ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ

Магдалина Тодорова
magda@fmi.uni-sofia.bg
todorova_magda@hotmail.com
кабинет 517, ФМИ

Тема 9

Деструктивни процедури и присвояване на стойност в езика Scheme

1. Присвояване на стойност

Въведеното досега подмножество на езика е достатъчно за решаване на голям клас задачи. За определен кръг задачи е полезно едно разширение на езика – ***присвояването на стойности***.

То нарушава функционалността му, но е въведено с цел – удобство при решаване на някои приложни задачи и по-голяма ефективност при изпълнението.

Реализира се чрез т.н. ***деструктивни операции***.

Дефиниция. Процедура, която променя всички или част от аргументите си, се нарича ***деструктивна***.

1. Присвояване на стойност. Специална форма set!

Синтаксис:

(set! <name> <expr>) в средата E

- set! е специален символ;

- <name> е символ, който предварително трябва да бъде дефиниран и свързан със стойност (не се оценява);

- <expr> е произволен израз.

Семантика: Оценява се **<expr>** в E и на променливата **<name>** се присвоява **[<expr>] в средата, в която <name> е свързана предварително със стойност**.

Оценката на обръщението към *set!* по стандарт е неопределена.

1. Присвояване на стойност. Специална форма set!

Забележка.

Действието на процедурата ***set!*** се свежда до разрушаване на съществуваща връзка на променлива със стойност и създаване на нова връзка на променливата с друга стойност в същата среда.

Следователно, процедурата ***set!*** е деструктивна. Имената на всички примитивни деструктивни процедури в Scheme завършват с "**!**".

1. Присвояване на стойност. Специална форма set!

Примери:

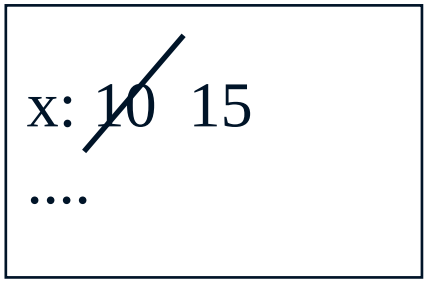
а)

>(define x 10)

x

>(set! x (+ x 5))

неопределено



глоб. среда

б) (set! a 1000)

води до грешка, тъй като със символа ***a*** предварително не е свързана оценка.

1. Присвояване на стойност. Специална форма begin.

Синтаксис:

(begin <израз$_1$> <израз$_2$> ... <израз$_n$>)

Семантика: Оценяват се последователно (от ляво на дясно) изразите **<израз$_i$>**. Оценката на обръщението към ***begin*** съвпада с **[<израз$_n$>]**.

1. Присвояване на стойност

Решенията на някои задачи се моделират чрез независими обекти, които се характеризират чрез състояния и поведението им се описва като промяна в състоянията им. По такъв начин по-нататъшното им поведение зависи от предишните им, т.е. от предисторията на съществуването им. За решаване на такива задачи се използва присвояването на стойности.

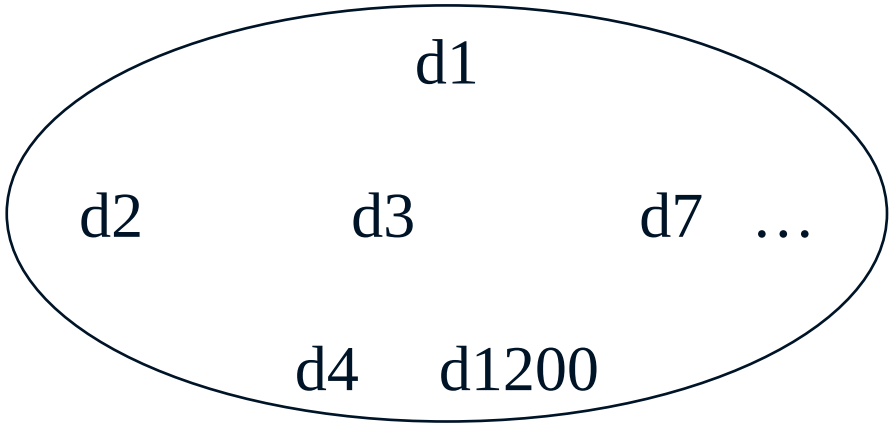
Задача

Да се дефинира процедура, която създава банкови сметки. Процедурата да позволява теглене на пари от банкова сметка.

