Dialekt Parse

1. Úvod

Dialekt Parse je jazyk specifické domény (domain-specific language - DSL) programovacího jazyka Red, který umožňuje stručné a přesné **ošetření** vstupu (input) podle zadaných gramatických pravidel (rules).

Ošetřením jsou zde míněny tyto procesy:

- **Hledání** - lokalizace zadaných vzorů
- **Validace** - ověření shody vstupu s určitou specifikací
- **Vyjmutí** - přefiltrování dat a agregace hodnot (e.g. scraping)
- **Modifikace** - transformace dat (vložení či odejmutí hodnot a změna detekovaných částí vstupů)
- **Zpracování jazyka** - použití kompilátorů, interpretů a lexikálních analyzátorů, zvláště pro DSL
- **Kódování a dekódování** - konverze datových formátů z jednoho do druhého.

1.1. Použití

Proces parsování se invokuje funkcí **parse** s použitím jednoduché skladby (podrobněji viz sekce Extra funkce):

```
parse <input> <rules>

<input> : jakákoli hodnota typu series! kromě image! a vector!
<rules> : hodnota typu block!, obsahující platný dialekt Parse (top-level rule)
```

Funkce `parse` vrací hodnotu typu `logic!` jako sdělení, zda zadaná gramatická pravidla byla či nebyla v úplné shodě se zadanými částmi vstupu.

1.2. Základní principy

Gramatická pravidla definují vzory, použité pro *vyhodnocení* vstupních dat. Základní hodnotící krok dialektu Parse je pravidlo *match*, které má jeden z těchto dvou výstupů:

- Pokud se zadané pravidlo *shoduje* s posuzovanou částí vstupu, je *úspěšné* a proces parse *postoupí* (advance) za konformní část vstupu (i pravidla);
- Pokud se *neshoduje*, pravidlo *selhává* (fails), načež se parse *vrátí* (backtracks) k *alternativním* pravidlům - pokud existují.

Hodnocení vstupu (parsing) je stále se opakující aplikace tohoto základního kroku, která je zastavena jednou z následujících dvou ukončujících podmínek:

- Neúspěšná aplikace pravidla: `parse` vrací `false`, signalizujíce neshodu
- Úplná shoda pravidla s porovnávaným vstupem a vyčerpání vstupu (to jest dosažení konce zkoumaného vstupu): `parse` vrací `true`, signalizujíce shodu.

POZOR: Nejsou-li splněny ukončovací podmínky, může vstoupit Parse do nekonečné smyčky.

1.3. Glosář

Dialekt parse je vylepšený člen rodiny formálních jazyků Parsing Expression Grammar (PEG), odlišující se širokou sadou vlastností a hlubokou integrací s jazykem Red, avšak sdílející obecné významy základních konstrukcí a operací:

Gramatická pravidla

Hierarchické výrazy s prakticky neomezenou skladebností. Jejich skladba a sémantika jsou popsány v sekci Pravidla dialektu Parse.

Postupování

Procházení (advancing) vstupní řadou postupným ověřováním shody jednotlivého elementu s gramatickými pravidly - až ke konci řady nebo k výskytu neshody.

Vyhledání

Hledání následného pravidla, k uplatnění za úspěšnou shodou (fetching).

Alternace (v PEG popsáno jako uspořádaný výběr)

V případě výskytu neshody s pravidlem se postupně pokoušet o shodu s následujícími

alternativními pravidly v témže bloku za znakem | ("pipe", "bar", "nebo").

Navrácení

Vrácení vstupu a pravidel na pozici před selháním pravidla (backtracking). Ostatní změny (vedlejší účinky a úpravy vstupu či pravidel) zůstávají.

Kumulativní chování

Parsovací pravidla (zejména Opakování) se vždycky snaží posoudit co nejvíce vstupních dat (possessive matching).

2. Režimy parsování

Procedura *parse* nabízí určitou flexibilitu provedení podporou různých režimů.

2.1. Rozlišení malých a velkých písmen

Implicitně má Parse shodnou sémantiku jako Red a je *case-insensitive*. Rozlišení velkých písmen lze zapnout upřesněním `/case` a zapnout či vypnout klíčovým slovem `case`. Slovo `case` mění režim srovnávání jenom pro následující pravidlo a poté jej vrací zpět nezávisle na úspěch či neúspěch (failure) pravidla.

