AIST

Coq 用 C コード生成器の線形性検査拡張

田中 哲 , Reynald Affeldt, Jacques Garrigue

2018-08-31

日本ソフトウェア科学会第 35 回大会

目的

Coq から効率的な C プログラムを生成する

- Coq により正しさを検証できる
- 手書きの C と同等の効率で動作する

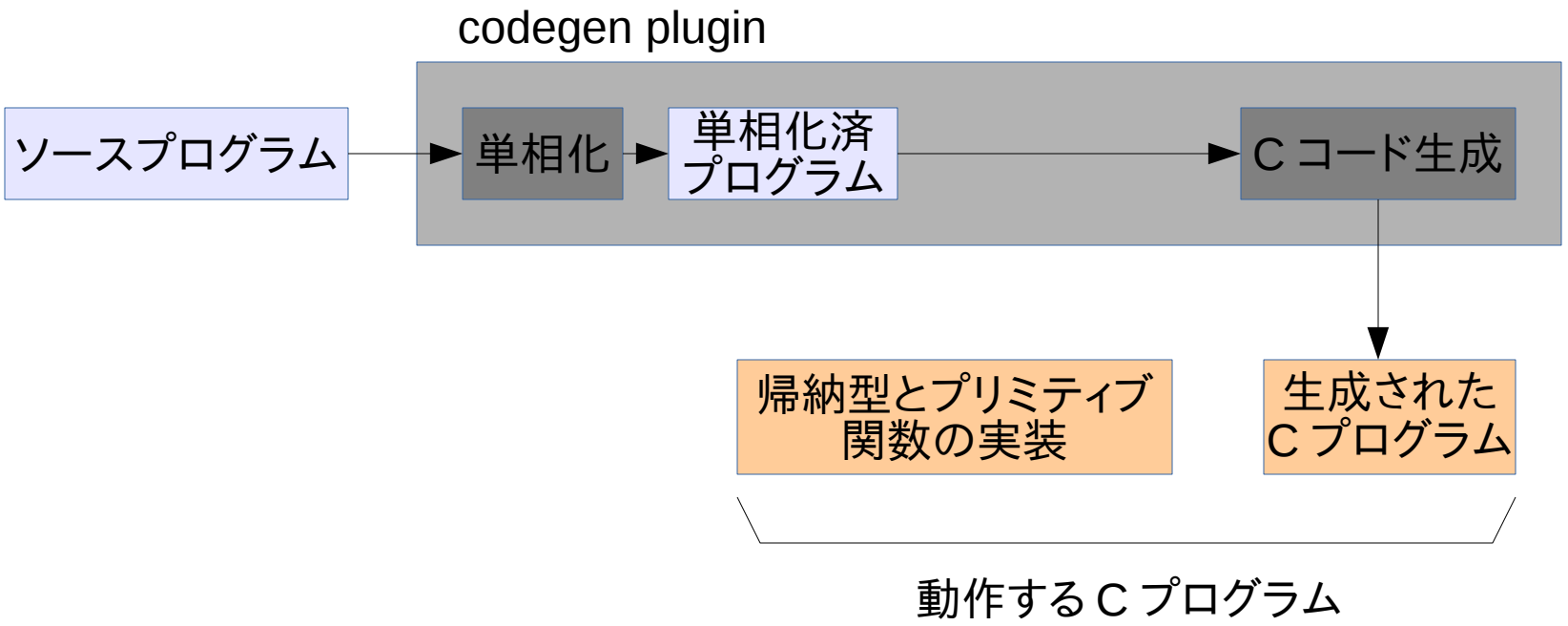
背景

- 我々は Gallina から C に変換する Coq plugin を開発している
  - Gallina は Coq 内部で使われる ML に似た言語
- Gallina は任意の評価順序で評価できる
  正格評価にすると、順序実行、変数束縛、関数呼び出しなど、C とかなり共通部分がある
- 共通部分を素直に変換するだけなので、オーバーヘッドも基本的になく効率がよい
- ただし、Gallina には C の破壊的変更に対応するものはない

