表 1　高速化施策と lib/Prelude.hs コンパイル時間 (-prof 追加以前)

項目	Esprimo (-O2)	Conoha (-O0)
高速化実施前	4.872s (100%)	38.143s (100%)
1. Map から Map.Strict	5.133s (95%)	36.724s (104%)
2. [Assump]	4.640s (105%)	32.095s (119%)
3. Subst	ー （逆効果だった）	ー
4. Codegen Text	ー （逆効果だった）	ー
5. CodeGen !	4.495s (108%)	27.844s (137%)
-prof 追加 (Esprimo のみ)	7.292s	ー

093: 単純な高速化の実施

↑ up

- issued: 2020-05-24
- 分類： 分類：C 改善項目
- status: Closed (2020-05-24)

概要

073 の対策追加により実行時間が著しくのびてしまい、環境によっては開発に支障をきたすレベルになってしまった (Conoha 上ではむちゃくちゃ時間がかかる上に、4-palla ではメモリ不足で落ちる)。そのため、プログラムの構造に影響を与えないような（一部を単純におきかえることで実現できる）高速化をいくつか実施したい。

結果サマリー

今回実施した各施策の効果を以下の表にまとめた。時間は以下を 3 回測定したものの最良値。

```
time ./no-prelude-compile.sh lib/Prelude.hs
```

これらの、コンパイル単体の（若干の）高速化にくわえて、現状は、make check などで 100 以上のテストを実施する際に、毎回 lib/Prelude.hs をコンパイルしなおしているので、これを make check のたびに一回だけ実施するようにした。

表 2　make check の高速化 (lib/Prelude.hs を 1 回だけコンパイル)

項目	Esprimo (-O2)	Conoha (-O0)
高速化実施前	8m28s	測定不能
lib/Prelude.hs 1 回のみに	4m47s	26m (1 つ fail)
sample152.hs 分割	4m58s (ここから -prof 追加)	10m5s

調査ログ

2020-05-24 (Sun)

高速化の候補と効果の実測

まだ高速化に着手するつもりはなかったのですが、いかにも効きそうで、かつ、プログラムが複雑になるなど副作用のないものに限って適用することにする。候補は以下：

1. Data.Map を使っている個所を Data.Map.Strict に置き換える (BindingGroup, CodeGen)
2. [Assump] を使っている Assump 辞書を Data.Map.Strict による実装に置き換える
3. Subst を Data.Map.Strict で実装
4. CodeGen で、出力の蓄積を String から Data.Text に変える
5. CodeGen における Gen State の各フィールドに！をつける

これらを順に実施していき、効果を確認していく。効果測定には、lib/Prelude.hs のコンパイル時間を以下のコマンドで測定する。

```
time ./no-prelude-compile.sh lib/Prelude.hs
```

結果を以下に示す。3 回測ったうちの最良値を測定結果とする。

`[Assump]` は、lookup 対象になる部分を Data.Map.Strict による実装におきかえた（ConstructorInfo の中身は [Assump] のまま)。一部、apply の際に fromList して toList で戻すという非効率が気になるが、トータルでは儲かっているようだ。

Subst の方は、lookup の高速化よりも、@@ のたびに toList して、処理して fromList というデメリットの方がおおきいらしく、逆効果だったので不採用に。

CodeGen では、出力文字列を String.(++) でどんどん連結しているのがいかにも非効率にみえたが、一部だけ Text にしたのでは逆効果だった。OverloadedString も用いて、Text のみで押すようにすれば速くなるかな。

make check の効率化

lib/Prelude.hs のコンパイルにこれだけ時間がかかっているのだから、make check で何度も何度も無駄にコンパイルしているのを省くと大幅に儲かるはず（わかっていたけど)。

そこで、make check の最初に一回コンパイルしたものを、全テストプログラムで共有するように、スクリ

表 3　高速化施策と lib/Prelude.hs コンパイル時間 (-prof 追加以前)

項目	Esprimo (-O2)	Conoha (-O0)
高速化実施前	5.287s	40.778s
	4.872s (100%)	46.225s
	4.885s	38.143s (100%)
1. Map から Map.Strict	5.128s	47.711s
	5.137s	36.724s (104%)
	5.133s (95%)	37.361s
2. [Assump]	4.735s	32.095s (119%)
	4.670s	33.549s
	4.640s (105%)	32.436s
3. Subst	5.998s	ー
	5.946s	
	5.940s	
4. Codegen Text	4.885s	55.468s
	4.838s	
	4.822s	
4b. Codegen Text.Lazy	ー	57.415
5. CodeGen !	4.518s	27.844s (137%)
	4.503s	28.262s
	4.495s (108%)	

プトを修正。結果はサマリに示した通り。

プロファイルの取得

これ以上の高速化は、めくらではなくプロファイルに基づいてやりたい。Esprimo 上では OFLAG 環境変数に `-rtsopts -prof -fprof-auto -O2` を設定するようにして、プロファイルをとれるようにした。実行時に `+RTS -p -RTS` をコマンドオプションとして与えればよい。

これによって、Esprimo 上での実行時間は 7.3m 程度まで悪化した。

-prof オプションをつけるには、ライブラリも profile 版をインストールする必要があった。`apt install ghc-prof` で base をインストールしなおしたあと、`cabal install -p hoge --reinstall --force-reinst` でいくつかのライブラリを置き換える必要があった。

sample152.hs の分割

現状では、sample152.hs のコンパイルに突出して時間がかかっている。そこで、これを 5 個に分割した (sample152[a-f].hs)

この程度の長さのプログラムがコンパイルできないようでは困るので、さらなる高速化は必須ですが、make check に無用な負荷を書ける必要はないため。

つぎの高速化の方針について

sample152.hs を少しだけ短くした sample152mod.hs でプロファイルを採取、@@ (68%), apply (11%) で大半の時間を消費しているらしい。

Subst をこれらに最適化した実装にかえる価値はありそう。（Data.Map.String は apply 向きではあるが、@@ は遅くなる）

本 issue は一旦ここで区切りとする（クローズ）