1. Присвояване на стойност

Задача

Да се дефинира процедура, която създава банкови сметки. Процедурата да позволява теглене на пари от банкова сметка.



1. Присвояване на стойност

Всеки обект d1, d2, ... е конкретна банкова сметка и се характеризира с наличната сума в сметката.

Искаме също, от всяка банкова сметка да може да се теглят пари. Например, ако в банковата сметка d1 има 1000 лв., да може да се изтеглят 200 лв., след това 300 лв. и т.н.

Операцията *теглене* променя наличната сума на сметката d1 и тя става 800 лв. след първото изтегляне. Състоянието, описвано чрез сумата 1000 лв. на банковата сметка d1, се променя и става 800 лв.

За реализиране на промяната на състоянието на обект е необходимо да се използва деструктивната операция – ***присвояване на стойност***.

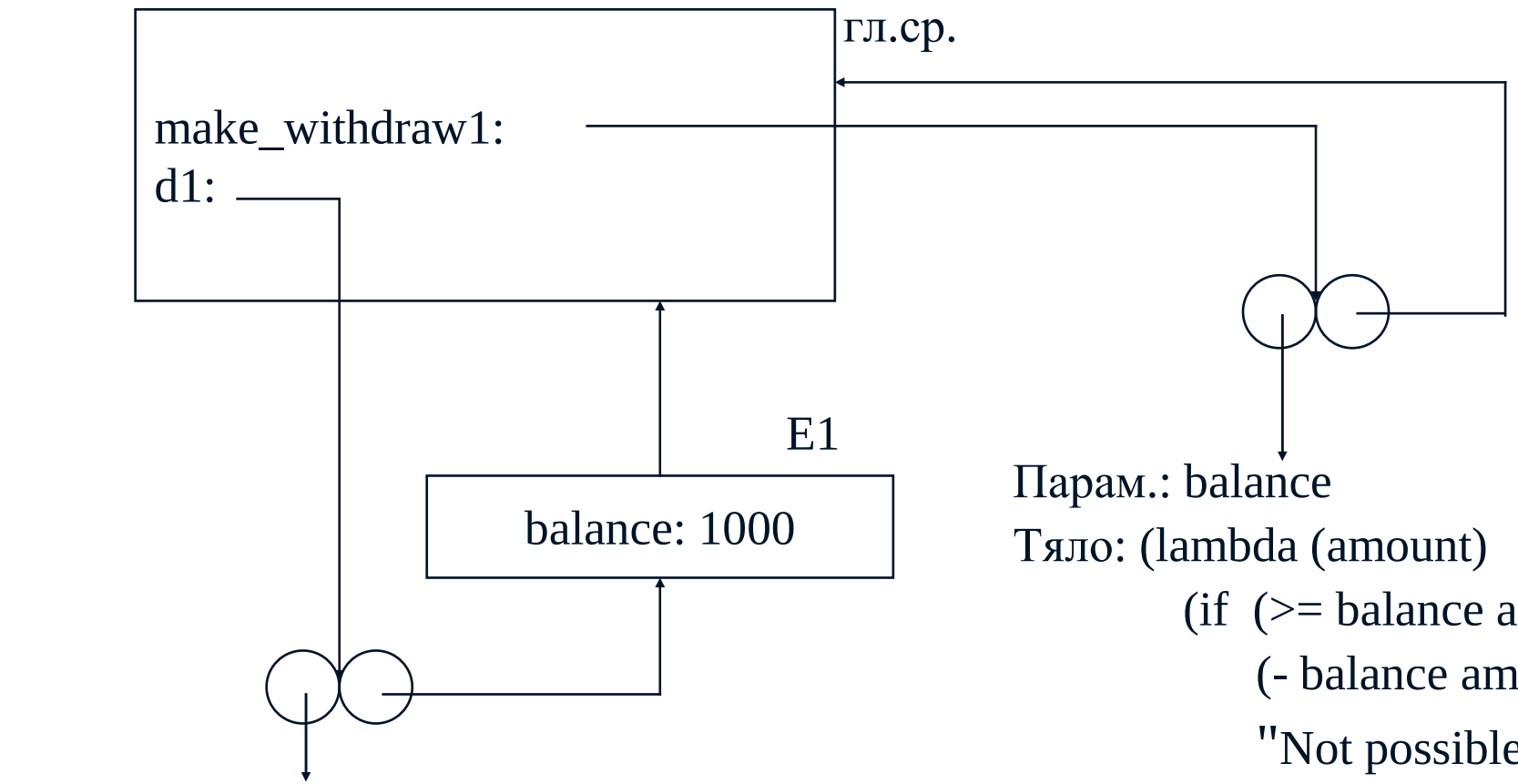
1. Присвояване на стойност

Неправилно решение:

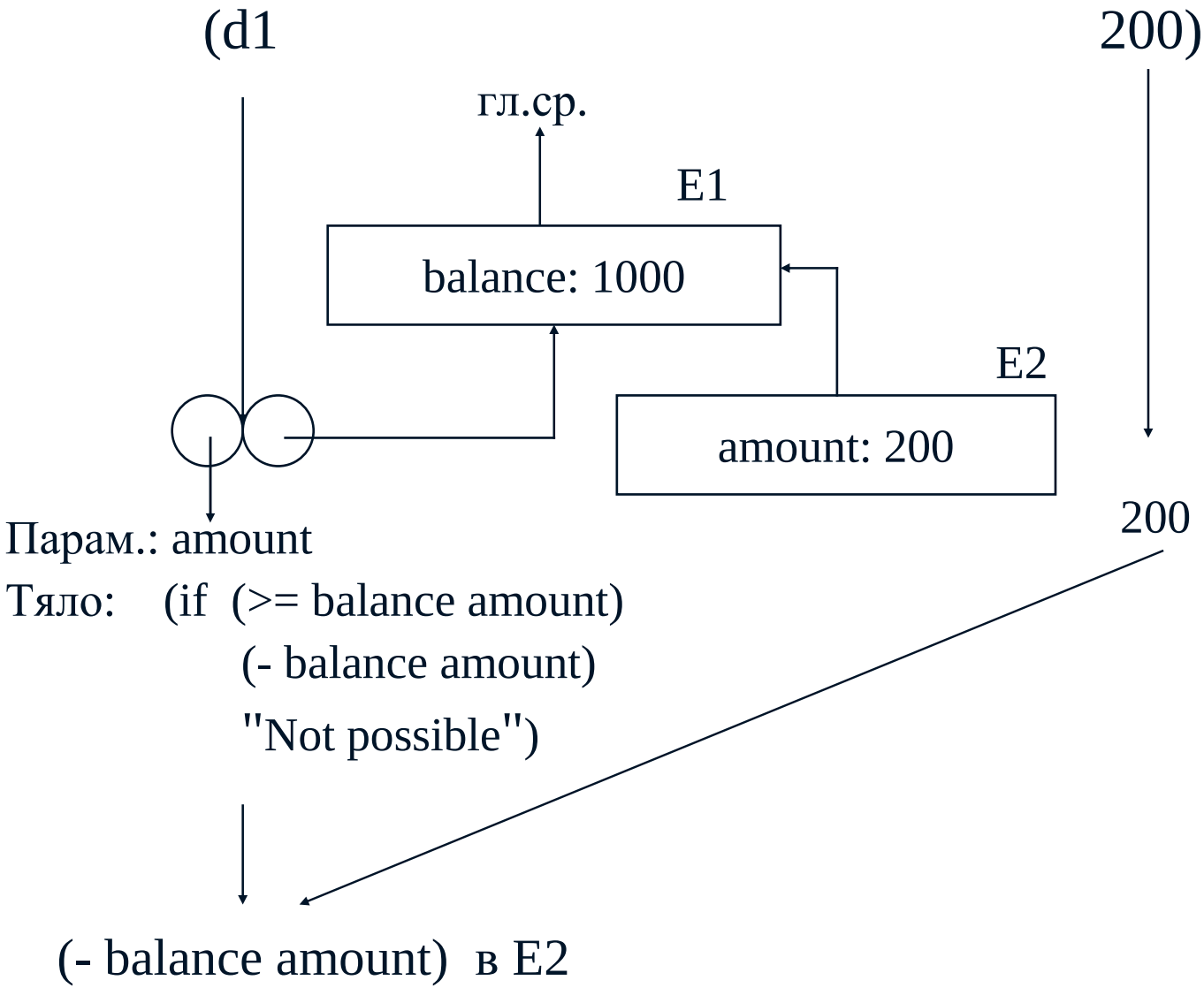
```
(define (make_withdraw1 balance)
    (lambda (amount)
        (if  (>= balance amount)
            (- balance amount)
            "Not possible")))

(define d1 (make_withdraw1 1000))
(define d2 (make_withdraw1 500))
  ...
```

d1 е банкова сметка с налична сума 1000 лв., d2 е банкова сметка с налична сума 500 лв. и т.н.