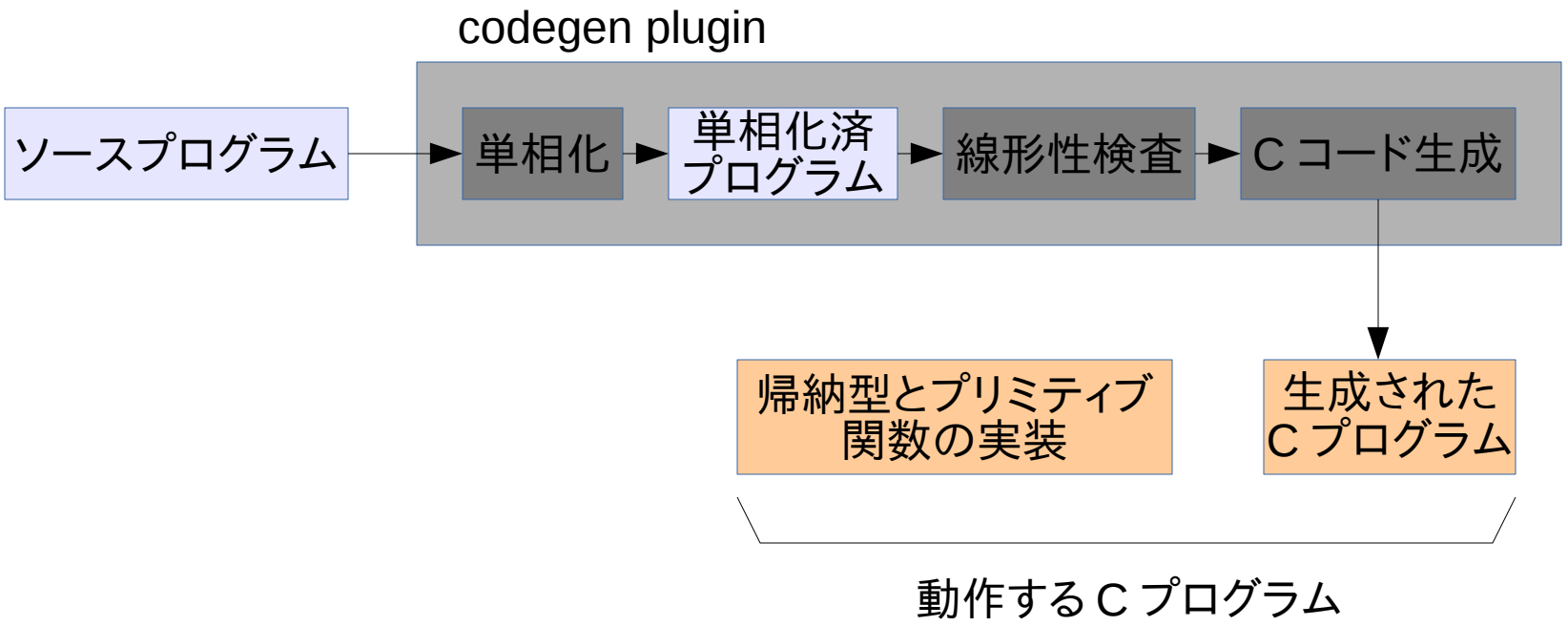
codegen plugin

- https://github.com/akr/codegen
- Gallina から C への素直な変換器
- まず convertible な範囲で可能な限り C 言語に近づける
  - 単相化、 A-normal form
  - 実際に convertible になっているかどうかは簡単に検査可能
    (reflexivity で証明できる)
- C 言語に変換するとき、データ表現を自由に変えられる
  - 個々の帰納型について最適な表現を選べる
    とくに、タグビットや、すべての値が 1 ワードといった制約はない
  - Gallina の値の一部しか扱えない表現を使ってもよい
    (nat を uint64_t に置き換えるなど)
    その場合、それでも問題が起きないことを証明する方法を用意している
    (monadification plugin)
- 単相化できないもの、高階関数など、素直に扱えないものはとりあえずあきらめる

