, ,

1 Wprowadzenie

W rozdziale tym wprowadzamy język do opisu systemów współbieżnych. Uwaga! Głównym celem jest wyrobienie odpowiedniej intuicji u Czytelnika, zatem nie podajemy tutaj *żadnych* ścisłych definicji. Tych znajdziecie Państwo aż nadto w kolejnych rozdziałach (o ile one zaistnieją :). Wszelkie wyjaśnienia przeprowadzamy za pomocą przykładów, chcąc odwołać się raczej do intuicji Czytelnika niż do Jego matematycznej biegłości.

Nasz język chcemy oprzeć na jak najmniejszej liczbie konstrukcji syntaktycznych i pojęć pierwotnych. Inymi słowy, chcemy aby był on jak najprostszy. Z drugiej strony, ma on być wystarczająco expresywny, aby było możliwe opisanie w nim systemów spotykanych w praktyce.

Systemy będziemy opisywać w sposób kompozycjonalny, tzn. system najczęściej będzie składał się z pewnej liczby mniejszych składowych. Zachowanie systemu składa się z niepodzielnych *zdarzeń*. Zdarzenie to

- albo komunikacja systemu z otoczeniem
- albo zdarzenie wewnętrze, zachodzące wewnątrz systemu, nieobserwowalne z zewnątrz.

W tym drugim przypadku, jest to znów komunikacja, tym razem pomiędzy składowymi systemu. Zatem, każde zdarzenie jest komunikacją! Komunikacja jest jedyną aktywnością procesów!

Z punktu widzenia semantyki, interesować nas będzie *obserwowalne zachowanie* procesu. Będziemy przez to rozumieć zdolność procesu do komunikacji z otoczeniem. Czyli obserwator zewnętrzy może obserwować dany system tylko komunikując się z nim.

Poczynimy pewną liczbę założeń. Po pierwsze, *ignorujemy czas*. Jeśli zatem chcemy opisać długotrwałą aktywność procesu, powinniśmy odróżnić co najmniej dwa zdarzenia: rozpoczęcie aktywności oraz zakończenie tej aktywności. Ponadto, dla uproszczenia nie będziemy dopuszczać jednoczesności (niezależności) zdarzeń. Zamiast jednoczesności, zdarzenia będą mogły być wykonane w dowolnej kolejności.

Po drugie, odrzucamy często spotykany podział na aktywne procesy obliczeniowe i pasywne media komunikacyjne. Medium komunikacyjne (np. bufor, pamięć dzielona, itp.) będzie też aktywnym procesem, pełnoprawnym uczestnikiem obliczeń. Ponadto, dla uproszczenia przyjmujemy iż w zdarzenie komunikacyjne zaangażowane są zawsze dokładnie dwa procesy. Wybieramy najprostszy model komunikacji:

synchroniczny, niebuforowany. Np. komunikację buforowaną można zaimplementować jako osobny proces.

Popatrzmy na kilka prostych przykładów, mających za cel wprowadzenie poszczególnych elementów naszego języka. Zacznijmy od bardzo prostego procesu C ktory naprzemienne odbiera wartość x i wysyła ją dalej:

$$C \stackrel{\text{def}}{=} in(x).C'(x) \tag{1}$$

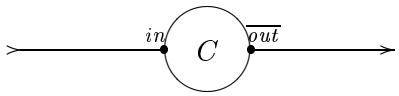
$$C'(x) \stackrel{\text{def}}{=} \overline{out}(x).C \tag{2}$$

albo krócej

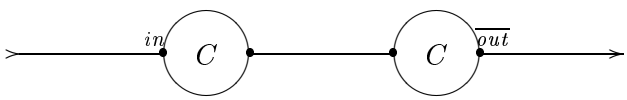
$$C \stackrel{\text{def}}{=} in(x).\overline{out}(x).C$$

Zwróćmy uwage na wiązanie identyfikatorów: $in(x)$ wiąże x w (1) oraz $C'(x)$ wiąże x w (2). Załóżmy tu, że x jest parametrem oznaczającym dowolną liczbę całkowitą. Konstrukcja postaci $\alpha.P$, oznaczająca: wykonaj α, a następnie kontynuuj wg P, będzie jedynym sposobem szeregowania zdarzeń w naszym języku. Na przykład, proces C może wykonać $in(7)$ i stać się procesem $C'(7)$. Czyli ze *stanu* C, po wykonaniu $in(7)$, przechodzimy do stanu $C'(7)$. Taką zmianę stanu będziemy oznaczać $C \stackrel{in(7)}{\longrightarrow} C'(7)$ i nazywać *zdarzeniem wykonanym przez* C.