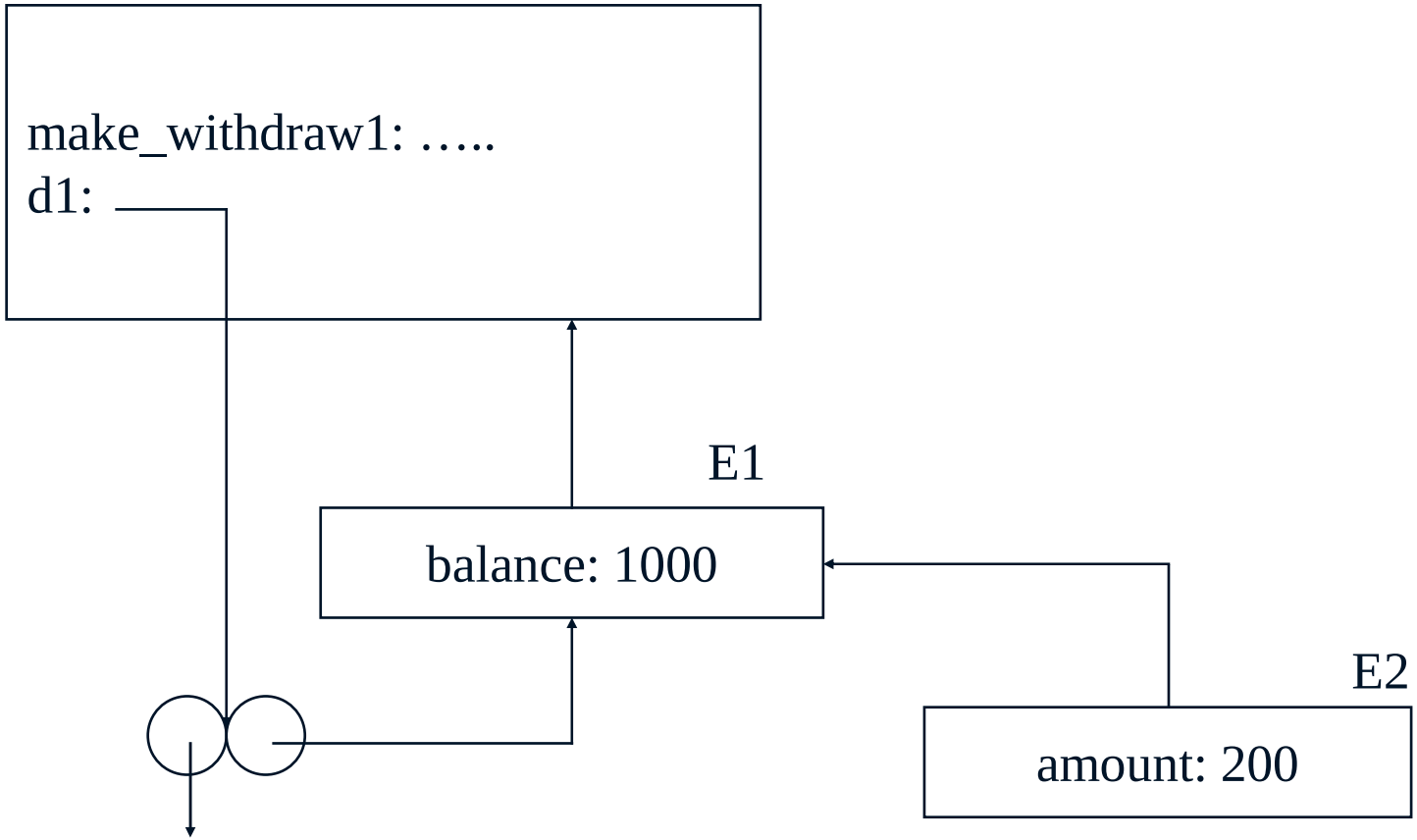
1. Присвояване на стойност



1. Присвояване на стойност



1. Присвояване на стойност



Парам.: amount

Тяло: (if (>= balance amount)
(- balance amount)
"Not possible")

1. Присвояване на стойност

Следователно, чрез процедурата make_withdraw1 могат да се създават банкови сметки d1, d2, ..., от които да се теглят пари. След обръщението (d1 200) от сумата 1000 лв. изтеглихме 200. Резултатът от тази оценка е 800. При това теглене обаче параметърът *balance*, означаващ състоянието на банковата сметка d1, не се е променил. Ако още веднъж се обърнем към d1 и искаме пак да изтеглим отново200 лв. ще получим пак 800 лв. като остатък в сметката.

