Dokumentace úlohy MKA: Minimalizace konečného automatu v Pythonu 3 do IPP 2016/2017
Jméno a příjmení: Michal Mrnuštík
Login: xmrnus01

Struktura projektu

Úkol projektu se dal dekomponovat do 5 logických celků.
Nejprve bylo potřeba zkontrolovat správnost argumentů, poté načíst konečný automat obsažený na vstupu, následně provést kontrolu, zda se jedná o dobře specifikovaný konečný automat a poté provést akce zadané v argumentech a zapsat jejich výstupy do zadaného výstupu.

Zpracování argumentů

Pro argumenty byla vytvořena třída `Arguments`, jejíž instance následně ponese informace o argumentech za běhu aplikace. Pro zpracování argumentů se zavolá funkce `parse_arguments()`, ve které se použije modul `argparse` k načtení argumentů z příkazové řádky. Objekt získaný za pomoci `argparse` se pak následně namapuje na instanci objektu `Arguments`, který je poté skriptu vrácen.

Načtení konečného automatu

Návrh objektů pro uložení automatu

Nejprve si musíme navrhnout objektový systém pro uložení všech informací o konečném automatu.
Konečný automat je definovaný jako uspořádané pětice, kde na prvním místě je množina možných stavů, na druhém místě množina symbolů vstupní abecedy, na třetím místě množina pravidel přechodů mezi stavy, na čtvrtém místě počáteční stav automatu a jako poslední množina finálních stavů.
Z toho je jasné, že musíme dohromady vytvořit objekty pro uložení informací o stavu, symbolu vstupní abecedy, pravidlu a konečném automatu jako celku.
V projektu tedy máme třídu `State`, která obsahuje jméno stavu, třídu `AlphabetCharacter`, která reprezentuje symbol vstupní abecedy, třídu `Rule`, která obsahuje informaci o přechodu mezi stavy (současný stav, vstupní symbol, následující stav). Následně jsme vytvořili třídu `FiniteStateMachine`, která obsahuje veškeré informace o konečném automatu (množinu stavů, množinu pravidel, vstupní abecedu, počáteční stav a množinu konečných stavů).

Lexikální a syntaktická analýza

Lexikální a sémantická kontrola probíhá za pomoci konečného automatu (schéma rozdělené na jednotlivé bloky můžete nalézt zde: http://www.stud.fit.vutbr.cz/~xmrnus01/IPP_FSM.zip), jehož přijímaným jazykem je jazyk popisující konečný automat v nám zadaném formátu. Automat prochází vstupní text po znacích a na základě stavu ve kterém se nachází kontroluje, zda je v daném stavu takový znak očekávaný.
Automat také při načtení celých informací potřebných pro jejich vytvoření postupně vytváří instance `State`, `AlphabetCharacter` a `Rule`. Tyto instance se pak zavoláním speciálních metod objektu `FiniteStateMachine` (`add_state`, `add_rule`, `add_alphabet_character`, `add_final_state` a `set_start`) přidávají do tohoto objektu.
Jediná lexikální kontrola, která se provádí mimo konečný automat, je provedena v konstruktoru `State`, kde se pomocí regulárního výrazu (`[a-zA-Z][_a-zA-Z0-9]*[a-zA-Z0-9]$` pro identifikátory delší než 1 a `[a-zA-Z]` pro jednoznakové identifikátory) kontroluje zda se jedná o validní identifikátor stavu.

Sémantická analýza

Veškeré sémantické kontroly (existence všech stavů a znaků abecedy, které se používají) se poté provádí při volání těchto metod.

Kontrola vstupního automatu

Před prováděním jakýchkoliv dalších akcí je nutné zkontrolovat, jestli námi načtený automat je dobře specifikovaný konečný automat. Objekt `FiniteStateMachine` obsahuje metodu `is_well_specified`, která veškeré tyto kontroly provede. Nejprve se zavoláním metody `find_unreachable` najdou nedostupné stavy, které by neměly existovat (metoda vrátí `None`). Poté se zkontroluje, že pro každý stav existuje právě jedno pravidlo pro každý symbol vstupní abecedy (cyklus volající metodu `__find_from_rules`). Následně se vyhledají všechny neukončující stavy (metoda `find_non_finishing`), jejich počet musí být menší nebo rovno 1. Pokud byly tyto podmínky splněny automat je dobře specifikovaný.

Provedení akce

Nalezení neukončujícího stavu

Vyhledávání neukončujícího stavu provádíme za pomoci následujícího algoritmu:

1. Vytvořme zásobník (v Pythonu možno použít list) a vložme na něj všechny konečné stavy. Vytvořme seznam `visited` do nějž budeme ukládat už navštívené stavy.
2. Pokud je zásobník prázdný ukončeme prohledávání a jdeme na krok 5.
3. Vezměme první stav ze zásobníku (`stack.pop()`) a najděme všechny pravidla, která mají tento stav jako cílový (metoda `__find_to_rules`). Vložme tento stav do seznamu `visited` (`stack.append`).
4. Projděme všechna pravidla a jejich výchozí stavy, pokud už nejsou v seznamu `visited`, vložme na zásobník. Vraťme se na krok 2.
5. Porovnejme seznam `visited` se seznamem stavů automatu. Stavy, které jsou ve `visited` ale ne v seznamu stavů, jsou neukončující.

Minimalizace

Minimalizace se provede zavoláním metody `minimize` nad objektem `FiniteStateMachine`, která vrátí novou instanci automatu odpovídající minimalizovanému konečnému automatu. Pro potřeby minimalizace jsem vytvořili speciální třídu `GroupState` jejíž instance budou reprezentovat skupinu stavů vzniklou při minimalizaci. Minimalizace probíhá pomocí následujícího algoritmu:

1. Vytvořme skupinu z finálních stavů a vložme ji do seznamu skupin. Následně vytvořme druhou skupinu z ostatních stavů a vložme ji také na seznam.
2. Pokud došlo k rozdělení nebo jde o první průchod pokračujme dál. Jinak jdeme na krok 5.
3. Projděme všechny skupiny v seznamu.
4. Najděme všechny možné přechody ze stavů ve skupině pro jeden symbol abecedy (metoda `__find_from_states`). Pokud se všechny koncové stavy přechodů nachází v jedné skupině nelze provést rozdělení (pokračujme dalším symbolem nebo další skupinou). V jiném případě rozdělme stavy do odpovídajících skupin a vraťme se na krok 2.
5. Vytvořme nový konečný automat se stavy odpovídajícími skupinám stavů vzniklým při minimalizaci. Upravme pravidla, počáteční stav a konečné stavy tak, aby odpovídali skupinám (statická metod `GroupState.findGroup`).

Zápis výstupu

Zápis výsledného automatu se následně provede převedením do stringu a následným výpisem na output získaný v argumentech.

Převedení se provádí zavoláním Python metody `__str__`, kterou jsme ve třídě `FiniteStateMachine` redefinovali, tak aby vracela konečný automat v odpovídajícím formátu.