コード生成の流れ


codegen plugin
ソースプログラム
単相化
単相化済
プログラム
C コード生成
帰納型とプリミティブ
関数の実装
生成された
C プログラム
動作する C プログラム

コード生成の流れ (線形性検査付)



動機： 破壊的変更を扱いたい

- 破壊的なデータ構造
  - C で扱うバッファを扱いたい（主要な動機）
  - 他にもさまざまな破壊的なデータ構造がある
- 入出力も破壊的な処理の一種と考えられる

いままでどうしていたか

- 破壊的なデータ構造に非破壊的な API をつけて使っていた
  - 非破壊的な帰納的な型として見えるなら、内部で破壊的な操作を行っていても問題ない
- 残念ながらオーバーヘッドがある
  - 破壊前の情報を保存しなければならない
  - バッファについては非破壊的な API をつけるのはそれほど難しくない
  - でも一般には非破壊的な API の実現は簡単でない

今回どうしたか

- C 言語への変換で線形性検査を実現した
  - 特定の型の変数が線形に使われることを検査する
- 線形に使われることが保証されるので、破壊的なデータ構造をそのまま使える
  - 破壊前のデータにアクセスされないことが保証される
  - 参照透明性は壊れない

具体例：バッファ

- 要素の並び
- 空バッファで始まる
- 最後に要素が追加されていく
- C 言語では単なる連続したメモリ領域
- 単純なので速い
- 本発表では以下を想定する
  - 簡単のため要素は bool 固定（主な用途は簡潔データ構造）
  - 必要に応じてサイズが拡大する
  - 既存の要素が変化することはない

Coq におけるバッファ API

- バッファ型 bits
- 作成
  - allocbits: unit → bits
- 状態問い合わせ
  - numbits: bits → nat
  - nthbit: bits → nat → bool
- 要素追加：純粋関数型なので新しい bits を返す
  - addbit: bits → bool → bits

バッファの Gallina 実装

- バッファ型：seq (SSReflect のリスト) で実装
  - Inductive bits : Type := buf of seq bool.
- 作成
  - Definition allocbits 'tt := buf [::].
- 状態問い合わせ
  - Definition numbits '(buf s) := size s.
  - Definition nthbit '(buf s) i := nth false s i.
- 要素追加
  - Definition addbit '(buf s) v := buf (rcons s v).

検証可能性と C 言語バッファ実装

- Gallina によるバッファ実装により、Gallina でバッファを使うプログラムを記述できる
- そのようなプログラムは Coq で検証できる
- プログラムを C 言語に変換する際、バッファの実装を C 言語による実装で置き換えて高速に実行できる
- ただし、置き換え先の addbit も非破壊的に動作しなければならない

