Vérification de génération de code pour le modèle polyédrique

Nathanaël Courant

5 septembre 2018

Contexte

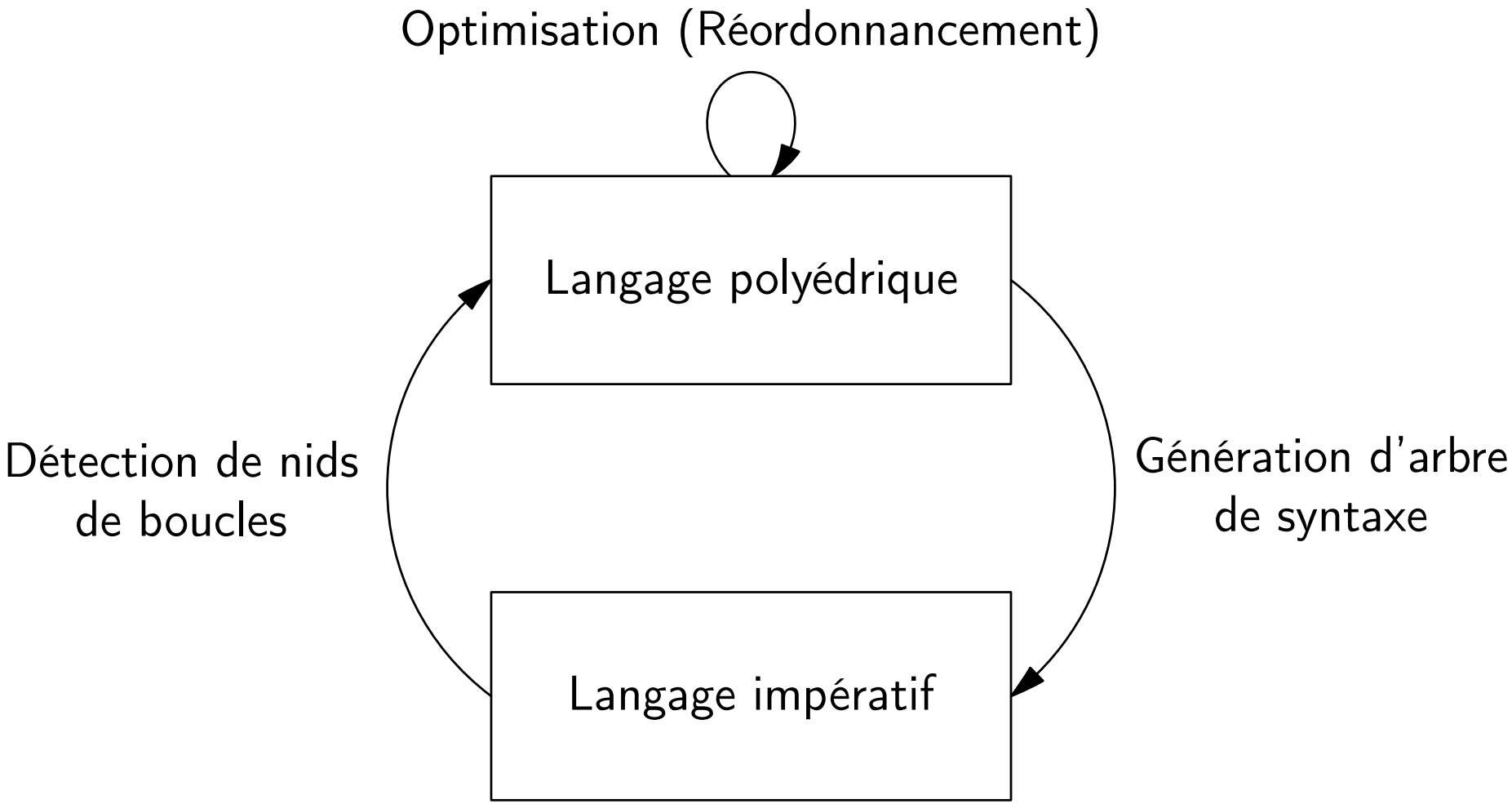
- Modèle polyédrique : optimisation de nids de boucles
- Optimisations de compilateur : risque d'erreur
- Objectif : vérification en Coq
- Ici : vérification de la génération de code