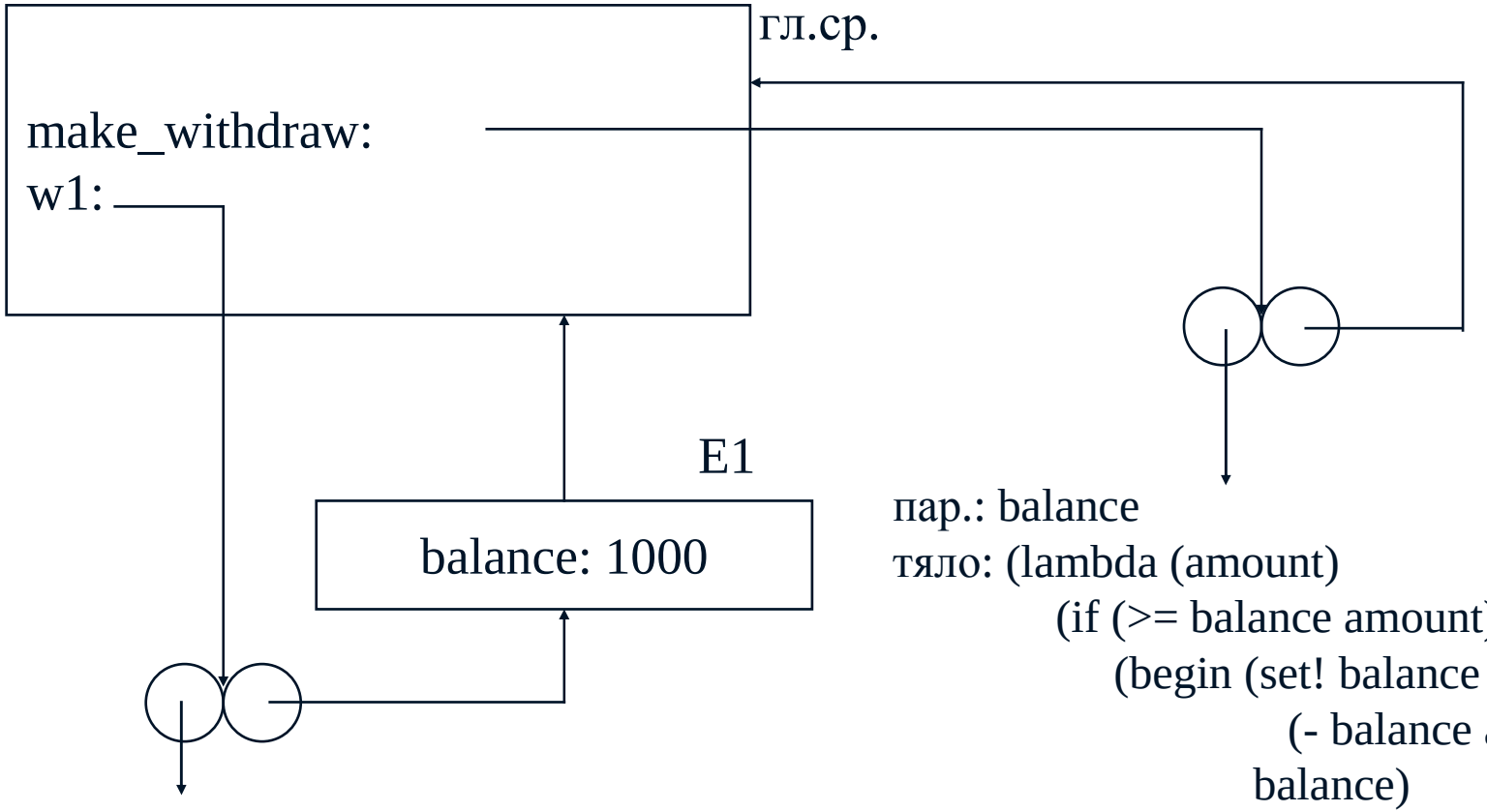
Направената реализация **не променя** състоянието на обекта банкова сметка.

