IB015 – Domácí úkol 12: emulátor výpočetního stroje

Termín: 4. prosince 23.59; do odevzdávárny, 5 pokusů.

V poslední velké úloze z neimperativního programování si vyzkoušíme, jak se ve funkcionálním paradigmatu udržují stavové informace napříč výpočtem. Budeme interpretovat instrukce jednoduchého procesoru a vytvoříme emulátor výpočetního stroje.

Kostru řešení s definicemi datových typů naleznete ve studijních materiálech předmětu.

Model stroje

Náš jednoduchý stroj sestává z:

- procesoru s několika registry,
- pracovní paměti,
- programu, tj. posloupnosti instrukcí, a
- výstupního proudu, do něhož během výpočtu stroj zapisuje výsledky.

Pásky

Důležitým stavebním prvkem našeho systému je paměťová páska se čtecí hlavou. Páska je rozdělena na políčka a pod čtecí hlavou se vždy nachází jedno z nich – tzv. *zaměřené*. Pouze zaměřené políčko je možné přímo číst a upravovat. Je-li potřeba pracovat s daty na jiném místě pásky, je potřeba pásku *převinout* a tím zaměřit jiné políčko.

```
data Tape a = Tape Int [a] [a]
```

V representaci typem `Tape a` považujeme první seznam za obsah pásky před čtecí hlavou (nižší indexy), první prvek druhého seznamu je aktivní políčko pod čtecí hlavou a zbytek druhého seznamu se nachází na vyšších indexech za čtecí hlavou. První seznam je jistým způsobem v opačném pořadí: jeho první prvek je políčko těsně před zaměřeným, druhý je políčko dva před zaměřeným atd. Číselný parametr uchovává index zaměřeného políčka.

Náš virtuální stroj obsahuje takové pásky dvě:

- **Programová páska** (typ `Program`) obsahuje vykonávané instrukce.
- **Pracovní páska** (typ `Data`) se používá jako hlavní pracovní paměť.

Obě jsou naplněny (od indexu nula výše) před spuštěním stroje: programová páska neměnnými instrukcemi, pracovní páska počátečními daty (typicky argumenty pro vykonávaný program).

Zvláštností pracovní pásky je, že je oboustranně nekonečná[1]: libovolný (i záporný) index je platný. Výchozí hodnotou je 0. Programová páska sice být nekonečná nemusí, protože program je konečný, ale při implementaci se bude hodit ji za nekonečnou považovat. Vhodná výchozí hodnota vyplyne z dalších požadavků na stroj.

Registry

Náš procesor sestává z několika celočíselných registrů, z nichž některé mají speciální význam:

- Čtyři všeobecné registry: `RA`, `RB`, `RC` a `RD`.
- Registr `PC` (*program counter*) obsahující index právě vykonávané instrukce na programové pásce.
- Registr `MI` (*memory index*) obsahující index zaměřeného políčka pracovní pásky.
- Virtuální registr `M` (*memory*), který je svázán se zaměřeným políčkem pracovní pásky.

Všechny registry kromě `M` jsou zpočátku nastaveny na nulu. Hodnota `M` samozřejmě závisí na počátečních datech předaných stroji při spuštění. Hodnoty všeobecných registrů jsou uloženy v typu `Regs` (což je jen alias pro čtveřici hodnot) v pořadí `RA`, `RB`, `RC`, `RD`.

Všechny registry je možné použít v aritmetických operacích, a to jak jako operandy, tak jako cíl, kam se zapíše výsledek. Sémantika práce se speciálními registry je dle očekávání:

- Čtení z `PC` a `MI` umožňuje zjistit index zaměřeného políčka příslušné pásky.
- Zápisem do `PC` dojde k nepodmíněnému skoku a další běh stroje probíhá od instrukce na nově zapsaném indexu.

[1] Ve skutečnosti je sice omezena indexací typem `Int`, ale to je z důvodu usnadnění implementace. Koncepčně nic nebrání indexovat neomezeným `Integer`em.

- Zápisem do `MI` se převíjí pracovní páska, aby bylo zaměřeno příslušné políčko.
- Čtením a zápisem pseudoregistru `M` se čte a zapisuje do zaměřeného políčka pracovní paměti.