Le modèle polyédrique

Génération de l'arbre de syntaxe

Formalisation en Coq

Compilation polyédrique



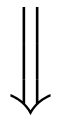
Instructions et programmes polyédriques

Instruction polyédrique : $(\mathtt{I}, \mathcal{D}, \theta, \mathcal{T})$

$\mathtt{I}$	instruction
$\mathcal{D} = \{x \in \mathbb{Z}^n / Ax \leq a\}$	domaine
$\theta = x \mapsto Kx + k$	fonction d'ordonnancement
$\mathcal{T} = x \mapsto Tx + t$	fonction de transformation

Programme polyédrique : ensemble d'instructions polyédriques

```
for (int m = 0; m < n; m++) {
  for (int k = 1; k < m; k++) {
    binom[m][k] = binom[m-1][k-1] + binom[m-1][k];
  }
}
```



$$\mathrm{I}(\mathrm{m},\mathrm{k}) = \texttt{binom[m][k]} = \texttt{binom[m-1][k-1] + binom[m-1][k]}$$

$$\mathcal{D} = \{(\mathrm{n},\mathrm{m},\mathrm{k})/1 \leq \mathrm{k} < \mathrm{m} < \mathrm{n}\}$$

$$\theta(\mathrm{n},\mathrm{m},\mathrm{k}) = (\mathrm{m},\mathrm{k})$$

$$\mathcal{T}(\mathrm{n},\mathrm{m},\mathrm{k}) = (\mathrm{m},\mathrm{k})$$

Sémantique

Sémantique dans un environnement $\mathcal{E}$:

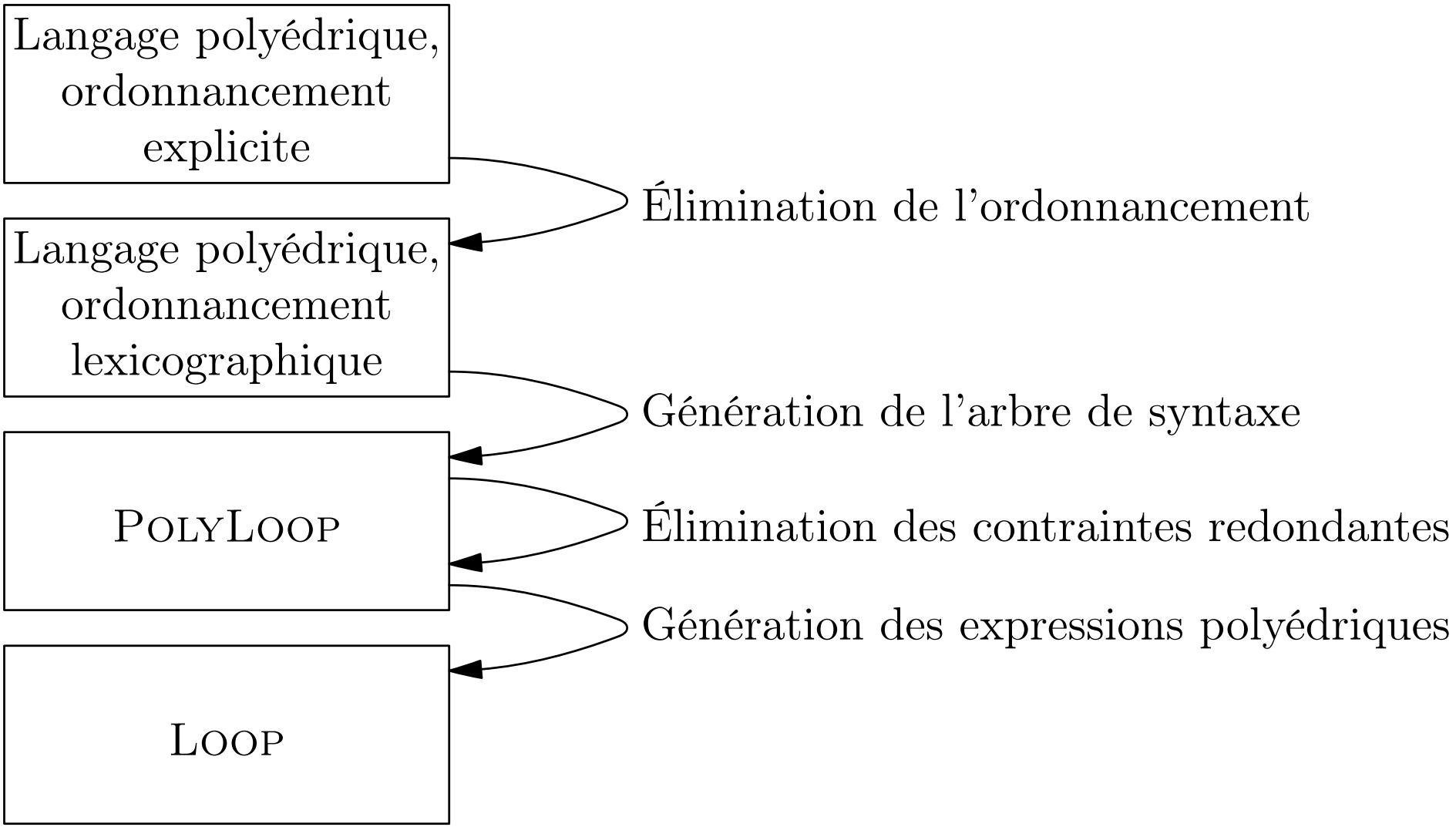
- Pour chaque instruction $(\mathtt{I}, \mathcal{D}, \theta, \mathcal{T})$, considérer les $x \in \mathcal{D}$ qui sont de la forme $\begin{pmatrix} \mathcal{E} \\ y \end{pmatrix}$
- Les trier par ordre lexicographique des $\theta(x)$
- Exécuter dans cet ordre les $\mathtt{I}(\mathcal{T}(x))$

