DJEMAI, Massyl

HADDED, Sawssen

Le compilateur Soufflé
Rapport d'éxposé
Programmation Comparée

1. Introduction

Le langage logique permet l'utilisation de nombreux outils logiques dont les clauses de Horn pour exprimer des faits, des règles ou des requêtes. Cela permet d'analyser ou de résoudre des problèmes de façon fiable. Il existe plusieurs langages logiques comme Prolog mais ici nous nous intéresserons à Datalog et plus particulièrement au compilateur Soufflé qui permet de générer un fichier binaire exécutable à partir d'un fichier écrit en Datalog.

2. Introduction a Datalog

Datalog est un langage logique, qui découle, historiquement, de Prolog.
Il est souvent présenté comme étant un sous-ensemble de ce dernier. On l'utilise régulièrement dans l'expression de requêtes sur des bases de données déductives.
L'intérêt de ce genre de requêtes est notamment l'utilisation de la récursion. Les requêtes SQL classiques sont limitées car elles ne permettent pas de connaître le nombre de jointures nécessaires pour explorer une structure de données arborescente.
L'énonciation de règles de déduction logique à travers un cas de base et un cas général, est possible à travers Datalog.
Un script Datalog, en utilisant les clauses de Horn, permet d'énoncer: des faits *(facts)*, des règles *(rules)* et des requêtes *(queries)*

- **Les faits** *(clauses de Horn dites positives)* sont les données d'entrée du programme, elles peuvent être renseignées dans le code source, mais également dans un fichier d'entrée.

  Il peut s'agir par exemple de fichiers CSV donnant les données traitées .

- **Les règles** *(clauses de Horn dites strictes)* sont les expressions logiques qui permettent de décrire des relations déduites à partir des inputs. Celles-ci sont fixes et ne changent pas selon un fichier d'entrée qu'on donnerait au programme.

- **Les requêtes** *(clauses de Horn dites négatives)* sont les requêtes qui décrivent ce que le programme est censé renvoyer à la suite de l'évaluation des règles logiques énoncées par le script.

Soufflé étend le langage datalog en permettant, par exemple, l'utilisation de foncteurs, ce qui fait que l'on peut exprimer de grands projets logiques.

3. Soufflé

3.1. Présentation

Soufflé (Systematic, Ontological, Undiscoveres, Fact, Finding, Logic, Engine) a été initialement conçu pour l'analyse statique de langage logique dirigé par Bernhard Scholz chez Oracle Labs.
Il a été mis en libre accès en mars 2016. Il permet d'exprimer de grands projets logiques en surmontant certaines limitations de Datalog par l'utilisation de foncteurs, enregistrements etc…
Il applique des techniques de compilation avancées pour les programmes logiques (projection de Futamura, évaluation partielle, etc…).
De ce fait, il permet de traduire efficacement les règles déclaratives en programmes C++ compatible sur des machines modernes.

3.2. Fonctionnement

Soufflé génère un fichier binaire exécutable permettant d'exécuter les règles que nous avons défini dans notre fichier Datalog sur des données qu'on lui fournira en entrée. Ci-dessous nous avons schématisé le fonctionnement de Soufflé :

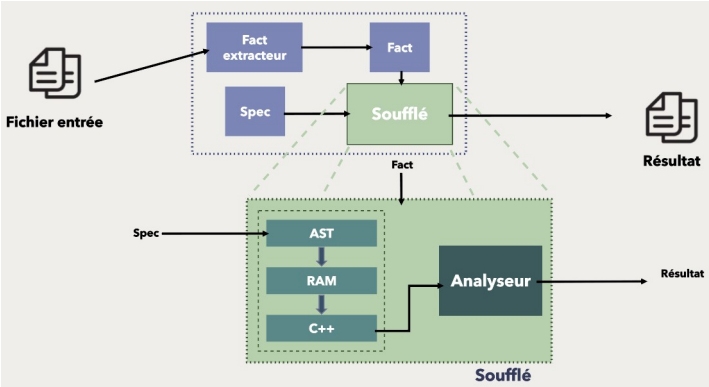

Figure 1 - Schéma du fonctionnement de Soufflé

On a, d'une part, une vue sur l'obtention du fichier binaire (Analyseur) et d'autre part, une vue sur l'exécution de notre programme.
Spec pour « specializer » est le résultat d'une projection de Futamura de notre code Datalog. La projection de Futamura est une technique d'optimisation.
Ce code est donné a soufflé ainsi que « fact » qui représente nos données sur lesquelles nous voulons exécuter notre « programme » datalog.
L'analyseur est le fichier binaire exécutable généré par la compilation de notre code (ici spec).

Notre code datalog est parser, on obtient un AST. Ici ont lieu des optimisations de haut niveau. Ensuite, il est traduit dans un langage intermédiaire qui se nomme RAM (Relational Algebra Machine).

Il transforme notre code déclaratif en un code impératif en ajoutant du contrôle de flot (boucle, if etc...). Ici ont lieu des optimisations de moyen niveau.
Pour finir, notre code est traduit en code C++ natif grâce à un générateur de code et est compilé avec le compilateur C++, qui nous donne notre fichier binaire.