Navíc platí, že kdekoli se může jako operand vyskytnout registr, může se také objevit číselná konstanta (`Imm`, *immediate*). Zápis do konstanty nemá žádný efekt. Všechny druhy operandů jsou representovány datovým typem `Operand`:

```
data Operand  = RA | RB | RC | RD | MI | M | PC | Imm Value
type Value = Int
```

Podmínkové příznaky

Kromě registrů si stroj udržuje ještě dvě stavové hodnoty, jejichž význam bude ozřejmen v sekci o instrukcích. Ukládají se v datovém typu `TestFlags` a jsou jimi:

- *znaménko* (typ `SignumFlag`): záporné, kladné, nebo nula; a
- *parita* (dělitelnost dvěma, typ `ParityFlag`): sudá, nebo lichá.

Zpočátku tyto příznaky odpovídají vlastnostem nuly, tj. `(ZeroFlag, EvenFlag)`.

Výstupní proud

Výstupem ze stroje je proud čísel representovaný seznamem. Ten může být i nekonečný; strojem je tak možno například generovat matematické řady. Výstupní proud není vnitřním stavem stroje – hodnoty odeslané na výstup nijak nemohou ovlivnit další výpočet.

Provádění instrukcí

Stroj vykonává instrukce v paměti programu od první (index 0). Obvykle po každé provedené instrukci pokračuje instrukcí následující, tj. zvýší `PC` o jedna. Výjimkami jsou skoky (instrukce zapisující do `PC`), podmíněné instrukce (které mohou `PC` zvýšit o dva) a instrukce zastavení, která `PC` nezvyšuje.

Jelikož je velké množství stavu stroje mapováno do speciálních registrů, může být instrukční sada velmi jednoduchá. Většina zajímavých akcí se děje přes tříregistrové sčítání. Instrukce jsou representovány datovým typem `Instruction` a jsou popsány níže.

Aritmetické instrukce

Základní instrukcí je `Add r a b`, která do operandu `r` uloží součet operandů `a` a `b`, vše typu `Operand`. Protože operandy a výsledkem mohou být i speciální registry či konstanta, nahrazuje tato instrukce spoustu jiných, které můžete z různých procesorů znát:

- Kopie mezi registry: `Add RC RA (Imm 0)`
- Absolutní skok na adresu v `RA`: `Add PC RA (Imm 0)`
- Relativní skok o pět instrukcí dále: `Add PC PC (Imm 5)`
- Zaměření třetího políčka od bázového registru `RB`: `Add MI RB (Imm 3)`
- Načtení z paměti do `RA`: `Add RA M (Imm 0)`
- Zvýšení hodnoty v paměti: `Add M M (Imm 1)`
- Zdvojnásobení hodnoty: `Add RD RD RD`

Další dvě aritmetické instrukce jsou unární, a ačkoli je také možné je použít se speciálními registry či konstantami, příliš užitečné to není.

- `Negate r a`: do `r` uloží opačnou hodnotu operandu `a` (tedy změna znaménka).
- `Halve r a`: do `r` uloží polovinu hodnoty operandu `a` zaokrouhlujíc dolů.

Podmíněné instrukce

- `Test r`: nastaví podmínkové příznaky (znaménko a paritu) ve stroji tak, aby odpovídaly vlastnostem hodnoty operandu `r` – tj. zachycovaly jeho zápornost/kladnost/nulovost a sudost/lichost.
- `If p`: Pokud predikát `p` platí, neděje se nic zvláštního a pokračuje se další instrukcí. Pokud neplatí, stroj přeskočí provádění následující instrukce a pokračuje až tou další.

  Predikátem (typ `Condition`) je dotaz na hodnotu jednoho z podmínkových příznaků: `Zero`, `Pos`, `Neg`, `Even` nebo `Odd`.

Příklad: sekvence instrukcí `Test RB`, `If Zero`, `Add PC M (Imm 0)` znamená skok na číslo instrukce v paměti (např. návrat z funkce), pokud je hodnota `RB` nulová.

Instrukce `Test` a `If` nemusí následovat těsně za sebou. Příznaky zůstávají nezměněny až do dalšího `Test`u a mohou tudíž ovlivňovat více `If`ů.

Výstupní instrukce

- `Out r` předá na výstup stroje hodnotu operandu `r`.
- `Halt` ukončí výstup a zastaví vykonávání stroje.