Le modèle polyédrique

Génération de l'arbre de syntaxe

Formalisation en Coq

Étapes de génération



Langage intermédiaire

- Impératif, arbre de syntaxe abstrait
- Très haut niveau : tests et boucles sur polyèdres
- But : générer la forme du programme sans rentrer dans les détails

$$e ::= (x_1, \ldots, x_k) \mapsto \left\lfloor \left(\sum_{i=1}^{k} a_i x_i + c \right) \Big/ d \right\rfloor$$

$$\mathcal{P} ::= \left\{ (x_1, \ldots, x_k) \in \mathbb{Z}^k \Big/ A \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix} \leq a \right\}$$

$$s ::= \texttt{skip} \mid s_1; s_2 \mid \texttt{I}(e_1, \ldots, e_k) \mid \texttt{guard}\ \mathcal{P}\ s \mid \texttt{loop}\ \mathcal{P}\ s$$

Sémantique

$$\textsc{Skip}\ \frac{}{\mathcal{E} \vdash (\texttt{skip}, \mathcal{M}) \Downarrow \mathcal{M}}$$

$$\textsc{Seq}\ \frac{\mathcal{E} \vdash (s_1, \mathcal{M}_1) \Downarrow \mathcal{M}_2 \qquad \mathcal{E} \vdash (s_2, \mathcal{M}_2) \Downarrow \mathcal{M}_3}{\mathcal{E} \vdash (s_1; s_2, \mathcal{M}_1) \Downarrow \mathcal{M}_3}$$

$$\textsc{Instr}\ \frac{(\mathtt{I}(e_1(\mathcal{E}), \dots, e_k(\mathcal{E})), \mathcal{M}_1) \Downarrow_{\mathtt{I}} \mathcal{M}_2}{\mathcal{E} \vdash (\mathtt{I}(e_1, \dots, e_k), \mathcal{M}_1) \Downarrow \mathcal{M}_2}$$

$$\textsc{GuardTrue}\ \frac{\mathcal{E} \in \mathcal{P} \qquad \mathcal{E} \vdash (s, \mathcal{M}_1) \Downarrow \mathcal{M}_2}{\mathcal{E} \vdash (\texttt{guard}\ \mathcal{P}\ s, \mathcal{M}_1) \Downarrow \mathcal{M}_2}$$

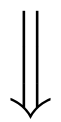
$$\textsc{GuardFalse}\ \frac{\mathcal{E} \notin \mathcal{P}}{\mathcal{E} \vdash (\texttt{guard}\ \mathcal{P}\ s, \mathcal{M}) \Downarrow \mathcal{M}}$$

$$\textsc{Loop}\ \frac{\forall x, \mathcal{E} :: x \in \mathcal{P} \Leftrightarrow a \leq x < b \qquad \forall x \in [\![a, b[\![, (\mathcal{E} :: x \vdash (s, \mathcal{M}_x) \Downarrow \mathcal{M}_{x+1})}{\mathcal{E} \vdash (\texttt{loop}\ \mathcal{P}\ s, \mathcal{M}_a) \Downarrow \mathcal{M}_b}$$

