メタ計算を用いた Continuation based C の検証手法

比嘉 健太 並列信頼研

プログラミング言語と信頼性

・信頼性の高いソフトウェアを提供することは重要である
・ソフトウェアが要求される仕様を満たすかどうか検証する
・モデル検査的アプローチと定理証明的アプローチの2つがある
  - モデル検査的アプローチ: プログラムの状態を数え上げ
    仕様に背く状態が無いか確認する
  - 定理証明的アプローチ: プログラムの正しさを直接証明する
・検証しやすい言語 Continuation based C (CbC)を開発している
・CbC では両アプローチによる検証が可能なことを示す

Continuation based C (CbC)

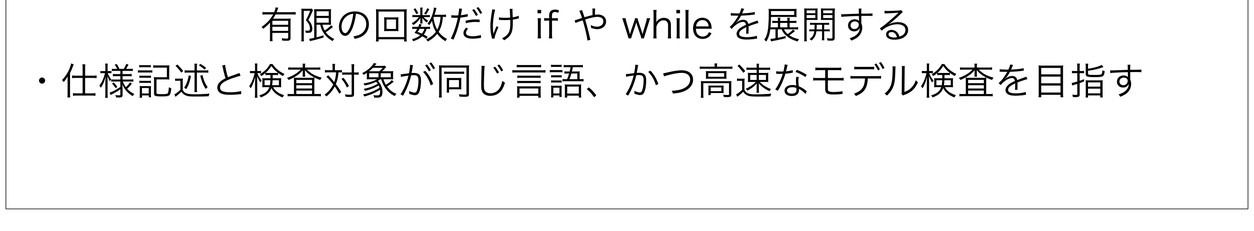
・アセンブラとC言語の中間のような言語で、構文はほとんど C 言語
・OS や組み込みソフトウェアなどが対象
・CodeSegment と DataSegment という単位でプログラミング
・CodeSegment を接続することでプログラムを構成する
・メタ計算の切り替えにより検証や並列実行、例外処理を行なう

```
__code cs0(int a, int b) {
  goto cs1(a+b);
}
__code cs1(int c) {
  goto cs2(c);
}
```

・CbC のプログラム例
・cs0 と cs1 が CodeSegment
・a と b の数値を加算する cs0
・引数部分が DataSegment
・goto が CodeSegment の接続

CbC とメタ計算

・メタ計算とはとある計算を支える計算
・ネットワーク処理、例外処理、並列実行など
・CbC は通常レベルの計算とメタ計算を分離して考える
  - 通常レベルではポインタは出てこない、など
・ CodeSegment の接続部分に処理を追加することで拡張する
  - メタ計算をする CodeSegment は Meta CodeSegment
  - メタ計算に必要な DataSegment は Meta DataSegment