Syntaxe

```
case <word>

<word> : word! value
```

S hodnotou, na níž odkazuje word se zachází jako s logickým praporkem (flag) podle standardní sémantiky Redu. Logické `true` umožňuje case-sensitivní režim, zatímco logické `false` jej znemožňuje.

2.2. Sběrný režim

Pravidlo `collect` přikazuje, aby `parse` vrátilo blok místo hodnoty `logic!`. Detaily lze nalézt v sekci Vyjmutí.

2.3. Typy vstupu

V závislosti na typu vstupních dat nejsou některá pravidla Parse uplatnitelná nebo se chovají odlišně.

- `any-block!`: porovnávání se sadou znaků nemá žádný význam a vždycky selže;
- `any-string!`: porovnávání s datovým typem nebo sadou typů není podporováno.
- `binary!`: porovnávání s datovým typem nebo sadou typů je podporováno pro hodnoty s kódováním UTF-8; takové porovnáníá je úspěšné, pokud porovnávané části vstupu reprezentují

některou z literálových forem datového typu. Prázdné znaky před tokeny jsou automaticky přeskočeny.

Příklad

```
parse to binary! "3 words: matching by datatype" [number! set-word! 3 word!]
```

3. Pravidla dialektu Parse

Gramatická pravidla v dialektu Parse mohou mít několik forem a obvykle mají vnořenou nebo rekurzivní strukturu. Každé pravidlo je jedním z následujících:

- Dialektem rezervované *klíčové slovo*, volitelně následované argumenty nebo možnostmi (viz níže).
- Hodnota některého z následujících datových typů:
  - `datatype!` nebo `typeset!` - porovnává vstupní hodnotu s jejím typem;
  - `bitset!` - reprezentuje znakovou sadu;
  - `word!` - odkazuje na *well-formed* sub-pravidlo;
  - `lit-word!` nebo `lit-path!` — zavedené zkratky pro parsování vstupních hodnot typu `word!` případně `path!`;
  - `set-word!` - se používá k nastavení slova na aktuální vstupní pozici;
  - `get-word!` - vrátí pozici vstupu k místu, označeném slovem;
  - `block!` - hodnota, která obsahuje libovolný počet sub-pravidel a znaků `|`, jež působí jako oddělovače pro alternativní pravidla;
  - `integer!` - hodnota, která slouží jako počítadlo pro opakování pravidla; dvě následující hodnoty typu `integer!` označují rozsah možných iterací;
  - `paren!` - hodnota, která působí jako únikový mechanizmus dialektu vyhodnocením obsaženého výrazu Red a pokračováním v parsování vstupu; některá klíčová slova Parse používají vrácené hodnoty z výrazu ve shodě se svou vlastní sémantikou;
- Jakákoliv jiná literálová hodnota výše nezmíněná, jež se používá *tak jak je* pro přímé porovnávání se vstupem.

POZN: Parse je konsistentní s Redem v používání volného přirovnání pro porovnávání s literálovými hodnotami.

Každé pravidlo (rule) je charakterizováno podmínkami, při kterých parsování pokročí vstupem a uspěje. Přehled pravidel (jak vyhražená, tak klíčová slova) je tabelárně uveden níže.

Table 1. Přehled pravidel dialektu Parse.

Pravidlo	Categorie	Pokročí	Uspěje
`case`	Režimy parsování	Never	Always

Pravidlo	Categorie	Pokročí	Uspěje
`block!`	Skladba	Depends	Depends
`word!`	Skladba	Depends	Depends
literal value	Přímé posouzení shody	Depends	Depends
`lit-word!`	Přímé posouzení shody	Depends	Depends
`lit-path!`	Přímé posouzení shody	Depends	Depends
`datatype!`	Přímé posouzení shody	Depends	Depends
`typeset!`	Přímé posouzení shody	Depends	Depends
`bitset!`	Přímé posouzení shody	Depends	Depends
`quote`	Přímé posouzení shody	Depends	Depends
`skip`	Přímé posouzení shody	Depends	Depends
`none`	Přímé posouzení shody	Never	Always
`end`	Přímé posouzení shody	Never	Depends
`opt`	Předstih	Depends	Always
`not`	Předstih	Never	Depends
`ahead`	Předstih	Never	Depends
`paren!`	Vyhodnocení výrazu	Never	Always
`set-word!`	Pozicování	Never	Always
`get-word!`	Pozicování	Depends	Always
`integer!`	Opakování	Depends	Depends
`any`	Opakování	Depends	Always
`some`	Opakování	Depends	Depends
`while`	Opakování	Depends	Always
`to`	Hledání	Depends	Depends
`thru`	Hledání	Depends	Depends
`if`	Řízený průběh	Never	Depends
`into`	Řízený průběh	Depends	Depends
`fail`	Řízený průběh	Never	Never
`break`	Řízený průběh	Never	Always
`reject`	Řízený průběh	Never	Never
`set`	Vyjmutí	Depends	Depends
`copy`	Vyjmutí	Depends	Depends
`collect`	Vyjmutí	Depends	Depends
`keep`	Vyjmutí	Depends	Depends