Чрез следващата реализация на процедурата make_withdraw могат да се създават банкови сметки, от които коректно да се теглят пари.

1. Присвояване на стойност

```
(define (make_withdraw balance)
    (lambda (amount)
        (if (>= balance amount)
            (begin (set! balance (- balance amount))
                   balance)
            "Not possible")))

(define w1 (make_withdraw 1000))
(define w2 (make_withdraw 500))
...
```

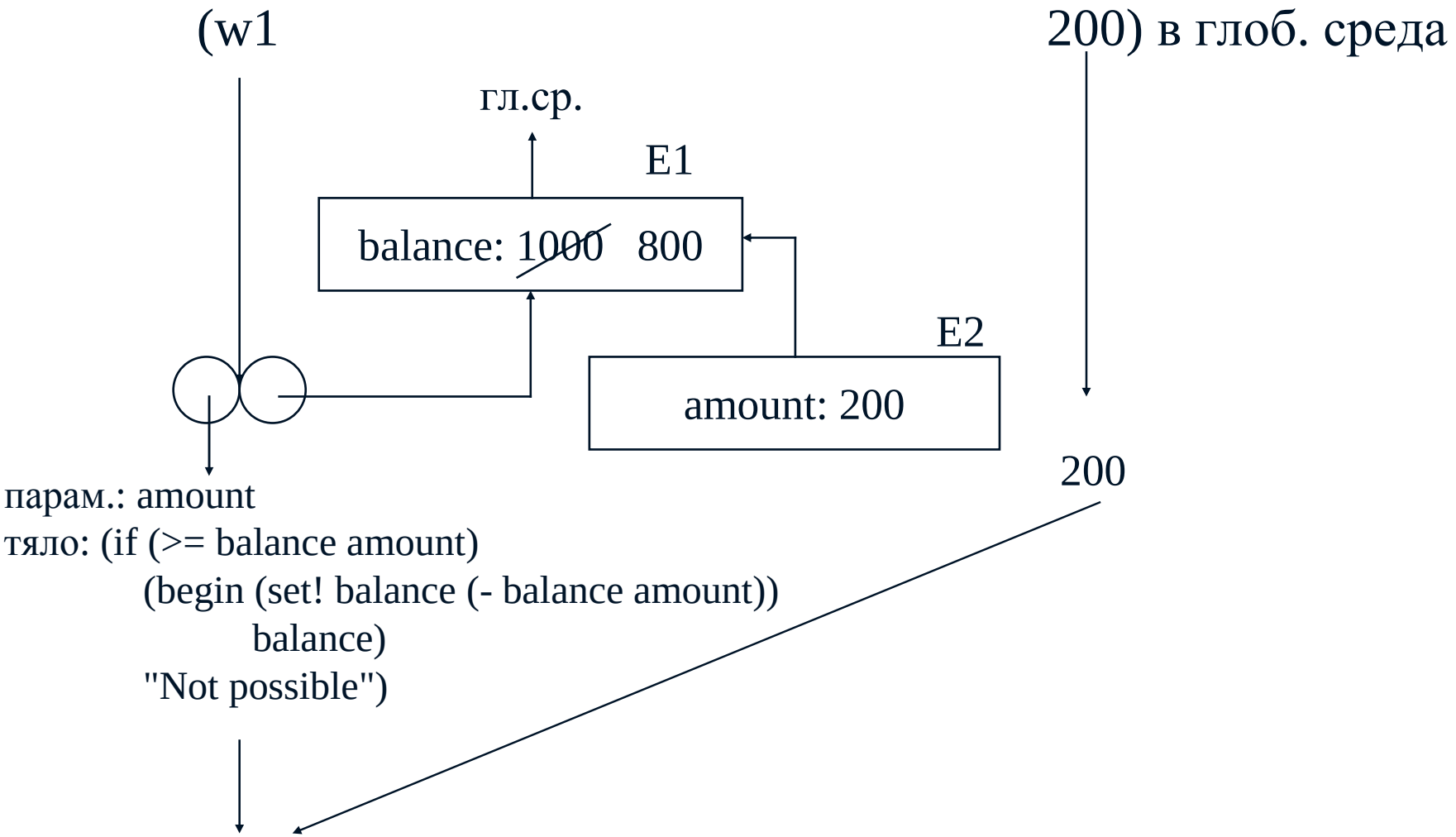
1. Присвояване на стойност



парам.: amount
тяло: (if (>= balance amount)
(begin (set! balance (- balance amount))
balance)
"Not possible")

пар.: balance
тяло: (lambda (amount)
(if (>= balance amount)
(begin (set! balance
(- balance amount))
balance)
"Not possible"))

1. Присвояване на стойност



1. Присвояване на стойност

Задача.