Będziemy też posługiwać się językiem graficznym, odzwierciedlającym strukturę połączeń komunikacyjnych pomiędzy procesami:



Nazwy *in* i *out* możemy intuicyjnie uważać za nazwy portów obsługiwanych przez proces C; $\overline{in}$ jest nazwą *komplementarną* do *in*. Przyjmujemy chwilowo konwencję, iż porty wyjściowe mają nazwy z kreską u góry, a porty wejściowe nie. Wyobraźmy sobie przez chwile, iż mamy do dospozycji operator dwuargumentowy $_ \frown _$ który zastosowany do dwóch kopii procesu C daje w wyniku $C \frown C$:



Dwa egzamplarze procesu C są teraz połączone kanałem komunikacyjnym. Zauważmy, iż nazwy portów przyłączonych do tego kanału są nieistotne a zatem nie są uwidocznione na rysunku. Niech $C^{(n)}$ oznacza $\overbrace{C \frown \ldots \frown C}^{n \text{ razy}}$. Zaniedbując wewnętrzną komunikację pomiędzy sąsiednimi procesami C, proces $C^{(n)}$ zachowuje się jak bufor

n-elementowy. Bufor taki można również opisać bezpośrednio, np. tak:

$$\begin{aligned}
\mathrm{Buff}_n(\epsilon) &\stackrel{\mathrm{def}}{=} in(x).\mathrm{Buff}_n(x) \\
\mathrm{Buff}_n(v_1 \ldots v_n) &\stackrel{\mathrm{def}}{=} \overline{out}(v_n).\mathrm{Buff}_n(v_1 \ldots n_{n-1}) \\
\mathrm{Buff}_n(v_1 \ldots v_k) &\stackrel{\mathrm{def}}{=} in(x).\mathrm{Buff}_n(xv_1 \ldots v_k) + \overline{out}(v_k).\mathrm{Buff}_n(v_1 \ldots v_{k-1}) \quad \text{o ile } 0<k<n
\end{aligned}$$

Operator $_._$ ma tutaj wyższy priorytet niż $_ + _$. Proces Buff_n posiada parametr, podobnie jak proces C' powyżej. Zawsze będziemy zakładać, iż parametry procesów należą do pewnego danego typu danych, w tym przypadku są to skończone listy liczb całkowitych.

Zauważmy, że proces $\mathrm{Buff}_n(\epsilon)$ wyraża tylko zewnętrzne zachowanie bufora, podczas gdy proces $C^{(n)}$ stanowi pewną implementację. Zatem, nasz język można uważać zarówno za język specyfikacji jak i programomowania. Myśląc ogólniej, będziemy w stanie opisać dany system na różnych poziomach uszczegółowienia, pomiędzy specyfikacją wymagań a końcową implementacją.

Czy procesy $C^{(n)}$ i $\mathrm{Buff}_n(\epsilon)$ są równoważne?

$$C^{(n)} = \mathrm{Buff}_n(\epsilon)?$$

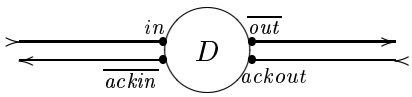
Aby umieć odpowiedzieć na to pytanie, musimy określić dokladniej o jaką równoważność nam chodzi. Zaproponowano wiele pojęć równoważności behawioralnej (obserwacyjnej) dla procesów współbieżnych. Z kilkoma z nich zapoznamy się poźniej, natomiast najważniejszą rolę odegra tzw. *równoważność bisymulacyjna.*