Pravidlo	Categorie	Pokročí	Uspěje
`remove`	Modifikace	Depends	Depends
`insert`	Modifikace	Always	Always
`change`	Modifikace	Depends	Depends

POZN: Všechna pravidla v dále uvedených odstavcích se plně shodují se svými vstupy.

3.1. Skladba

Pravidla `block!` přímo seskupují ostatní prvidla, jsouce prostředkem pro kombinaci. Pravidla `word!` nepřímo odkazují na jiná pravidla, jsouce tak prostředkem abstrakce. Společně tvoří základ mluvnické skladby dialektu Parse.

Na strukturální úrovni je gramatika dialektu Parse složena ze *sekvencí* a *alternativ*.

- Sekvence pravidel je skupina nula či více pravidel, individuelně ukončených *koncem* sekvence. Tato sekvence je úspěšná, dospěje-li parsování (postupně úspěšnou aplikací svých sub-pravidel) ke svému konci. V případě selhání některého sub-pravidla se proces vrátí (backtracks) na počátek neúspěšné sekvence.
- Koncem sekvence pravidel je buď konec vymezujícího bloku nebo *hraniční* znak `|` alternativy.
- Alternativa je volitelná sekvence, kterou se Parse pokusí posoudit v případě, že předchozí (to jest před hranicí `|`) sekvence selže.

3.2. Přímé posouzení shody

Pravidla, popsaná v této části, přímo posuzují shodu vstupnch dat a slouží jako základní stavební bloky pro sestavování složitějších pravidel.

3.2.1. Literálová hodnota

Posouzení shody literálové hodnoty je úspěšné a vede k pokročení zadaným vstupem, je-li posuzovaná literálová hodnota totožná s hodnotou na aktuální pozici.

POZN: Parse používá *volnou komparaci* pro ověření rovnosti. Case-sensitivní režim vynucuje porovnání s rozlišením malých a velkých písmen.

Příklad

```
parse [today is 5-September-2012 #"," 20.3 degrees/celsius][
    'yesterday 'was | 'today 'is 05/09/12 comma 2030e-2 ['degrees/fahrenheit |
'degrees/celsius]
]
```

POZN: Pro porovnávání literálových hodnot, vymezených v dialektu Parse, se jako únikový mechanizmus používá klíčové slovo `quote`.

3.2.2. Datový typ

Ověření shody podle datového typu (datatype) je úspěšné a vede k pokročení vstupem, pokud je ověřovaná hodnota daného typu.

Příklad

```
parse [#a 'bird /is :the word][issue! lit-word! refinement! get-word! word!]
```

POZN: Ověření shody podle datového typu není podporováno pro vstup typu `binary!` a typy typesetu `any-string!`. Pravidla jsou popsána v sekci Typy vstupu.

3.2.3. Typová sada

Ověření shody podle typové sady (type set) je úspěšné a vede k pokročení vstupem, patří-li datový typ vstupní hodnoty k dané typové sadě.

Příklad

```
banner: [
                |
              [_]
             [___]
            [_____]
    Red programming language
    https://www.red-lang.org
]

parse banner [default! series! any-block! any-list! all-word! any-word! any-type! any-
string!]
```

POZN: Ověření shody podle typové sady není podporováno pro vstup typu `binary!` a typy typesetu `any-string!`. Pravidla jsou popsána v sekci Typy vstupu.

3.2.4. Znaková sada

Jsou-li vstupní data typu `any-string!` nebo `binary!` a vstupní hodnotu reprezentuje Unicode Code Point (UCP), který patří k dané sadě znaků, je ověření shody úspěšné a vede k posunu vstupem. Ve všech ostatních případech je ověření shody neúspěšné.

Podrobnosti o vytvoření datové sady lze nalézt v documentaci k datovému typu `bitset!`.

