Cytowanie i dane symboliczne

Aby sensownie mówić o cytowaniu warto zastanowić się nad rozróżnieniem które do tej pory nie było specjalnie widoczne: nad tym czym w naszym języku są *wyrażenia*, a czym *wartości*. W najprostszym przypadku wartości liczbowych widzieliśmy że wyrażenia mogły być dowolnie skomplikowanymi kombinacjami, które potrafiliśmy *uprościć* do — mniej skomplikowanych — wartości. Zdefiniowaliśmy *model obliczeń*, który dał nam mechaniczną procedurę obliczania: ta procedura przenosi nas między dwoma światami — ze świata wyrażeń, w którym występuje na przykład kombinacja `(+ 100 23)`, do świata wartości, w którym odpowiada jej liczba `123`. Jednocześnie jednak w tym prostym systemie wartości same były *podzbiorem* wyrażeń: `123`, podobnie jak inne liczby, samo w sobie jest poprawnie sformowanym wyrażeniem.

Tak długo jak pracowaliśmy przede wszystkim z danymi liczbowymi ta sytuacja się utrzymywała.[1] Jednak wraz z pojawieniem się konstrukcji par, nawet te wartości które Racket wypisuje w całości — takie jak wynik obliczenia `(cons 1 2)`, czyli parę `(1 . 2)` nie są poprawnymi wyrażeniami. O operacji cytowania możemy teraz myśleć jak o operacji pozwalającej nam *zanurzyć* świat (wypisywalnych) wartości z powrotem w świecie wyrażeń. Przyjrzyjmy się zatem czym cytowanie jest i jak działa.

W swojej najbardziej bazowej postaci cytowanie jest nową formą specjalną, którą dodajemy do języka: forma `(quote` v`)` jest poprawnie sformowanym wyrażeniem jeśli v jest (wypisywalną) wartością. Przykładowo, poprawnymi wyrażeniami są `(quote 1)`, `(quote ())` czy `(quote (1 . 2))`. Jak w przypadku każdej formy specjalnej, musimy zdefiniować dla cytowania *regułę obliczania*, przenoszącą nas ze świata wyrażeń do świata wartości. W tym przypadku może ona być bardzo prosta — w końcu mamy pod ręką wartość! Mówimy zatem, że dla dowolnej wartości v wyrażenie `(quote` v`)` oblicza się po prostu do v. Zatem w przypadku `(quote 1)` dostaniemy jako wynik liczbę `1`, w przypadku `(quote ())` — listę pustą `()`, a w przypadku `(quote (1 . 2))` — parę

[1] Sprawa jest bardziej skomplikowana jeśli chodzi o procedury: nasz model obliczeń mówi że forma `lambda` oblicza się do samej siebie — a więc jest dobrym wyrażeniem — ale próba wpisania do interpretera `#<procedure:+>` (którą to wartością Racket odpowiada na wyrażenie `+`) zakończy się niepowodzeniem. Wynika to z faktu że podstawieniowy model obliczeń jest tylko pewnym, niedokładnym przybliżeniem tego jak faktycznie liczy Racket — a czego dowiemy się w najbliższych tygodniach

`(1 . 2)`. Oczywiście używanie formy `(quote 1)` bezpośrednio mija się z celem, ale w przypadku dużej, rozgałęzionej struktury danych zacytowanie jej może być dużo wygodniejsze niż jawna budowa za pomocą konstruktorów (czy nawet wieloargumentowych procedur takich jak `list`). Żeby uczynić używanie cytowania wygodniejszym, dodajemy też do języka cukier składniowy: forma `'`v jest składniowo równa cytowaniu wartości v, czyli `(quote` v`)`. Stąd też bierze się znak cytowania który Racket automatycznie dodaje wyświetlając wartość która jest parą: interpreter Racketa wyświetla wartości w postaci zacytowanej, co pozwala, na przykład, na wygodne skopiowanie ich do kodu programu.

Dane symboliczne

Widzieliśmy że nasze wartości wyglądają łudząco podobnie do programów: wartość programu `(list 1 2 3 4)`, czyli lista `(1 2 3 4)`, jest *strukturalnie* bardzo podobna do programu który ją oblicza. Możemy się zastanowić czy można zacytować sam *program* — i co mogłoby to oznaczać. Jeśli napisalibyśmy `(quote (list 1 2 3 4))`, to nasza reguła obliczania mówi że wartością powinna być pięcioelementowa lista. Z liczbami nie ma problemu, wiemy że są dobrymi wartościami — ale czym powinien być pierwszy element tej listy? Przyjmujemy że jest to nowy rodzaj wartości — *symbol*. Nie zastanawiamy się zanadto jak taki symbol jest reprezentowany, jedyne co przyjmujemy to umiejętność porównywania symboli co do równości: przyjmujemy wbudowaną dwuargumentową procedurę `eq?`, która zaaplikowana do dwóch symboli zwraca prawdę tylko gdy symbole są tą samą, zacytowaną nazwą.

Zauważmy że symbole są jedynymi obiektami do których reprezentowania jako wyrażeń programu *potrzebujemy* cytowania: inne obiekty, takie jak pary czy liczby możemy zapisać jawnie w swojej podstawowej formie. W tym sensie, wyrażenie `'(list 1 2 3 4)` jest równoważne wyrażeniu `(list 'list 1 2 3 4)`: obydwa obliczają się do tej samej, pięcioelementowej listy. Danych symbolicznych będziemy używać głównie po to by odróżnić różne konstruktory danych: przykładowo, jeśli chcemy zbudować drzewo binarne z etykietami w liściach, ale etykiety same będą listami, to bez użycia jakiegoś rodzaju rozróżnienia trudno będzie nam stwierdzić gdzie kończy się drzewo a zaczyna lista. Jeśli jednak „zaetykietujemy” wierzchołki drzewa symbolami (na przykład `'node` i `'leaf` — zawsze będziemy w stanie odróżnić od siebie rodzaje węzłów drzewa.