Załóżmy, iż mamy dane dwa systemy, np. $C^{(n)}$ i $\mathrm{Buff}_n(\epsilon)$. Wyobraźmy sobie taraz grę rozgrywaną przez dwóch graczy, Kowalskiego i Nowaka. Celem Kowalskiego jest wykazanie, iż dwa procesy są równowazne. Celem jego oponenta jest dowiedzenie, iż sie różnią. Konfiguracją (stanem) gry jest para stanów: stan procesu $C^{(n)}$ oraz stan procesu $\mathrm{Buff}_n(\epsilon)$. Gracze wykonują ruchy naprzemiennie. Najpierw Nowak wybiera dowolny z dwóch systemów i wykonuje w nim jedno z możliwych zdarzeń. Teraz następuje ruch Kowalskiego: musi on odpowiedzieć w drugim systemie za pomocą takiego samego zdarzenia. Rozgrywka toczy się dalej aż do momentu gdy jeden z graczy nie ma żadnego dozwolonego ruchu – w takiej sytuacji gracz ten przegrywa. W przeciwnym przypadku, tzn. w przypadku gdy rozgrywka sie nie kończy i jest nieskończona, wygrywa Kowalski, gdyż Nowakowi nie udało się odróżnić systemów. Mówimy, iż dwa systemy są równoważne, gdy Kowalski zawsze wygrywa, tzn. gdy ma strategię wygrywającą.[1]

Ćwiczenie 1 *Czy procesy $C^{(n)}$ i $\mathrm{Buff}_n(\epsilon)$ są równoważne?*

[1] Wszelkie wątpliwości nasuwające się uważnemu Czytelnikowi w tym miejscu zostaną rozwiane poźniej.

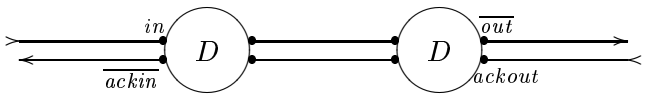
Rozważmy teraz pewną odmianę procesu C:



zdefiniowaną następująco:

$$D \stackrel{\text{def}}{=} in(x).\overline{out}(x).ackout.\overline{ackin}.D$$

Zdarzenia $\overline{ackin}$ i $ackout$ nie są sparametryzowane, stanowią więc one tylko synchronizację procesów. Operator $_\frown_$ modyfikujemy tak aby port $\overline{out}$ został połączony z portem in a port $\overline{ackin}$ z portem $ackout$. Czyli $D\frown D$ wygląda następująco:



Ćwiczenie 2 *Czy* $D^{(n)} = \mathrm{Buff}_n(\epsilon)$ *? Podpowiedź: rozważ* $D\frown D$.

Zmodyfikujmy D następująco:

$$D \stackrel{\text{def}}{=} in(x).\overline{ackin}.\overline{out}(x).ackout.D$$

Czy teraz $D^{(n)}$ zachowuje się znów jak bufor?

Ćwiczenie 3 *Zmodyfikuj definicje procesu* Buff_n *tak, aby zachodziła równoważność* $D^{(n)} = \mathrm{Buff}_n$.

Nasz język jest już wystarczająco bogaty aby opisać maszyny skończeniestanowe. Wyobraźmy sobie maszynę sprzedającą dwa rodzaje czekoladek: duże kosztujące 2zł i małe kosztujące 1zł. Posiada ona dwa otwory wrzutowe, jeden dla monet 1zł a drugi dla monet 2zł. Po wrzuceniu jednej monety, użytkownik wciska jeden z dwóch guzików: guzik `duża` jeśli chce kupić dużą czekoladkę a guzik `mała`, jeśli małą. Następnie powinien on wyciągnąć czekoladkę z podajnika – jego opróżnienie jest niezbędne dla wznowienia działania maszyny.

Tak proste zachowanie można z łatwością opisać następująco:

$$\text{Maszyna} \stackrel{\text{def}}{=} \texttt{2z.dua.we}.\text{Maszyna} + \texttt{1z.maa.we}.\text{Maszyna}$$

Ćwiczenie 4 *Zmodyfikuj proces* `Maszyna` *tak, aby po wrzuceniu monety 1zł było wciąż możliwe kupienie dużej czekoladki.*

Ćwiczenie 5 *Czy poprawnym rozwiązaniem poprzedniego zadania jest proces:*

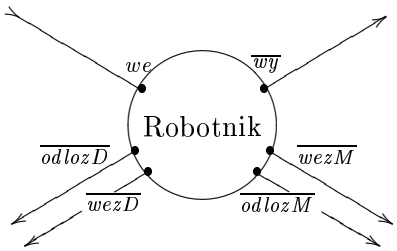
$$\text{M} \stackrel{def}{=} \texttt{2z.dua.we}.\text{M} + \texttt{1z.maa.we}.\text{M} + \texttt{1z.1z.dua.we}.\text{M}?$$

Jeśli nie, to wskaż strategię Nowaka w odpowiedniej grze.