バッファの素直な破壊的実装

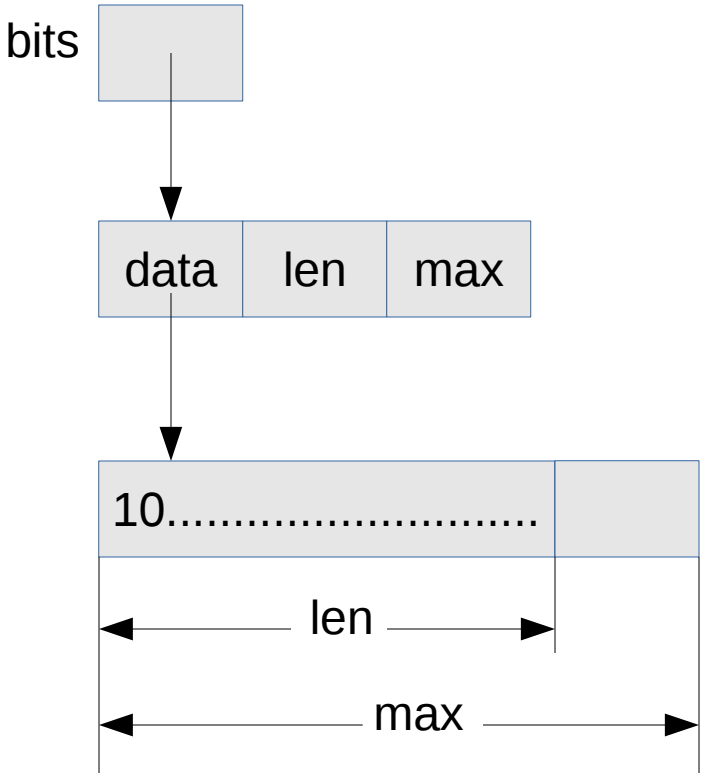
- typedef struct {
  uint64_t *data;
  nat len; /* current length [bit] */
  nat max; /* maximum length [bit] */
  } *bits;
- 実際の要素列に加えて現在の長さと確保したメモリサイズを保持する
- 要素追加は、空きがあればそこに書き込み、len をインクリメントする
- 空きがなければ data を realloc する
- 非破壊的な addbit を提供できない