Příklad

```
animal: charset [#"🐓" #"^(1F418)" 128007]
follow: charset "🐑🐏🐐"

parse "🐓 the white 🐑" [follow " the white " animal]
```

POZN: Varianty *lowercase/uppercase* téhož písmena mají různá UCP. Z toho vyplývá, že ověřování shody podle znakové sady je *case-sensitive* bez ohledu na režim parsování.

POZN: Pro vstup typu `binary!` mají význam pouze hodnoty UCP menší než `255`, protože parsování v tomto režimu je *byte-granular*.

3.2.5. `quote`

Působí jako únikový (escape) mechanizmus ze sémantiky dialektu Parse doslovným ověřením shody následující hodnoty. Toto pravidlo je úspěšné a vede k posunu vstupem, jestliže je ověření shody úspěšné.

Syntaxe

```
quote <value>

<value> : literal value to match
```

Příklad

```
parse [[integer!] matches 20][quote [integer!] quote matches quote 20]
```

3.2.6. `skip`

Shoduje se s libobolnou hodnotou a pokročí vstupem. Selže pouze v případě, kdy je pozice vstupu na chvostu (tail), protože tam není co posuzovat.

Příklad

```
parse <🐓> [skip | the beat]
```

3.2.7. `none`

Pravidlo *no-op* nebo *catch-all*, vždy se shoduje a nikdy nevede k postupu vstupem.

Příklad

```
parse reduce [none none][none #[none] ['none | none] none! none]
```

3.2.8. `end`

Pravidlo je úspěšné, je-li pozice vstupu na jeho chvostu (tail) a nikdy nevede k postupu vstupem, protože již není kam se posouvat.

Příklad

```
parse [(□□□□□□)][end | skip [skip | end]]
```

3.3. Předstih

Pravidla s *předstihem* (look-ahead) nabízejí podrobnější nastavení pro ověřování shody, couvání (backtracking) a posun vstupem.

3.3.1. `opt`

Volitelně posuzuje shodu s daným pravidlem, která vede či nevede k posunu vstupem. Pravidlo je vždy úspěšné (== true) bez ohledu na shodu.

Syntaxe

```
opt <rule>

<rule> : Parse rule (option) to match
```

Příklad

```
parse "maybe" [opt "or" "may" opt [#"b" #"e"] opt "not"]
```

3.3.2. `not`

Toto pravidlo je úspěšné, jestliže zadané pravidlo selže a opačně. Nikdy nevede k posunu vstupem, bez ohledu na shodu či neshodu.

Syntaxe

```
not <rule>

<rule> : Parse rule to invert
```

Příklad

```
parse [panama][not 'man not ['plan | 'canal] not word! | skip]
```

3.3.3. ahead

Přednostně řeší shodu s daným pravidlem. Selže v případě selhání pravidla, jinak je úspěšné bez posunu vstupem.

Syntaxe

```
ahead <rule>

<rule> : Parse rule to look ahead
```

Příklad

```
parse [great times ahead][ahead ['great 'times] 'great ahead ['times ahead word!
'ahead] 'times skip]
```

3.4. Vyhodnocení výrazu

Pravidlo typu `paren!` obsahuje libovolný výraz Redu, který se v případě shody vyhodnotí. Toto pravidlo je vždy úspěšné ale nevede k postupu vstupem.

Příklad

```
parse [(did it match?)][
    block! (not matched)
    | (probe 'backtracked) quote (did it match?) (probe 'matched!)
]
```

3.5. Pozicování

Je možné označit aktuální pozici vstupu nebo *přejít* (rewind/fast-forward) na jinou pozici v téže vstupní řadě.

3.5.1. Označení

Pravidlo `set-word!` nastaví slovo k aktuální pozici vstupní řady. Je vždy úspěšné a nikdy nevede k postupu vstupem.

Příklad

```
check: quote (probe reduce [start :failed before after current end])
match: [before: 'this none after:]

parse [match this input][
    start: quote [false start] failed:
    | ahead [skip match] current: ['match 'this 'input] end: check
]
```

3.5.2. Přemístění

Pravidlo `get-word!` nastaví pozici vstupu do místa, označeného zadaným slovem. Je vždy úspěšné a buď posouvá vpřed, zůstává stát nebo posouvá vzad - v závislosti na postavení markeru vzhledem k aktuální pozici vstupu.