Ćwiczenie 6 *Zmodyfikuj proces* `Maszyna` *tak, aby maszyna pamiętała kwotę wrzuconą dotychczas, z przedziału 0zł . . . 4zł. Maszyna nie musi wydawać reszty.*

Zwróćmy uwagę na rzecz oczywistą ale zarazem bardzo ważną: opisując maszyne, wzięliśmy pod uwagę tylko te aspekty, które nas interesowały. Inaczej mówiąc, abstrahowaliśmy od nieistotnych dla nas w danym momencie szczegółów. I tak będzie zawsze: w naszym języku będziemy w stanie wyrazić tylko pewien *model* systemu rzeczywistego.

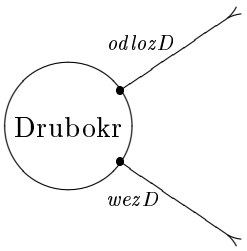
Ostatni przykład będzie miał za cel wprowadzenie wszystkich pozostałych konstrukcji syntaktycznych naszego języka. Opiszemy bardzo prostą linię produkcyją w pewnej fabryce, zatrudniającej dwóch robotników:



Zauważmy, iż graficzny zapis powyżej zawiera pełną informację o nazwach kanałów obsługiwanych przez process Robotnik. W przyszłości będziemy również używać zapisu postaci:

$$\text{Robotnik} : \{we, \overline{wy}, \overline{wezD}, \overline{odlozD}, \overline{wezM}, \overline{odlozM}\};$$

zbiór nazw po prawej stronie będziemy nazywać *rodzajem* procesu Robotnik. Każdy z robotników używa podczas pracy dwóch narzędzi: dużego i małego śrubokręta.

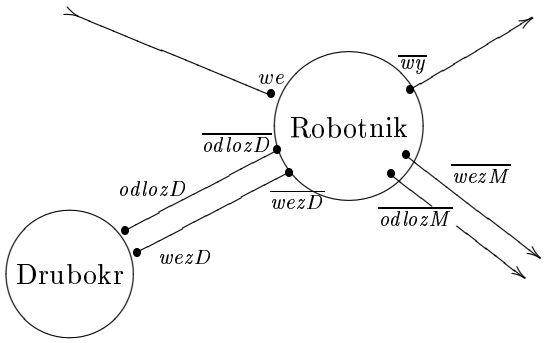


$$\text{DSrubokr} \stackrel{\text{def}}{=} wezD.odlozD.\text{Drubokr}$$

Zachowanie procesu Robotnik jest następujące:

$$
\begin{aligned}
\text{Robotnik} &\stackrel{\text{def}}{=} we(\text{część}).\text{Start}(\text{część}) \\
\text{Start}(\text{część}) &\stackrel{\text{def}}{=} \textbf{if}\ \ \text{atwa}(\text{część})\ \textbf{then}\ \ \text{Zakocz}(\text{część}) \\
&\quad\ \ \textbf{else}\ \ \textbf{if}\ \ \text{trudna}(\text{część})\ \textbf{then}\ \ \text{UyjDrub}(\text{część}) \\
&\quad\ \ \textbf{else}\ \ (\text{UyjDrub}(\text{część}) + \text{UyjMrub}(\text{część})) \\
\text{UyjDrub}(\text{część}) &\stackrel{\text{def}}{=} \overline{wezD}.\overline{odlozD}.\text{Zakocz}(\text{część}) \\
\text{UyjMrub}(\text{część}) &\stackrel{\text{def}}{=} \overline{wezM}.\overline{odlozM}.\text{Zakocz}(\text{część}) \\
\text{Zakocz}(\text{część}) &\stackrel{\text{def}}{=} \overline{wy}(gotowa(\text{część})).\text{Robotnik}
\end{aligned}
$$

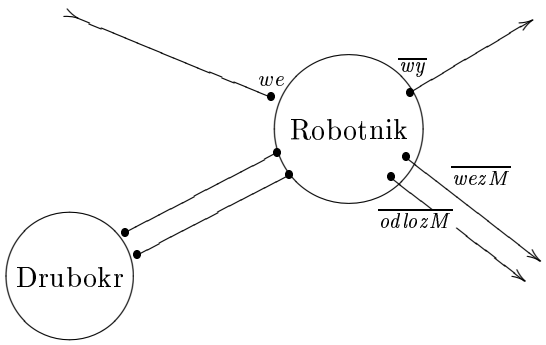
Robotnik komunikuje się z procesem Drubokr, zatem fragment naszego systemu będzie stanowiło *złożenie równoległe* procesów Robotnik i DSrubokr, które będziemy zapisywać Robotnik|DSrubokr. Graficznie:



Komplementarne porty obydwu procesów zostały połączone. Oznacza to, iż dwa procesy mogą wspólnie dokonać komunikacji (przypomnijmy, komunikacja jest synchroniczna). Tym niemniej, każdy z procesów może wciąż komunikować się przez te porty z innymi procesami lub ze światem otaczającym. Gdybyśmy chcieli tę dodatkową zdolność do komunikacji wykluczyć, możemy użyć kolejnego operatora. Proces

$$(\text{Robotnik}|\text{DSrubokr})\backslash\{wezD, odlozD\}$$

wygląda następująco:



jedyna różnica to wymazanie nazw portów ze zbioru $\{wezD, odlozD\}$ oraz nazw komplementarnych. Pamiętajmy jednak, iż opisujemy fabrykę zatrudniającą dwóch robotników korzystających z narzędzi. Rozważmy proces

$$((\text{Robotnik}|\text{Drubokr})\backslash\{wezD, odlozD\})|\text{Robotnik} \tag{3}$$

i zauważmy, iż drugi robotnik nie ma dostępu do dużego śrubokręta! Zatem poprawna kolejność składania systemu ze składowych jest następująca:

$$(\text{Robotnik}|\text{Drubokr}|\text{Robotnik})\backslash\{wezD, odlozD\} \tag{4}$$

a cały system może zostać zdefiniowany następująco:

$$\text{Fabryka} \stackrel{\text{def}}{=} (\text{Robotnik}|\text{Drubokr}|\text{Mrubokr}|\text{Robotnik})\backslash\{wezD, odlozD, wezM, odlozM\} \tag{5}$$

Ćwiczenie 7 *Narysuj reprezentacje graficzne procesów (3), (4) oraz (5). Pamiętaj, że złożenie równoległe nie wymazuje nazw połączonych portów.*

Ćwiczenie 8 *Uzasadnij, że operator złożenia równoległego jest symetryczny i łączny, tzn. złożenie równoległe wielu procesów zachowuje się tak samo, niezależnie od kolejności składania.*

Ćwiczenie 9 *Podaj wszystkie możliwe wyrażenia inne niż (5) definiujące poprawnie fabrykę. Nie odróżniamy wyrażeń różniących się tylko kolejnością złożenia równoległego.*

Ostatnią konstrukcją syntaktyczną jest możliwosc przemianowania nazw portów. Np. proces Mrubokr, którego definicję (rozmyślnie) dotychczas pominęliśmy, zachowuje się dokladnie tak samo jak Drubokr, aczkolwiek nazwy portów różnią się nieznacznie. Zdefiniujemy go następujaco:

$$\text{Mrubokr} \stackrel{\text{def}}{=} \text{Drubokr}[wezM/wezD, odlozM/odlozD]$$

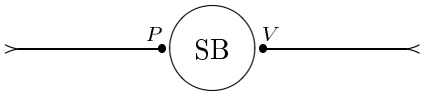
Ćwiczenie 10 *Rozważmy następujacą specyfikację fabryki:*

$$\text{SpecFabr} \stackrel{def}{=} \text{Rob}|\text{Rob}$$
$$\text{Rob} \stackrel{def}{=} in(\textit{część}).\overline{out}(\textit{gotowa}(\textit{część})).\text{Rob}$$

Czy proces Fabryka *jest poprawną implementacją specyfikacji* SpecFabr*? Tzn., czy* Fabryka = SpecFabr*?*

Ćwiczenie 11 *Rozważ zmodyfikowane zachowanie robotnika: odkłada on narzędzie dopiero po wysłaniu gotowej części. Zmodyfikuj odpowiednio opis fabryki. Czy wciąż zachodzi równoważność* Fabryka = SpecFabr*?*

Ćwiczenie 12 *Zdefiniuj semafor binarny:*



oraz semafor ogólny SO_n*, dla* $n \geq 1$*. Czy zachodzi*

$$\text{SO}_n = \overbrace{SB|\ldots|SB}^{n \text{ razy}} \ ?$$