mapsto \langle \tau_0, .., \tau, .., \tau_n \rangle, \Psi_2; \Gamma \vdash e \text{ wf}\end{array}}{\Delta; \Psi; \Gamma \vdash v_1.i := v_2; e \text{ wf}}$$