Zadanie domowe 7

Kolorowanie grafów

W tym zadaniu zajmiemy się zaimplementowaniem algorytmu Kempego-Chaitina kolorowania grafu w alokatorze rejestrów.

Załóżcie nowy katalog **lab7** i skopiujcie do niego całą zawartość katalogu **lab6**. Następnie ściągnijcie plik

`www.math.us.edu.pl/~pgladki/teaching/2013-2014/tk_lab7.zip`

i rozpakujcie go; w szczególności znajdziecie tam pliki regalloc.sml, color.sig oraz color.sml.

- Zaimplementujcie funkcję `verify` w color.sml.
- Zaimplementujcie funkcję `color` w color.sml.
- Zdebugujcie poprzez uruchomienie `Compile.compile "whatever.fun"` Plik wyjściowy `whatever.fun` nie będzie tak do końca działającym kodem MIPS-a, wiele z tymczasowych rejestrów zostanie alokowanych do prawdziwych rejestrów, ale wszystkie będą "$x" pseudorejestrami.
- Prześlijcie mailem pliki README oraz color.sml. Termin zadania mija w **czwartek, 30 stycznia.** Proszę pamiętać o oznaczeniu maila tagiem `[aghtk]`.

Weryfikacja

Wasza funkcja weryfikacyjna ma spełniać następujące warunki:

- `alloc(t)` musi przyjmować wartości w palecie dozwolonych kolorów zdefiniowanej w `palette`;
- każdy wirtualny (czyli nie pokolorowany wcześniej) rejestr wzmiankowany w `func` jest albo w `alloc` albo w `spills`;
- żaden z wcześniej pokolorowanych rejestrów nie znajduje się w `alloc` lub w `spills`;
- żadne dwa rejestry, które interferują (wszystko jedno, pokolorowane wcześniej, czy nie) nie mogą mieć tego samego koloru – "kolor" wirtualnego rejestru jest w `alloc`, kolor pokolorowanego wcześniej rejestru to on sam;
- żaden z rejestrów "spilled" nie ma `spillCost >= spillConstInfinity`.

Wygodny sposób, aby osiągnąć to, co powyżej, to zdefiniować funkcje `"mention"` oraz `"interfere"`, a następnie wywołać je przez `Liveness.analyze`.

W razie wykrycia konfliktów, po prostu wywołajcie `ErrorMsg.impossible` z odpowiednim komunikatem – im więcej informacji o konflikcie znajdzie się w komunikacie, tym lepiej!

Kolorowanie

Zaimplementujcie kolorowanie grafu **bez** cieniowania, tj. kompletnie zignorujcie parametr `moves` funkcji `color`.

Wykorzystując moduł `RegSet` możecie naprawdę prosto poradzić sobie ze zbiorami rejestrów. Oznacza to, że możecie uzyskać elegancką i naturalną implementację algorytmu opisanego w Section 11.1 podręcznika.

Jądrem Waszego algorytmu będzie funkcja wyglądająca mniej więcej tak:

```
fun f (lowdegs: RS.set, highdegs: RS.set) : coloring
```

gdzie `lowdegs` jest zbiorem wcześniej nie pokolorowanych rejestrów wciąż pozostających w grafie interferencji, które mają stopień $\leq k$, natomiast `highdegs` jest zbiorem wcześniej nie pokolorowanych rejestrów wciąż pozostających w grafie interferencji, które mają stopień $> k$, natomiast k oznacza liczbę kolorów w palecie `palette`.

Jeśli `lowdegs` jest niepusty, to zastosujcie heurystykę `Simplify` opisaną na stronie 229 podręcznika z drobną modyfikacją: zamiast dorzucań nowe rejestry na stos, lepiej wywołajcie rekursywnie funkcję `f`.

Jeśli zarówno `lowdegs` jak i `highdegs` są niepuste, zastosujcie heurystykę `Spill` opisaną na stronie 229 podręcznika z najniższym możliwym kosztem. Znowu, postarajcie się zastosować wywołanie rekursywne, zamiast nadużywać stosu.

Pamiętajcie o tym, że `Simplify` oraz `Spill` zmieniają stopień istniejących węzłów podczas wywołania rekursywnego.

Jeśli obydwa zbiory są puste, wróćcie z funkcji `f` – wówczas kolory zostaną przypisane w drodze wyjściowej z najgłębszej pętli rekursywnej.