Okrajové stavy

- **Skok na neplatnou instrukci** (tj. mimo paměť na instrukce) není chybou a provede se. Protože ale už není dále co vykonávat, **v dalším kroku** je práce stroje ukončena stejně, jako to dělá instrukce `Halt`.
- Pracovní paměť je oboustranně nekonečná a žádné „mimo“ nemá. Všechna doposud nezměněná políčka nesou hodnotu 0.
- **Zápis do konstanty** (např. `Negate (Imm 0) RA`) není chybou, ale prázdnou operací – instrukce se provede, ale nemá žádný efekt kromě posunutí `PC` na další instrukci.
- **Po skoku** (zápise do `PC`) se už programová páska **nepřevíjí** o jedno políčko vpřed jako u jiných instrukcí. Tudíž `Add PC PC (Imm 0)` vede k zacyklení stroje a `Add PC PC (Imm 1)` je zbytečné stejným způsobem jako zápis do konstanty.

Funkce k implementaci

Na rozehřátí implementujte sadu funkcí na práci s páskami:

- `fromList :: a -> [a] -> Tape a`: vyrobí oboustranně nekonečnou pásku naplněnou výchozí hodnotou `a` (od indexu 0 výše) počátečními daty. Je zaměřeno políčko s indexem 0. Například tedy:

  ```
  fromList x [a, b, c, d]  ~>* Tape 0 [x, x, ...] [a, b, c, d, x, x, ...]
  ```

- `tapeIndex :: Tape a -> Int`: vrátí index zaměřeného políčka.
- `readTape :: Tape a -> a`: vrátí hodnotu zaměřeného políčka.
- `writeTape :: a -> Tape a -> Tape a`: změní hodnotu zaměřeného políčka.
- `advanceTape :: Tape a -> Tape a`: posune pásku o jedno políčko vpřed.
- `seekTape :: Int -> Tape a -> Tape a`: převine pásku na políčko se zadaným indexem.

Následně napište pomocné funkce na čtení a zápis operandů:

- `getValue :: Operand -> Machine -> Value`
- `setValue :: Operand -> Value -> Machine -> Machine`

Úplné definice typů a typových aliasů naleznete v kostře. Rámcově zde uveďme, že `Value` je číslo, `Operand` registr (i speciální) nebo konstanta a `Machine` obsahuje celý vnitřní stav stroje.

S využitím pomocných funkcí navrhněte interpret instrukcí, který provede pouze jeden krok výpočtu a vrátí nový stav stroje. „Vnější“ hodnota `Nothing` znamená, že stroj má zastavit,[2] „vnitřní“ značí, že stroj v kroku nevyprodukoval žádný výstup.

- `evalStep :: Machine -> Maybe (Maybe Value, Machine)`

Konečně s využitím této funkce vytvořte emulátor stroje, který bude provádět zadaný program se zadanými počátečními daty a výstupní seznam bude líně plnit výsledky instrukcí `Out`.

- `eval :: [Instruction] -> [Value] -> [Value]`

Požadavek na lenost zde znamená, že i kdyby výpočet nikdy neskončil, průběžné výsledky bude možné z výstupního seznamu číst (např. pomocí funkce `take`).

[2] To jest, že vykonal instrukci `Halt` nebo nemá co provádět, protože `PC` už ukazoval mimo paměť pro program. Například instrukce `Negate PC PC` se ještě vykoná a funkce vrátí nový stav stroje (se záporným `PC`). Až v dalším kroku je výsledkem `Nothing`, protože na záporném indexu už není co dalšího vykonávat.

Tip: Vyhledejte si funkci `catMaybes` z balíku `base`. A pokud patříte mezi ty, kteří se nezaleknou typu funkce `unfoldr`, můžete `eval` napsat velice úsporně.

Poznámky a tipy

- Importovat smíte moduly z balíku `base`.[3]
- Nevynalézejte znovu kolo! Hledejte mezi knihovními funkcemi fakultní instancí Hoogle.
- Řiďte se tipy k psaní dobrého kódu ze Sbírky! Zejména neduplikujte kód, ale tvořte abstrakce.
- Nejste-li si jisti nějakou částí zadání, zeptejte se v diskusním fóru.
- Nezapomeňte, že **opisování je zakázáno** a bude postihováno podle disciplinárního řádu.
- Přebíráte-li kód odjinud, uveďte zdroj, jinak bude na vaši práci pohlíženo jako na plagiát.
- Než řešení odevzdáte, **pečlivě si přečtěte následující sekci** a ujistěte se, že váš kód splňuje všechny náležitosti. Neztrácejte body jen kvůli nepozornému čtení pokynů.

