Derekursywacja i optymalizacja algorytmów

# Czyli przekształcanie algorytmów rekurencyjnych na ich postać iteracyjną

W przypadku funkcji rekurencyjnych kłopotliwe jest intensywne wykorzystanie stosu, który służy do odtwarzania ”zamrożonych” egzemplarzy tej samej funkcji. Przy każdym takim nieczynnym chwilowo egzemplarzu trzeba zachować pełny zestaw jego parametrów wywołania, zmiennych lokalnych czy wreszcie adres powrotu. To tyle, jeśli chodzi o zajętość pamięci. Nie zapominajmy jednak, że zarządzanie przez kompilator/interpreter całym tym bałaganem zużywa cenny czas procesora, który się dodaje do ogólnego czasu wykonania programu.

Rozwiązanie: podczas tworzenia oprogramowania wykorzystajmy całą siłę i elegancję algorytmów rekurencyjnych, natomiast w momencie pisania wersji końcowej (tej, która ma być używana w praktyce) wykonajmy transformacje na analogiczną postać iteracyjną (jeśli to konieczne ze względu na parametry czasowe aplikacji lub jej ograniczone zasoby pamięci).

**Procedura iteracyjna I() jest równoważna procedurze rekurencyjnej P(), jeśli wykonuje dokładnie to samo zadanie co P(), dając identyczne rezultaty.**

**Wywołanie rekurencyjne procedury P jest zwane *terminalnym* (ang. end-recursion), jeśli nie następuje po nim żadna instrukcja procedury.**

**Twierdzenie 1.**

**Następujące procedury, P1 i P2, są wzajemnie równoważne, pod warunkiem że P1 zawiera tylko jedno rekurencyjne wywołanie terminalne:**

|  |  |
| --- | --- |
| **def P1(x):**  **if (Cond(x)):**  **Instr1(x)**  **else:**  **Instr2(x)**  **P1(F(x))** | **def P2(x):**  **while not Cond(x):**  **Instr2(x)**  **x = F(x)**  **Inst1(x)** |