Cas d'une seule instruction polyédrique

- Projeter sur les d premières dimensions
- Créer un nœud `loop` correspondant au polyèdre obtenu
- Recommencer avec une dimension de plus

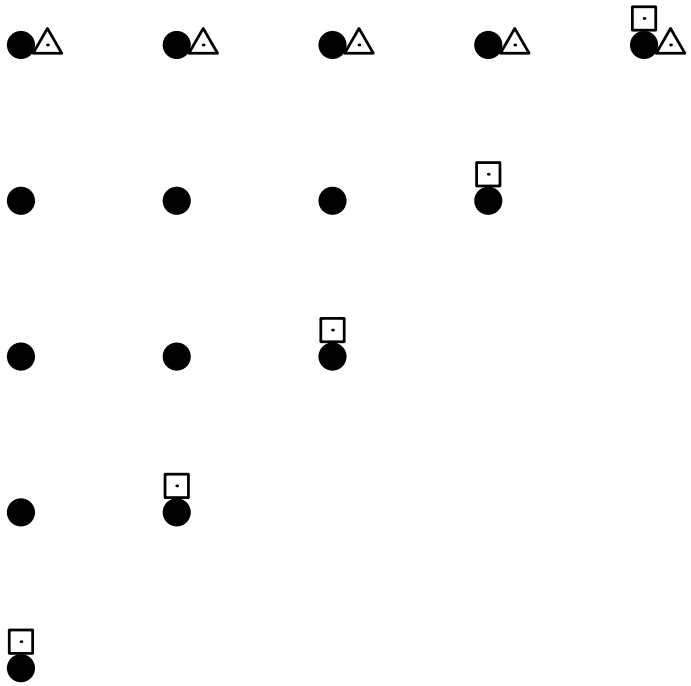
$$\mathtt{I}(\mathtt{m},\mathtt{k}) = \dots$$
$$\mathcal{D} = \{(\mathtt{n},\mathtt{m},\mathtt{k})/1 \leq \mathtt{k} < m < n\}$$
$$\mathcal{T}(\mathtt{n},\mathtt{m},\mathtt{k}) = (\mathtt{m},\mathtt{k})$$

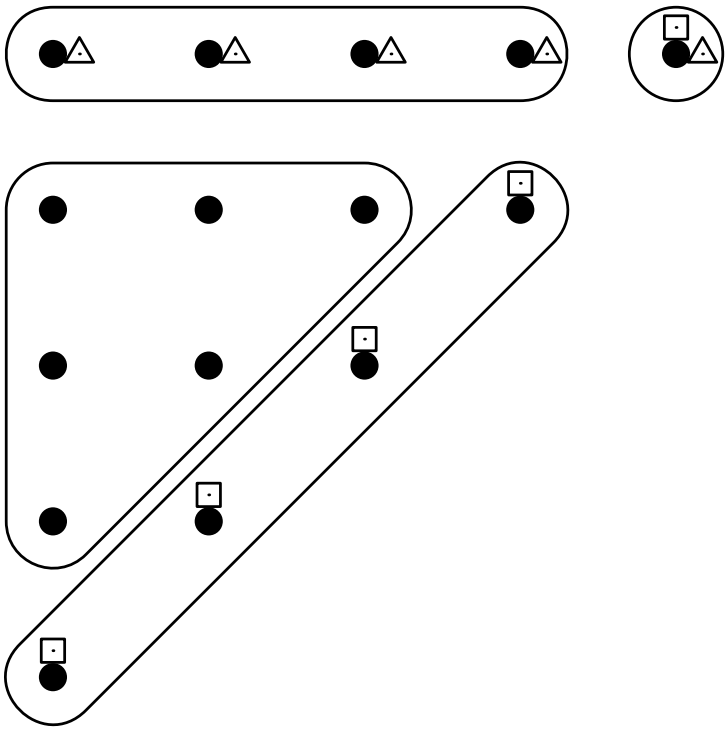


$$\texttt{loop}\ \{(\mathtt{n},\mathtt{m})/2 \leq \mathtt{m} < \mathtt{n}\}$$
$$\texttt{loop}\ \{(\mathtt{n},\mathtt{m},\mathtt{k})/1 \leq \mathtt{k} < \mathtt{m} < \mathtt{n}\}$$
$$\mathtt{I}(\mathtt{m},\mathtt{k})$$