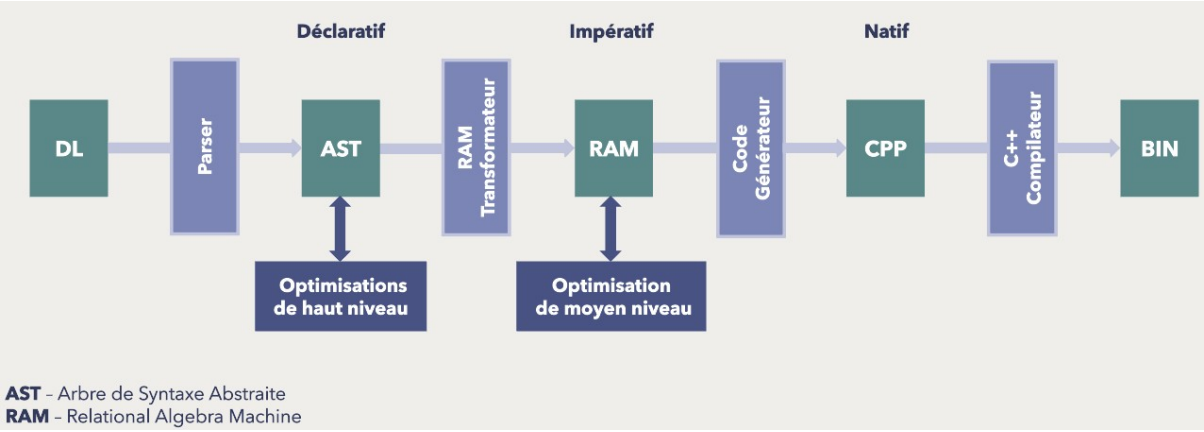

Figure 2 - Schéma des étapes de compilation de Soufflé

3.3. Exemple

Prenons ce code datalog :

```
1   .decl edge(x:number, y:number)
2   .input edge
3
4   .decl path(x:number, y:number)
5   .output path
6
7   path(x, y) :- edge(x, y).
8   path(x, y) :- path(x, z), edge(z, y).
```

Figure 3 – Code datalog de notre exemple

Ici on déclare edge comme étant une arête (ligne 1) et qui recuperera ces arêtes le fichier (l'« input ») se nommant « edge » (ligne 2). On déclare path, un chemin qu'il devra écrire dans le fichier (l'« output ») se nommant path (ligne 5) .
Ligne 7 et 8 on dit : « Un chemin x - y si une arête x – y existe », « Un chemin x - y si un chemin x – z et une arête z – y existent ».

On tape la première commande pour générer notre exécutable et la deuxième pour obtenir le fichier C++ généré. Les options -F et -D servent respectivement à indiquer le chemin du dossier d'entrée et de sortie.

```
./src/souffle -F./DOSSIER_INPUT -D./DOSSIER_OUTPUT ./fichier.dl

ou

./src/souffle -F./DOSSIER_INPUT -D./DOSSIER_OUTPUT -o./DOSSIER_CPP/NOM ./fichier.dl
```

Figure 4 – Commandes d'exécution de Soufflé

Voici l'entrée que nous avons passé a notre fichier binaire exécutable et le résultat obtenu :

```
sawssen@bloopi:~/Documents/souffle/build/DEMO$ cat input/edge.facts
1       2
2       3
sawssen@bloopi:~/Documents/souffle/build/DEMO$ cat output/path.csv
1       2
1       3
2       3
sawssen@bloopi:~/Documents/souffle/build/DEMO$
```

Figure 5 – Entrée et sortie de notre exemple

Pour finir, voici une partie du code traduit en C++, il s'agit des lignes 1 et 4.

```
1   .decl edge(x:number, y:number)
2   .input edge
3
4   .decl path(x:number, y:number)
5   .output path
6
7   path(x, y) :- edge(x, y).
8   path(x, y) :- path(x, z), edge(z, y).
```

```
public:
    Sf_example()
        : wrapper_rel_1_edge(0, *rel_1_edge, *this, "edge",
        std::array<const char *, 2>{{"i:number", "i:number"}},
        std::array<const char *, 2>{{"x", "y"}}, 0),
        wrapper_rel_2_path(1, *rel_2_path, *this, "path",
            std::array<const char *, 2>{{"i:number", "i:number"}},
            std::array<const char *, 2>{{"x", "y"}}, 0)
    {
        addRelation("edge", wrapper_rel_1_edge, true, false);
        addRelation("path", wrapper_rel_2_path, false, true);
    }
    ~Sf_example()
    {
    }
```

Figure 6 – Code datalog et sa traduction en C++ par Soufflé

4. Conclusion

Soufflé est un outil qui permet de traduire un code datalog en un code natif C++ et d'en générer un fichier binaire exécutable de façon optimale.

Ceci permet de faire de la vérification fiable, en supposant que la traduction du code source est toujours correct, puisqu'on part de règles énoncées a l'aide d'outils logique.

C'est pourquoi Soufflé est utilisé dans de nombreux projets notamment dans des projets de vérification de protocole de sécurité (AWS cloud d'Amazon). Soufflé a d'autres très nombreux cas d'usage que nous n'avons pas traité dans cet exposé afin de respecter le format, vous pouvez les retrouver facilement via nos références.

5. Références

Dépôt officiel
Page d'Oracle Labs
Documentation
The Choice Construct in the Soufflé Language
Engineering Static Analyzers with Soufflé