Да се дефинира процедура, която генерира банкови сметки, от които могат, както да се теглят, така и да се внасят пари.

```
(define (make-account balance)

    (define (withdraw amount)
        (if (>= balance amount)
            (begin (set! balance (- balance amount)) balance)
            "Not possible"))
```

1. Присвояване на стойност

```
(define (deposit amount)
    (set! balance (+ balance amount))
    balance)

(define (dispatch m)
    (cond ((eq? m 'withdraw) withdraw)
          ((eq? m 'deposit) deposit)
          (else (error "Unknown request!" m))))

dispatch)
```

1. Присвояване на стойност

Използване:

```
(define w1 (make-account 1000))

> ((w1 'withdraw) 800)                ; обръщение към withdraw
200
> ((w1 'withdraw) 500)                ; обръщение към withdraw
"Not possible"
> ((w1 'deposit) 1000)                ; обръщение към deposit
1200
```

2. Проблеми при присвояването на стойност

Първи проблем:

Изрази, които съдържат деструктивни операции не могат да бъдат оценявани чрез апликативния, както и чрез нормалния модел на заместване.

Пример: Да направим опит за оценка на (w1 200) като използваме апликативния модел.

2. Проблеми при присвояването на стойност

Имаме:

```
(w1 200)
((make_withdraw 1000) 200)    ; balance се свързва с 1000
((lambda (amount)
      (if (>= 1000 amount)
          (begin (set! 1000 (- 1000 amount))
                 1000)
          "Not possible")) 200)    ; amount се свързва с 200

(begin (set! 1000 (- 1000 200)) 1000)          ; което няма смисъл
```

2. Проблеми при присвояването на стойност

За оценяване на изрази, които съдържат деструктивни операции се използва моделът на средите. Причината е, че апликативният модел се базира на означението, че символите са имена на оценки.

Моделът на средите свързва всяка променлива със среда, в която оценката се запомня и може да се променя.

2. Проблеми при присвояването на стойност

Втори проблем:

Нека разгледаме процедурите d1 и d2, описани по следния начин:

(define d1 (make_withdraw1 1000))

(define d2 (make_withdraw1 1000))

Възниква въпросът: **Еквивалентни ли са (взаимно заменяеми ли са) d1 и d2 след като имат една и съща дефиниция?**

Приемлив отговор е ***да***, тъй като d1 и d2 се дефинират чрез един и същ израз и имат едно и също поведение - всяка една от тях изважда аргументът си от 1000.

d1 може да бъде заместена от d2 във всеки израз без да се промени резултатът.

2. Проблеми при присвояването на стойност

Нека разгледаме също и процедурите:

```
(define w1 (make_withdraw 1000))
(define w2 (make_withdraw 1000))
```

Еквивалентни ли са w1 и w2?

Отговорът е не, защото обръщенията към w1 и w2 водят до получаване на различни резултати. Например:

```
>(w1 200)
800
```

```
>(w1 200)
600
>(w2 200)
800
```

2. Проблеми при присвояването на стойност

Следователно, въпреки че w1 и w2 са еднакви в смисъл, че са дефинирани чрез един и същ израз, w1 **НЕ** може да се замени от w2 във всеки израз без да се промени резултатът на оценявания израз.

Дефиниция: Език, който поддържа концепцията, че равно може да бъде заместено с равно без промяна на оценките на изразите, се нарича ***референциално транспарентен (referential transparency)***.

2. Проблеми при присвояването на стойност

Подмножеството на Scheme, което използвахме преди въвеждането на специалната форма set! е референциално транспарентно.

Референциалната транспарентност се разрушава с въвеждането на set! и други деструктивни операции.

Втори проблем: Включването на специалната форма set! в езика разрушава неговата референциална транспарентност.

Разсъжденията за програми, които използват присвояване, стават значително по-сложни.