バッファの非破壊的実装

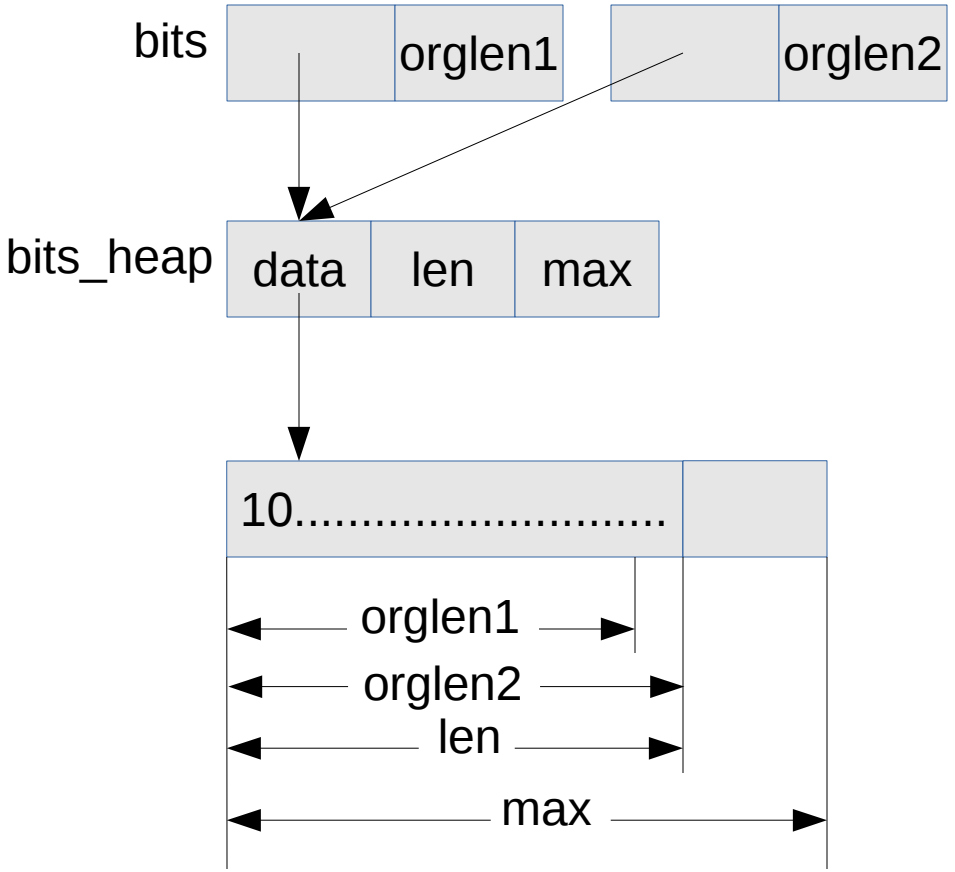
- bits 型に元々の長さ orglen を保持しておく
- ```
  typedef struct {
    uint64_t *data;
    nat len; /* current length [bit] */
    nat max; /* maximum length [bit] */
  } bits_heap;
  typedef struct {
    bits_heap *heap;
    nat orglen; /* original length [bit] */
  } bits;
  ```
- 破壊前の情報はすべて残っているので、非破壊的 addbit を実装できる
- バッファがリストのように分岐する場合にはデータを複製するコストが発生するが、バッファを使うのはそのような分岐が不要な場合なので問題ない
  ```

破壊的バッファと非破壊的バッファ

破壊的バッファ



非破壊的バッファ



バッファ実装の得失

破壊的バッファの利点

- C で普通に利用するバッファと同じ形
- orglen がないので効率がよい

非破壊的バッファの利点

- Gallina による定義と動作が一致するので置き換えても安全
  (証明した性質が成り立つことが期待できる)

破壊的バッファと線形性検査

- 手書きの C プログラムと同じ効率にしたい
  → 効率的な破壊的バッファを Gallina から安全に利用したい
- Gallina からの利用のしかたを制限すればよい
- このためにはバッファ型の変数は必ず線形に使われることを確認すればよい
- バッファ型が線形にしか使われないなら、破壊的バッファをそのまま Gallina から使っても問題ない (Coq で証明した性質が壊れない)

線形性

- 変数が必ず１回使われるという性質
- 変数を１回使うと使えなくなってしまうので古い値を参照できない
- 古い値を参照できないので、使ったときに破壊的変更を行っても参照透明性が壊れない

linear と unrestricted

- 線形性検査を行う型を linear な型と呼ぶ
  そうでない型を unrestricted な型と呼ぶ
- どの型について線形性を検査するかユーザが指定する
  - 破壊的バッファを使う場合、バッファ型 bits を指定
- linear な型をフィールドに持つ帰納型は linear
- それ以外はすべて unrestricted
  とくに、関数型はすべて unrestricted

線形性検査を行うコード生成

- linear な型の宣言
  CodeGen Linear bits.
- C コード生成
  GenC 関数名 .
- 線形性検査に成功する例
  Definition appb2 (b : bits) (v : bool) :=
    addbit (addbit b v) v.
- 線形性検査に失敗する例
  Definition invalid (b : bits) (v : bool) :=
    (b, addbit b v).

線形性検査アルゴリズム

- 基本アイデア
  - 変数の出現回数を数える
  - 関数型は unrestricted なので、関数抽象の内部での外側の変数の参照は禁止
    - なお、 curry 化された関数は C 言語に変換する時に uncurry するので、カスケードした関数抽象はひとまとまりとして扱う
  - C コードを生成する対象についてだけ検査を行う
    証明などそれ以外では線形でなくてよい
- 詳細は論文を参照

ベンチマーク

- ひとつのバッファに 100000000 回 addbit で要素を追加した
- 非破壊的バッファに対し、破壊的バッファは約 1.3 倍の速度向上となった
- orglen の引数受け渡しや len との比較がなくなったため速くなっている
  - x86_64 では非破壊的 bits 型の引数はレジスタで受け渡される
  - ベンチマークプログラムを gcc -O2 でコンパイルした結果では関数内部でもレジスタに配置されていた

まとめ

- Coq の C コード生成プラグインに線形性検査を実装した
- 破壊的データ構造を直接利用できるようになり、手書き同様な高速なプログラムを実現できるようになった