実装言語独立でモジュラリティーの良いコンパイラキット

– Emacs Lisp で実装されたマルチソース・マルチターゲットコンパイラ開発環境 –

1　背景

コンパイラキット (インフラストラクチャ) はユーザにとっての使いやすさが重要である。コンパイラの扱うデータは複雑であるから、ソースが公開されていて、ドキュメントが整備されているだけでは十分とは言えない。

例えば、現在最も広く使われているリターゲッタブルコンパイラである GCC は C 言語で実装された優れたオープンソフトであり、中間言語やリターゲッタブルコード生成 系のドキュメントも整っている強力なコンパイラであるが、いざ内部をいじろうとすると、複雑なデータ構造、グローバル変数の使用、メモリー管理の危険性などが障害となり、それは容易ではない。本来、GCC は強力で実用的なフリーのリターゲッタブルコンパイラを作成するのが目的であったから、コンパイラキットとしての視点、すなわち各部品のモジュラリティーの良さや簡潔さは考慮されていない。

もっとコンパイラ研究者にとって使いやすいコンパイラインフラを作ろうという反省から、COINS (COmpiler INfraStructure) プロジェクトが発足した。開発代表者も COINS プロジェクトに参加していた。COINS コンパイラはその点実装言語が Java になったことから改善されている。C 言語で実装されている gcc における複雑で危険なメモリー管理からは開放され、C 言語特有のトリッキーなポインタ操作も Java のオブジェクトレベルでの安全な操作で可能になっている。

一方で、我々は依然としてこれらのコンパイラキット（インフラ）には大きな問題点があると考える。それは我々のプロジェクトのタイトルにもある「実装言語からの独立性」である。モジュラリティーはどのようなソフトウエアでも重要であるが、特にソフトウエァの個々の構成部品を部分的に利用したり、あるいは差し替えたりする必要性があるコンパイラキットでは、モジュラリティーこそ使いやすさの最重要ポイントである。

2　目的

SCK では、データ構造というものを実装言語とは全く無関係な、独立した簡潔なプログラミング言語として定義し、さらに徹底的なモジュール分割を行うことで、実装言語から独立したコンパイラ部品を提供する。実際の実装言語は Emacs Lisp であるが、Emacs Lisp の知識がなくても SCK を利用することが出来る。これが実装言語独立の意味である。

さらに、Emacs Lisp は記号処理向きの柔軟なプログラミング言語であり、SCK を Emacs 上で利用するユーザには、強力で快適なプログラミング環境が提供される。また、マルチソースに対応するために、LALR(1) に基づくパーサジェネレータを提供し、コンパイラで多用されるグラフ構造を美しく表示するための graphviz (ATT) への Emacs Lisp インタフェースも提供する。

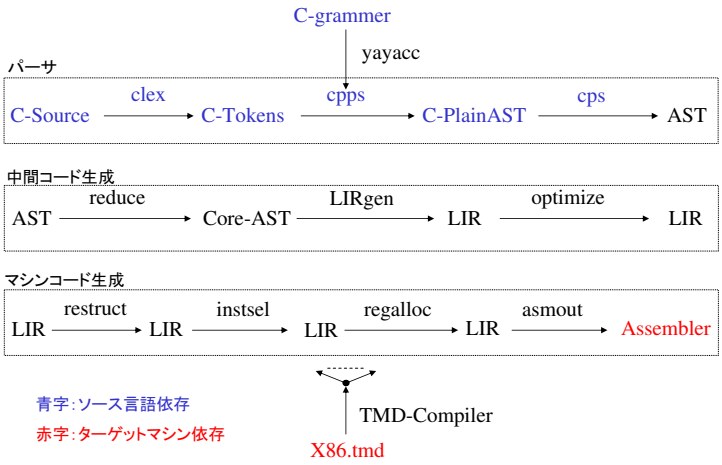
3　開発の内容

開発したものは以下の通りであり、全体で役 3 万行の Emacs Lisp コードである。

1. 実装言語独立のコンパイラモジュール群 (構成図参照)
2. C 言語のフロントエンド
3. x86 のマシン記述、バックエンド
4. グラフ可視化ツール Graphviz (ATT) への Emacs Lisp インタフェス
5. LALR(1) に基づいたパーサジェネレータ yayacc
6. Emacs Lisp 演習問題集

以下に SCK コンパイラの構成を説明する。

SCK 構成図



ソース言語が C 言語、ターゲットマシンが X86 の場合を例として、SCK コンパイラの全体構成を説明する。まず C 言語のソース C-Source が clex により字句解析をされ、トークンのリスト C-Tokens に変換される。SCK の C コンパイラではプリプロセッサは提供せずに既成のものを利用するので、C-Source は正確にはプリプロセッサを通した C 言語のソースである。