Příklad

```
phrase: "and so on and so forth, 'til it gets boring"
goes: skip find phrase comma 2
end: tail phrase

parse phrase [again: "and" :again ['it | :goes] "until the" | :end]
```

POZN: Přemístění pozice do jiné řady než vstupní není dovoleno.

3.6. Opakování

Pravidla níže popsaná působí při posouzení shody jako smyčky nebo iterátory buď určeným počtem opakování nebo až do dosažení neshody.

POZN: Opakovací pravidla mají vlastnické chování a posoudí shodu co možná nejrozsáhlejšího vstupu.

3.6.1. Počet iterací

Provede posouzení shody s daným pravidlem zadaným počtem opakování. Je-li použita skladba *range*, je jako úspěšný akceptován libovolný počet shod v zadaném rozsahu.

Syntaxe

```
<count> <rule>
<count> <count> <rule>

<count> : non-negative integer! value or word! referring to such value
<rule>  : Parse rule to match a specified number of times
```

POZN: Při použití skladby range musí být první celé číslo (spodní mez) menší nebo roven druhému

celému číslu (horní mez).

Příklad

```
tuple:  [2 word!]
triple: [3 skip]
THX:    1138

parse [G A T T A C A][2 3 tuple triple | 0 thx [triple tuple] 1 tuple 0 triple]
```

3.6.2. Rekurze

Pravidla dialektu Parse lze rekurzivně skládat. Úroveň rekurze je limitována hloubkou interní paměti stack.

Příklad

```
ping: [none pong]
pong: [skip ping | end]

parse https://google.com ping
```

3.6.3. any

Porovná dané pravidlo nula či vícekrát (Kleene star), porovnávání končí při výskytu neshody nebo když nedojde k posunu vstupem. Pravidlo je vždy úspěšné.

Syntaxe

```
any <rule>

<rule> : Parse rule to match zero or more times
```

Příklad

```
letter: charset [#"a" - #"z" #"A" - #"Z"]
digit:  charset [#"0" - #"9"]

parse "Wow, 20 horses at 12,000 RPM!" [
    any "Twin ceramic rotor drives on each wheel!"
    "Wow" any [
        comma any space any digit
        space any letter any [not comma skip]
    ]
]
```

3.6.4. `some`

Porovná dané pravidlo jednou či vícekrát (Kleene plus), porovnávání končí při výskytu neshody nebo když nedojde k posunu vstupem. Pravidlo je úspěšné při nalezení alespoň jedné shody.

Syntaxe

```
some <rule>

<rule> : Parse rule to match one or more times
```

Příklad

```
parse [
    skidamarink a dink a dink
    skidamarink a doo
][
    some [
        some none 'skidamarink
        [some ['a 'dink] | 'a 'doo]
    ]
]
```

3.6.5. `while`

Opakovaně porovnává dané pravidlo. Zastaví se pouze po selhání pravidla. Vždycky úspěšné.

POZOR: Jestliže pravidlo neselže, uvízlo `while` v nekonečné smyčce.

Syntaxe

```
while <rule>

<rule> : Parse rule to match repeatedly
```

Příklad

```
parse [throw for a loop][
    while [word! | (print "failed and backtracked on matching the end") [not end]
:explicit failure]
    | [while none] :infinite loop
]
```

3.7. Hledání

Pravidla této skupiny (search) hledají určený vzor procházejíc vstupem až k výskytu shody.

3.7.1. to

Opakovaně se pokouší nalézt shodu s daným pravidlem až k dosažení úplné shody. Pokud řečené pravidlo selže, postoupí se vstupem o jeden element, což se počítá jako částečná shoda. V případě úplné shody je pozice vstupu nastavena do čela (head) posuzované části. Succeeds if rule match succeeded.

Syntaxe

```
to <rule>

<rule> : Parse rule (pattern to put input position at)
```

Příklad

```
matrix: #{
    416C6C20492073656520697320626C6F6E6465
    2C206272756E657474652C201337526564C0DE
}

parse matrix [
    to #{FACEFEED}
    | to #{1337} #{1337} start: to #{C0DE} end: (print to string! copy/part start end)
2 skip
]
```

3.7.2. thru

Opakovaně se pokouší nalézt shodu s daným pravidlem až k dosažení úplné shody. Pokud řečené pravidlo selže, postoupí se vstupem o jeden element, což se počítá jako částečná shoda. V případě úplné shody je pozice vstupu nastavena do chvostu (tail) posuzované části. Succeeds if rule match succeeded.