Cas de plusieurs instructions polyédriques

- Projeter sur les d premières dimensions
- *Séparer les projections en une liste de polyèdres disjoints*
- *Ordonner les projections en respectant les dépendances*
- Créer un nœud `loop` pour chaque polyèdre obtenu
- Recommencer avec une dimension de plus





Élimination des contraintes redondantes

- Simplification d'une commande dans un polyèdre de contexte
- Récursion pour simplifier les boucles internes
- Élimine la plupart des contraintes ne faisant pas intervenir la variable de boucle

$$\texttt{loop}\ \{(\mathtt{n},\mathtt{m})/2 \le \mathtt{m} < \mathtt{n}\}\quad \texttt{loop}\ \{(\mathtt{n},\mathtt{m},\mathtt{k})/1 \le \mathtt{k} < \mathtt{m} < \mathtt{n}\}\quad \mathtt{I}(\mathtt{m},\mathtt{k}) \quad\Rightarrow\quad \texttt{loop}\ \{(\mathtt{n},\mathtt{m})/2 \le \mathtt{m} < \mathtt{n}\}\quad \texttt{loop}\ \{(\mathtt{n},\mathtt{m},\mathtt{k})/1 \le \mathtt{k} < \mathtt{m}\}\quad \mathtt{I}(\mathtt{m},\mathtt{k})$$

Langage cible

- Impératif, forme identique au langage intermédiaire
- Plus bas niveau
- Rend explicite les conditions de boucle

$$e ::= c \mid x \mid e_1 + e_2 \mid c \cdot e \mid \lfloor e/c \rfloor \mid e \texttt{ mod } c \mid \texttt{min}(e_1, e_2) \mid \texttt{max}(e_1, e_2)$$
$$t ::= e_1 = e_2 \mid e_1 \leq e_2 \mid t_1 \texttt{ \&\& } t_2 \mid t_1 \texttt{ || } t_2 \mid \texttt{!}t \mid \texttt{true} \mid \texttt{false}$$
$$s ::= \mid \texttt{skip} \mid s_1\texttt{;}\, s_2 \mid \texttt{I}(e_1, \ldots, e_k) \mid \texttt{if } (t) \; s$$
$$\mid \texttt{for } (i = e_1\texttt{;}\; i < e_2\texttt{;}\; i\texttt{++}) \; s$$

Sémantique

$$\textsc{Skip}\ \frac{}{\mathcal{E} \vdash (\texttt{skip}, \mathcal{M}) \Downarrow \mathcal{M}} \qquad \textsc{Seq}\ \frac{\mathcal{E} \vdash (s_1, \mathcal{M}_1) \Downarrow \mathcal{M}_2 \qquad \mathcal{E} \vdash (s_2, \mathcal{M}_2) \Downarrow \mathcal{M}_3}{\mathcal{E} \vdash (s_1\,\texttt{;}\,s_2, \mathcal{M}_1) \Downarrow \mathcal{M}_3}$$

$$\textsc{Instr}\ \frac{(\texttt{I}(\texttt{eval}(\mathcal{E}, e_1), \ldots, \texttt{eval}(\mathcal{E}, e_k)), \mathcal{M}_1) \Downarrow_{\texttt{I}} \mathcal{M}_2}{\mathcal{E} \vdash (\texttt{I}(e_1, \ldots, e_k), \mathcal{M}_1) \Downarrow \mathcal{M}_2}$$

$$\textsc{IfTrue}\ \frac{\texttt{eval}(\mathcal{E}, t) = \texttt{true} \qquad \mathcal{E} \vdash (s, \mathcal{M}_1) \Downarrow \mathcal{M}_2}{\mathcal{E} \vdash (\texttt{if}\ \ (t)\ \ s, \mathcal{M}_1) \Downarrow \mathcal{M}_2} \qquad \textsc{IfFalse}\ \frac{\texttt{eval}(\mathcal{E}, t) = \texttt{false}}{\mathcal{E} \vdash (\texttt{if}\ \ (t)\ \ s, \mathcal{M}) \Downarrow \mathcal{M}}$$

$$\textsc{For}\ \frac{\forall x \in [\![\texttt{eval}(\mathcal{E}, e_1), \texttt{eval}(\mathcal{E}, e_2)[\![, (\mathcal{E}, i : x \vdash (s, \mathcal{M}_x) \Downarrow \mathcal{M}_{x+1})}{\mathcal{E} \vdash (\texttt{for}\ \ (i\ =\ e_1\texttt{;}\ \ i\ <\ e_2\texttt{;}\ \ i\texttt{++})\ \ s, \mathcal{M}_{\texttt{eval}(\mathcal{E}, e_1)}) \Downarrow \mathcal{M}_{\texttt{eval}(\mathcal{E}, e_2)}}$$

