Dokumentace úlohy MKA: Minimalizace konečného automatu v Pythonu 3 do IPP 2011/2012
Jméno a příjmení: Jan Pacner
Login: xpacne00

Rozhraní

Pro řešení uvedeného zadání byly zvoleny 3 dodatečné moduly (*os*, *sys*, *operator*). Vzhledem k omezením na zpracování parametrů ze zadání, nebylo možné využít existujicích modulů pro jednoduché zpracování parametrů. Toto je tedy řešeno ručním parsováním.

Lexikální analýza

Lexikální analyzátor je rozdělen do dvou částí. První (*lex_symbol()*) je velice specifická, určená pouze pro zpracování symbolů vstupní abecedy a druhá zajišťuje zbytek. Pro pohodlnější práci bylo zvoleno načtení kompletního vstupního řetězce do pamětí a až poté provádění analýz. Předpokládá se, že konečné automaty nebudou příliš veliké. K pohybu ve vstupním řetězci je používána indexace (simulace *getchar()*), která je ve výjimečných případech porušena např. „vrácením“ se o 2 indexy zpět. Celý lexikální analyzátor zapouzdřuje funkce *get_token()*, která vrací dvojici symbol a význam. Význam je třída ve třídě, čímž je nasimulováno bezpečné použití enum.

Syntaktická analýza

Syntaktický analyzátor je vytvořen postupným čtením jednotlivých komponent v předepsaném pořadí. Pro čtení položek z komponent, které mohou obsahovat položek více je užito cyklů. Jedinou výjimkou je sémantická kontrola determinizace vstupního automatu, která je prováděna ihned při vyhodnocování „existence“ daného pravidla.

Sémantická analýza a kontroly

Sémantická kontrolá sestává zejména z předepsaných omezení. Využíváno je zde implementačních možností jazyka Python – např. přítomnost datové struktury množina spolu s přetíženými operátory.

Nejdůležitější součástí je však zpracování samotného automatu. Nejprve ověřování, zdali se jedná o dobře specifikovaný automat. Toto zahrnuje v následujícím pořadí. Nepřítomnost epsilon (vymazávacího) pravidla, což je zajištěno projitím všech pravidel a kontrolou, zdali symbol není prázdný. Následuje vyhledání nedostupných stavů, což je realizováno vybudováním zjednodušených pravidel coby pouhých přechodů mezi stavy (tzn. bez přítomnosti symbolu), avšak v opačném pořadí. Poté se tato struktura prochází metodou Breadth First Search (slepé prohledávání do šířky) dokud není nalezen startovací stav (to v případe dostupného stavu) a nebo dokud není projitý celý graf, což znamená, že tento stav je nedostupný. V tom případě program končí s chybou.

Totožný přístup je též použit pro kontrolu na přítomnost neukončujících stavů. K tomu je využita obrácená verze výše uvedených zjednodušených pravidel. Kontrola však neprobíhá na startující stav, nýbrž na množinu koncových stavů. Je zde ještě jeden rozdíl, a sice funkce vrací neukončující stav, pokud nějaký najde (více než jeden neukončující stav je pochopitelně chyba ukončující program).

Poslední vlastností dobře specifikovaného konečného automatu je úplnost. Kontroluje se tedy, zdali se v každém ze vstupních stavů nelze zaseknout tím, že by nebylo možné přečíst další symbol na vstupu. Toto je zabezpečeno projitím všech stavů a kontrolou, zdali z každého z nich „vede“ cesta do některého ze vstupních stavů (lze např. i sám do sebe), a to pro každý symbol ze vstupní abecedy.

Minimalizace dobře specifikovaného automatu

Vlastní minimalizace je provedena úvodním rozdělením stavů na koncové a jejich doplněk. Tyto dvě části jsou reprezentovány „sub-seznamy“ obsaženými v dočasném seznamu. Myšlenkou je pro každý symbol štěpit aktuální sub-seznamy na menší dokud to je možné. Tento přístup zajišťuje, že budou provedena všechna možná štěpení. Samotné štěpení je rozdělení sub-seznamu na dvě části podle pravidel štěpení. Je vybrán první dostupný sub-seznam, který je poté rozštěpen (pokud možno), sub-seznam je ze seznamu odstraněn a výsledek (ať již původní sub-seznam, či štěpiny) je vždy uložen do dočasného seznamu, který je po výprázdnění výchozího seznamu prohozen právě za ten vyprázdněný. Tímto je v každé iteraci zajištěna aktuálnost seznamu obsahujícího sub-seznamy, reprezentující štěpiny.

Výsledný dočasný seznam se štěpinami je poté převeden na lineární seznam se jmény reprezentujícími sloučené stavy v jednotlivých sub-seznamech. Tento seznam je následně použit jako indexovací tabulka pro převod všech ostatních struktur jako nově vytvořená pravidla, startující stav a koncové stavy do normalizované podoby. Tyto struktury obsahují v průběhu výpočtu pouze indexy umístění sub-seznamů v hlavním (výsledném) seznamu. Tímto vzniknou znovu úplně nové, avšak již finální, struktury s minimalizovaným automatem.

Výstup v normální formě

Normální forma výstupu je zajištěna integrovanými funkcemi a metodami jako *join()*, *sorted()* a *itemgetter()*. *join()* zajistí spojení seznamu s definovaným „meziřetězcem“, kde tento seznam je ještě předtím seřazen lexikograficky

funkcí *sorted()*, a to včetně složených klíčů, předpřipravených funkcí *itemgetter()*. Pro snadné použití *itemgetter()* je nejprve vnitřní reprezentace pravidel převedena do pravidelné tabulky o 3 sloupcích a N řádcích, kde N je počet pravidel. Tato tabulka je však znovu vnitřně reprezentována seznamem.