Syntaxe

```
thru <rule>

<rule> : Parse rule (pattern to advance thru)
```

Příklad

```
parse 'per/aspera/ad/astra [thru 'aspera ad: to 'astra thru end (probe ad)]
```

3.8. Řízený průběh

Pravidla této skupiny (control flow) reguluje provedení procesu Parse smyčkami (Opakování), změnou vstupu, předčasným ukončením a podmíněným porovnáním.

3.8.1. `if`

Podmíněná shoda - je úspěšná, když se daný výraz Redu vyhodnotí na true. Nikdy se neposune vstupem.

Syntaxe

```
if <expression>

<expression> : paren! expression
```

Příklad

```
parse [4 8 15 16 23 42][
    some [mark: skip if (any [even? probe mark/1 find [15 23] first mark])]
]
```

3.8.2. `into`

Je-li datový typ hodnoty na aktuální pozici vstupu podporován dialektem Parse, pravidlo `into` dočasně přemístí vstup k této hodnotě a posoudí ji z hlediska daného pravidla. Po skončeném posouzení se vstup vrátí do původní pozice a parsování pokračuje za shodující se hodnotou.

Syntaxe

```
into <rule>

<rule> : block! rule or word! that refers to such rule
```

Příklad

```
rule: [some [word! | into rule]]

parse [we [need [to [go [deeper]]]]] rule
```

3.8.3. `fail`

Tento příkaz vynutí neshodu s pravidlem, pokud je umístěn na jeho konci. Nikdy neuspěje ani nepokročí vstupem.

Příklad

```
parse foo@bar.baz [["quux" | some fail | "foo"] "@" [fail] | thru "bar.baz"]
```

3.8.4. `break`

Vynutí okamžitou shodu aktuálního pravidla `block!`. Ukončí průběh smyčky, je-li použito v nejvyšší úrovni opakovacího pravidla. Vždy uspěje a nikdy nepokročí vstupem.

Příklad

```
parse [break away from everything][some [break] 0 1 [break] [2 [break] | 3 word!
[break] skip]]
```

3.8.5. `reject`

Vynutí okamžitou neshodu aktuálního pravidla `block!`. Ukončí průběh smyčky, je-li použito v nejvyšší úrovni opakovacího pravidla. Nikdy neuspěje a nepokročí vstupem.

Příklad

```
parse quote (I made a choice that I regret) [
    any [reject now] some [5 word! what: reject I see] is
    | :what 'I [[reject get] | skip]
]
```

3.9. Vyjmutí

Vyjímací (extraction) pravidla kopírují shodné hodnoty ze vstupních řad.

3.9.1. `set`

Přiřadí dané slovo první hodnotě v konformní části vstupu.

POZN: Slovu je přiřazena hodnota `none`, pokud porovnávané pravidlo neposunulo pozici vstupu.

POZN: Pro vstup typu `binary!` je slovo nastaveno na hodnotu typu `integer!` mezi `0` a `255`.

Syntaxe

```
set <word> <rule>

<word> : word! value to set
<rule> : Parse rule
```

Příklad

```
parse "□□□" [set hole ahead [2 skip] set donut [to end]]
```

3.9.2. copy

Přiřadí dané slovo kopii shodující se části vstupu.

NOTE	Pokud porovnávané pravidlo nepokročilo vstupem, je slovu přiřazena prázdá řada (series) stejného typu jako vstup.

Syntaxe

```
copy <word> <rule>

<word> : word! value to set
<rule> : Parse rule
```

Příklad

```
parse [Huston do you copy?][2 word! copy Huston [2 word!] copy we opt "have a
problem"]
```

3.9.3. collect

Shromáždí konformní hodnoty, které jsou označeny klíčovým slovem `keep`. Uspěje, uspěje-li dané pravidlo - postupujíc za konformní (matched) část vstupu.

Pravidlo `keep` uspěje, uspěje-li poskytnuté pravidlo - vkládajíc konformní hodnoty do bloku, vymezeného pravidlem `collect`.

POZN: Použití klíčového slova `keep` bez souvislosti s pravidlem `collect` je zapovězeno.

Syntaxe

```
collect <rule>
collect set <word> <rule>
collect into <word> <rule>
collect after <word> <rule>

<word> : word! value
<rule> : Parse rule
```

Hodnoty jsou implicitně vkládány do chvostu (tail) bloku. Toto chování lze změnit níže popsanými volbami.

Table 2. Volby pro pravidlo `collect`.