Génération des contraintes de boucle

- Passage du langage intermédiaire au langage cible
- Contraintes indépendantes de la variable de boucle : nœud `if`
- Contraintes sur la variable de boucle : calcul des bornes inférieure/supérieure avec `min` et `max`, nœud `for`
- Peut échouer si pas de borne inférieure ou supérieure

```
loop {(n, m)/2 ≤ m < n}
  loop {(n, m, k)/1 ≤ k < m < n}
    I(m, k)
```

⇒

```
for (m = 2; m < n; m++)
  if (m < n)
    for (k = 1; k < m; k++)
      I(m, k)
```

Le modèle polyédrique

Génération de l'arbre de syntaxe

Formalisation en Coq

Formalisation en Coq

- Formalisation des sémantiques et des transformations
- Preuves de correction
- Utilisation de la VPL pour tester si un polyèdre est vide et canonisation de polyèdre
- Autres opérations polyédriques : code naïf + canonisation
- ≈ 8000 lignes de Coq
- Code Coq : `https://github.com/Ekdohibs/PolyGen`

Théorème principal

- Théorèmes pour chaque transformation
- Composition : si la génération réussit, la sémantique est respectée

```
Theorem complete_generate_many_preserve_sem :
  forall es n pis env mem1 mem2,
    (es ≤ n)%nat →
    WHEN st ← complete_generate_many es n pis THEN
    loop_semantics st env mem1 mem2 →
    length env = es →
    pis_have_dimension pis n →
    (forall pi, In pi pis → (poly_nrl pi.(pi_schedule) ≤ n)%nat) →
    env_poly_semantics (rev env) n pis mem1 mem2.
```

Conclusion

- Génération de code vérifiée pour le modèle polyédrique
- Fonctionne dans le cas de plusieurs polyèdres

Travaux futurs :

- Simplification du code produit
- Formalisation des autres étapes (en particulier le réordonnement)
- Utilisation de CompCert pour produire du code machine vérifié

Élimination de l'ordonnancement

- Boucles imbriquées : ordre lexicographique
- Idée : préfixer les $\theta(x)$ dans le domaine pour avoir un ordre lexicographique

$$\mathcal{D} = \{y / A \cdot y \leq a\} \qquad \mathcal{D}' = \left\{ \begin{pmatrix} u \\ y \end{pmatrix} \middle/ A \cdot y \leq a \wedge u = K \cdot y + k \right\}$$

$$\theta(y) = K \cdot y + k$$

$$\mathcal{T}(y) = T \cdot y + t \qquad \mathcal{T}' \begin{pmatrix} u \\ y \end{pmatrix} = T \cdot y + t$$

- Environnements à préfixer au début : légèrement plus complexe

$$\mathtt{I}(\mathtt{m},\mathtt{k}) = \dots$$
$$\mathcal{D} = \{(\mathtt{n},\mathtt{m},\mathtt{k})/1 \leq \mathtt{k} < \mathtt{m} < \mathtt{n}\}$$
$$\theta(\mathtt{n},\mathtt{m},\mathtt{k}) = (\mathtt{m},\mathtt{k})$$
$$\mathcal{T}(\mathtt{n},\mathtt{m},\mathtt{k}) = (\mathtt{m},\mathtt{k})$$

$$\Downarrow$$

$$\mathtt{I}(\mathtt{m},\mathtt{k}) = \dots$$
$$\mathcal{D} = \{(\mathtt{n},u,v,\mathtt{m},\mathtt{k})/1 \leq \mathtt{k} < \mathtt{m} < \mathtt{n} \wedge u = \mathtt{m} \wedge v = \mathtt{k}\}$$
$$\mathcal{T}(\mathtt{n},u,v,\mathtt{m},\mathtt{k}) = (\mathtt{m},\mathtt{k})$$