トークンのリスト C-Tokens は cpps (C Plain Parser) により C-PlainAST に変換される。各部については後に詳しく説明するが、PlainAST は純粋に構文的な表現としての AST (Abstract Syntax Tree) であり、cpps は構文的なチェック以外、例えば変数が宣言されているとか、式の型が合っているとかいったチェックは一切行わない。

C-PlainAST は C に依存した意味的なチェックを行う cps により、型の付いた AST (これを我々は単に AST と呼んでいる) に変換される。次に AST から型情報のみを

残し、他の全てのシンタックスシュガー的な要素 (ネストした副作用、制御構造等) をより基本的な要素に置き換えたものが Core-AST である。

Core-AST はまだ高級言語としての型を保持しているが、それらをマシンレベルの型に落とすのが LIRgen である。LIR は後のフェーズで共通に使われる実装言語、ターゲットマシン独立の中間言語である。これは開発者代表が COINS の低水準中間言語として設計したものとほぼ同じものを利用している。

下段はリターゲッタブルコンパイラの最重要部分、マシンコード生成部である。これらのフェーズの各部品は、部分的にマシン記述ファイル、この場合は X86.tmd から TMD コンパイラによって生成されたコードを取り入れることで、ターゲットマシンに特化された処理を行う。

ターゲットマシン独立の最適化が終了した LIR に対して行う最初のターゲットマシン依存の変換は restruct である。ここでは LIR のレベルで抽象的表現されてるが、実際のコード生成では具体化する必要のあるものをターゲットマシン依存のコードに変換する。そして、命令選択 instsel がマシン記述ファイルで定義されている X86 に実在する命令のみでプログラムが実行出来るように LIR を変換する。

次にレジスタアロケータ regalloc により LIR の仮想レジスタが実レジスタに置き換えられる。この段階の LIR は事実上 X86 の命令と一対一に対応する命令の列に変換されているが、シンタックスは LIR のままである。そして最後に asmout が LIR をターゲットアセンブラの表記に変換して、コンパイルフェーズが終了する。

4 従来の技術との相違

SCK の特徴は大きく二つに分けられる。それはモジュラリティーの良さと開発環境の良さである。前者はコンパイラの部品を実装言語と独立な S-式ベースの単純なインタフェースを持つモジュールに分割することで、コンパイラ開発者、研究者にとって使いやすい部品を提供している。以下に主要なモジュールのモジュラリティー (独立性) をまとめておく。

我々が知る限り、このように実装言語から、さらに部分的には中間言語からも独立なモジュールで構成されているコンパイラ開発支援ツールは SCK 以外存在しない。

	実装言語独立	C 言語独立	LIR 独立
yayacc-lir	Yes	No	Yes
cpps	Yes	No	Yes
cps	Yes	No	Yes
reduce	Yes	yes	Yes
tmd	No	Yes	No
loop	Yes	Yes	Yes
live	Yes	Yes	Yes
regalloc	Yes	Yes	Yes

一方、後者は Emacs Lisp という実装言語により SCK を実装したことにより、Lisp のもつ強力さ、ラピッドプロトタイピングのしやすさに加え、Emacs のインタラクティブな環境を利用出来るという点である。実際、今までの例で見てきたように Emacs のエディタコマンドとして手軽に個々の部品を実行し、テストすることが出来るという特徴は他に類を見ない。

5 期待される効果

コンパイラという分野は重要で面白く、奥が深く、人を引き付ける魅力に満ちた分野 (であったハズ) である。一方で、近年情報系学部の必修科目からも外されるという憂き目に会っている。にもかかわらず、新たなコンパイラの必要性は依然としてある。特に近年容易に新たなハードを容易に作成することが可能となり、その需要は組み込み系の分野等、限定されはいるにせよ、むしろ高まりつつあると言える。

使いやすく内部構造が分かりやすいコンパイラキットの必要性は明らかである。SCK は新たなコンパイラを開発する場合にも威力を発揮するし、また個々のコンパイルフェーズが行う処理を理解することも容易になるためコンパイラ教育の教材としても優れている。付属の　graphviz へのインタフェースによる可視化ツールは、デバッグのみならず、コンパイラの教材を作る上でも、また、コンパイラの授業をアクティブで楽しいものにする上でも、重要な価値がある。

6 普及の見通し

生まれて間もない SCK ではあるが、デモを見た人は例外なく「面白い」という感想を漏らしてくれる。今後は学会発表等で積極的にユーザを増やしていく予定である。現在、大学 (東京大学、農工大学) の授業に使用してもらう予定がある。今後は書籍の出版も考えている。Emacs Lisp と絡めて、簡潔だが中身が濃く、かつ面白いコンパイラのテキストを作りたいと考えている。

7 開発者名

- 阿部正佳 (フリープログラマ)
- 山崎淳 (ミラクルアーツ、執行役員)
- 三津原敏 (ミラクルアーツ、代表取締役社長)
- SCK ホームページ `http://hq.os.cs.tuat.ac.jp/sck/`