Volba	Popis
`set`	Přiřadí danému slovu blok shromážděných (collected) hodnot.
`into`	Vloží shromážděné hodnoty do řady (series), označené slovem, přenese index řady do jejího čela.
`after`	Vloží shromážděné hodnoty do řady (series), označené slovem, přemístí index řady za vloženou část.

- Je-li v kterémkoli pravidlu použit pokyn `collect` bez volby `into` či `after`, vráti funkce `parse` blok shromážděných hodnot (viz Režimy parsování); je-li pokyn `collect` použit s volbou `set`, vrátí funkce `parse` hodnotu typu `logic!` jako obvykle.
- První použití pokynu `collect` alokuje nový blok, který je vrácen funkcí `parse`, každé další použití pokynu `collect` alokuje hodnoty (blok) na chvostu (tail) předchozího bloku; pokyn `collect` s volbou `into` či `after` použije již vytvořený buffer spíše než alokaci nového bloku.

Syntaxe pro `keep`:

```
keep <rule>
keep pick <rule>
keep <expression>
keep pick <expression>

<rule>       : Parse rule
<expression> : paren! expression
```

- Jestliže porovnávané pravidlo nepokročilo vstupem, příkaz `keep` nic nezadrží.
- Jestliže pravidlo vyčlenilo jedinou hodnotu - tato je zadržena (is kept). Je-li `keep` následováno pravidlem `copy`, potom je posuzovaná hodnota přiřazena stejnému typu z typesetu `series` jako vstup.
- Jestliže pravidlo vyčlenilo více hodnot, jsou tyto seskupeny do objektu stejného typu jako vstup; při volbě `pick` nejsou hodnoty seskupeny ale uchovány odděleně.
- Je-li příkaz `keep` použit s výrazem typu `paren!`, je výsledek jeho vyhodnocení uchován tak, jak je.

Example

```
fruit: charset [#"^(1F346)" - #"^(1F353)"]
plate: "tropical stuff: □□ and other healthy food: □□□"

parse plate [
    collect [
        keep (quote fruits:) collect [some [keep fruit | skip] fail]
        | keep (quote vegetables:) collect [to [#"□" | "Pickle Rick!"] keep pick [to
end]]
    ]
]
```

3.10. Modifikace

Akce parse může modifikovat své vstupy vložením nových hodnot a odebrat či změnit odpovídající části vstupu.

3.10.1. `remove`

Buď odebere část vstupu, konformní s daným pravidlem nebo odebere vstup mezi aktuální a zadanou pozicí; poté zachová aktuální vstupní pozici.

POZN: Odebírání hodnot je "forward-consuming" operace. Jinými slovy, počítá se jako shoda, přesto že nedojde k pokročení vstupem.

Syntaxe

```
remove <rule>
remove <word>

<rule> : Parse rule
<word> : input postion
```

Příklad

```
parse [remove me <and me also> "but leave me be"][some [remove word!] mark: to string!
remove mark skip]
```

3.10.2. `insert`

Vloží literálovou hodnotu nebo výsledek vyhodnocení výrazu buď do aktuální nebo zadané (marked) pozice. Akce je vždy úspěšná a pokročí vstupem za místo vložení, pokud byla provedena v aktuální pozici, jinak je zachována vstupní pozice.

Syntaxe

```
insert <value>
insert <expression>

insert <word> <value>
insert <word> <expression>

insert only <value>
insert only <expression>

insert only <word> <value>
insert only <word> <expression>

<word>       : input position
<value>      : literal value
<expression> : paren! expression
```

Je-li literálová hodnota typu `word!`, použije se hodnota, na níž slovo odkazuje. Volba `only` prosadí sémantiku `insert/only`.

Příklad

```
ikea: [assembly]
here: tail ikea

parse ikea [
    insert only here [□□ □□]
    insert only (load "[manual]")
    word!
    insert ikea [some]
    block!
    insert [required]
]
```

3.10.3. change

Mění konformní (matched) část vstupu na literálovou hodnotu nebo na výsledek vyhodnocení výrazu. Navíc, může změnit část vstupu mezi aktuální a označenou pozicí. Byla-li změna provedena v aktuální pozici, je úspěšná a posune vstup za upravovanou část; v opačném případě je vstupní pozice zachována.