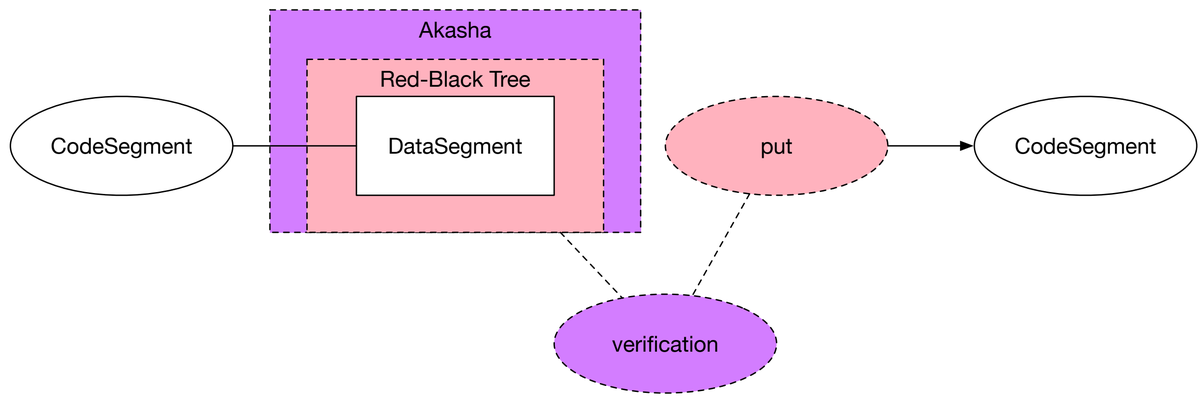


モデル検査

・ソフトウェアが仕様を満たすかチェックする
・仕様に背く状態があれば反例としてその実行状態を提出
・既存のモデル検査器
  - spin : promela と呼ばれる言語で記述。
    仕様記述は実行可能な式。例えば (x < 10) など
    並列動作を検証可能で実行可能な C ソースを生成可能
    しかし C と promela は記述がかなり異なる
  - CBMC : C/C++ を記号実行可能。 こちらも仕様記述は式。
    記号実行により実行経路を列挙する
    有限の回数だけ if や while を展開する
・仕様記述と検査対象が同じ言語、かつ高速なモデル検査を目指す

メタ計算ライブラリ akasha

・CbC に対するモデル検査的アプローチ
・CodeSegment の接続部分をメタ計算として定義
・網羅的に実行するよう接続部分を上書きすることで状態を列挙



・非破壊赤黒木の挿入操作に関する仕様を検証
・要素数13までは木がバランスすることを保証
・恣意的にバグを仕込むと仕様に背く状態を返却
・CBMC ではバグに由来した反例を検出できず

```
if (context->data[AkashaInfo]->akashaInfo.maxHeight >
    2*context->data[AkashaInfo]->akashaInfo.minHeight)
```

定理証明とプログラム

・Curry-Howard Isomorphism により証明とプログラムの型は対応
・論理式は型に相当し、証明はその型を持つ値の導出
・Coq、Agda といった強力な型を持つ言語では証明が記述可能
・三段論法の自然演繹による証明木は以下のようになる
  - 三段論法: ((A ならば B) かつ (B ならば C)) ならば (A ならば C)

$$\dfrac{\dfrac{[A]\ {\scriptstyle (1)} \qquad \dfrac{[(A \Rightarrow B) \wedge (B \Rightarrow C)]\ {\scriptstyle (2)}}{(A \Rightarrow B)} \wedge 1\mathcal{E}}{B} \Rightarrow \mathcal{E} \qquad \dfrac{[(A \Rightarrow B) \wedge (B \Rightarrow C)]\ {\scriptstyle (2)}}{(B \Rightarrow C)} \wedge 2\mathcal{E}}{\dfrac{C}{A \Rightarrow C} \Rightarrow \mathcal{I}_{(1)}} \Rightarrow \mathcal{E}}{((A \Rightarrow B) \wedge (B \Rightarrow C)) \Rightarrow (A \Rightarrow C)} \Rightarrow \mathcal{I}_{(2)}$$

・三段論法の Agda による証明は以下のようになる

```
f : {A B C : Set} -> ((A -> B) × (B -> C)) -> (A -> C)
f = \p x -> (snd p) ((fst p) x)
```

Agda と Continuation based C

・CbC で CbC 自身を証明したいが現状ではできない
・証明支援系 Agda 上で CbC を記述することで形式的な定義を得る
・DataSegemnt はレコード型となり、CodeSegment は関数型となる
・メタ計算は部分型を用いることで定義可能

```
record ds0 : Set where
  field
    a : Int
    b : Int

cs0 : CodeSegment ds0 ds1
cs0 = cs (\d -> goto cs1 (record {c = (ds0.a d) + (ds0.b d)}))
```

・Agda に CbC を変換することで証明が行なえる
・SingleLinkedStack に対する操作の性質を証明した
・「あるスタックに対してn回だけ値を積んだ後、
  同じ回数だけ値を取り出すと元のスタックに等しい」
・Agda で性質を定義すると以下のようになる

```
n-push-pop-type n cn ce st = M.exec (M.csComp (n-pop  n) (n-push n)) m ≡ m
-- goto (pop*n . push*n) mds ≡ mds
```