Odevzdání a hodnocení

Celkově můžete za úlohu získat **až 2,5 bodu** rozdělených podle implementovaných funkcí:

Funkcionalita	Body
jednoduché operace s `Tape`	0,1
`seekTape`	0,1
`getValue` a `setValue`	0,5
`evalStep`	1,3
`eval` končících programů	0,2
`eval` nekončících programů	0,3

Úloha je hodnocena pouze automatickými testy. Snažte se však přesto psát úhledný a dobře dekomponovaný kód – daleko snáze se vám bude ladit.

Odevzdání

Tento domácí úkol se neodevzdává přes odpovědník, nýbrž přes příslušnou odevzdávárnu v Informačním systému.

- Do odevzdávárny vkládejte **jediný** soubor s příponou `.hs` obsahující vaši implementaci **všech** požadovaných funkcí.
  - Pokud chcete odevzdat znovu, můžete soubor přepsat či přidat nový, nezáleží na tom – vyhodnocuje se vždy nejnovější odevzdaný soubor.
  - Žádné jiné soubory nevkládejte.
  - Již vložené soubory nepřejmenovávejte, mohou se pak vyhodnotit dvakrát.
- Vaše řešení **musí zachovat všechny definice datových typů tak, jak jsou v zadání**, jediná povolená modifikace je přidání instance typové třídy.
- Řešení musí jít přeložit překladačem GHC 9.2.1. Je možné, že na svých počítačích máte starší verzi, což by nemělo vadit, jde-li o verzi 8.4 a vyšší. I přesto vám doporučujeme, abyste si před odevzdáním svůj kód zkusili zkompilovat a spustit na Aise (nezapomeňte přidat modul s novým GHC), kde máte k dispozici GHC ve verzi 9.2.1. V případě selhání testů už při kompilaci nepřijdete o žádný pokus.
- **Všechny globální funkce musí mít typovou signaturu**.
- V odevzdaném souboru neuvádějte hlavičku `module` (pokud nevíte, o co se jedná, vůbec to nevadí).

Vyhodnocování

- Vyhodnocení po nahrání souboru **není** okamžité.
  - Automatický testovací nástroj kontroluje soubory v odevzdávárně v pravidelných intervalech několika minut. Podle času odevzdání a vytížení vyhodnocovacího serveru může vyhodnocení trvat několik desítek minut.
  - Pokud se vám výsledky neobjevily cca do půl hodiny a neblíží se zatím čas deadline, dejte nám vědět ve fóru.

[3] Výjimku tvoří moduly, které jsou „`Unsafe`“, ty však určitě nebudete potřebovat.

- Po vyhodnocení se získané body a případný výpis testů, které na vašem řešení selhaly, **objeví v poznámkovém bloku**.
  - U nesprávně implementovaných funkcí se dozvíte příklad vstupu, na němž se váš výsledek neshoduje s očekávaným.
- Máte **pět možností odevzdání**, započítává se nejlepší z nich.
  - Pokud při odevzdání neprojde kontrola syntaxe, tak se do tohoto limitu nepočítá. Pokud však kontrola projde, je automaticky započítáno a nelze to zvrátit.
- Další odevzdání provedete tak, že do odevzdávárny nahrajete novou verzi.
- Vzhledem k prodlevám při vyhodnocování neodkládejte práci na poslední chvíli, ať možnost vícenásobného odevzdání v případě potřeby vůbec stihnete využít.
  - S blížícím se termínem uzavření odevzdáváren očekávejte větší (i několikahodinové) prodlevy.
  - Není žádná garance rychlosti vyhodnocování (může se tedy stát, že výsledky řešení odevzdaného v neděli ve 21.00 neuvidíte do konce deadline).
  - Na jakákoli odevzdání po termínu nebude brán zřetel.

S odevzdávárnou zacházejte s rozvahou, abyste nepřišli o možnosti odevzdání. I když nahrajete nové řešení ještě před zveřejněním výsledku v poznámkovém bloku, vyhodnocovací nástroj už může mít (a pravděpodobně má) vaše dřívější odevzdání ve frontě. Z jeho pohledu tak došlo ke dvěma odevzdáním a vy si vyplýtváte jeden pokus. Podobně se vám mohou započítat odevzdání navíc, pokud do odevzdávárny omylem vložíte více než jeden soubor nebo pokud soubor přejmenujete (každý soubor se vezme jako samostatné odevzdání).