Syntaxe

```
change <rule> <value>
change <rule> <expression>

change <word> <value>
change <word> <expression>

change only <rule> <value>
change only <rule> <expression>
change only <word> <value>
change only <word> <expression>

<rule>       : Parse rule
<word>       : input position
<value>      : literal value
<expression> : paren! expression
```

Je-li literálová hodnota typu `word!`, použije se její odkazovaná hodnota. Volba `only` prosadí sémantiku `change/only`.

Příklad

```
parse [some things never change][
    change none (quote and) 2 skip mark: to end change only mark [do]
]
```

4. Události procesu parse

Dialekt Parse je implementován jako pushdown automaton (PDA), využívající paměti typu stack. Při každé změně stavu emituje *událost* (event, hodnota typu `word!`), která informuje uživatele o parsovacím procesu. Interakce mezi událostmi a interním stavem aktivity parse je dosaženo upřesněním `/trace` a "callback" funkcí (viz další odstavec).

Níže je uveden seznam všech událostí s podmínkami, které je vyvolávají (štos = stack):

Table 3. Seznam událostí Parse.

Event	Description
`push`	Poté co je pravidlo vloženo na štos.
`pop`	Předtím než je pravidlo staženo ze štosu.
`fetch`	Předtím než je přiřazeno nové pravidlo.
`match`	Poté co byla nalezena shoda hodnoty s pravidlem.
`iterate`	Po započetí nového iteračního kola (viz Opakování).
`paren`	Po vyhodnocení výrazu typu `paren!`.
`end`	Po dosažení konce vstupu.

5. Extra funkce

Vstupním bodem do dialektu Parse je nativní funkce `parse`, která přijme vstupní **objekt** typu series!, **blok** s pravidly a která podporuje dodatečná upřesnění (refinements):

Table 4. `parse` refinements.

Refinement	Description
`/case`	Umožnit case-sensitive režim.
`/part`	Limitovat parsování určenou délkou nebo pozicí vstupu.
`/trace`	Spolupůsobit s rozhraním PDA přes zadaný *callback*.

Při použití upřesnění `/trace` musí být určena funkce "zpětného volání " (callback, hodnota typu `function!`) s následující specifikací:

Table 5. Callback function specification.

Argument	Type	Description
`event`	`word!`	Některá z Události procesu parse.
`match?`	`logic!`	Výsledek poslední shody
`rule`	`block!`	Aktuální pravidlo v aktuální pozici.
`input`	`series!`	Vstupní objekt ze sady series! v aktuální pozcici
`stack`	`block!`	Interní `stack` pravidel Parse.

Callback funkce musí vrátit hodnotu typu `logic!`, jež indikuje, zda se má v parsování pokračovat (`true`) či nikoli (`false`).

Za účelem ladění je implicitně poskytnuto zpětné volání (callback) `on-parse-event` a jeho `parse-trace` wrapper.

6. Implementační poznámky

V této části jsou stručně zmíněny některé údaje o návrhu a implementaci dialektu Parse.

6.1. Volné přirovnání

Jak již bylo dříve zmíněno, Parse používá volné přirovnání (loose comparison) pro porovnávání literálových hodnot, což je konsistentní s Redem.

Příklad

```
parse [I'm 100% <sure>][quote :I'M 1.0 "sure"]
```

6.2. Prostý formát pravidla

Do jisté míry podporuje Parse prostý (flat) formát, při němž jsou pravidla psána lineárně jako výrazy s proměnnou aritou, spíše než s pomocí vnořených bloků.

Příklad

```
parse [on the count of three 1 2 3][collect set stash keep pick to ahead some 1 3
integer! remove any skip]
```

6.3. Otevřené problémy

Nevyřešené chyby a inkozistence návrhu, související s dialektem Parse jsou vypsány níže:

Table 6. Nevyřešené problémy.

Affected rules	Description	Tickets
`remove <position>`	The case where position comes after the current one is not handled.	#4199
`break`, `reject`	Preemptive break of [Repetition] rules.	#4193
`fail`, `break`, `reject`	Design of some [Control flow] rules is not finalized.	#3478, #3398
`lit-word!`, `lit-path!`	Case-sensitive comparison is not handled properly.	#3029

7. Poznámka k překladu

Kromě anglických i obecně cizích slov jsou v překladu použity jisté rádoby ekvivalentní výrazy, na něž chci zde upozornit:

- series! - řada - vstup - vstupní objekt, náležející do typové sady series! případně název pravidla Parse
- matched - konformní - shodující se s pravidlem
- top-level-rule ? - řídící pravidlo