

# Operátory v zápisu algoritmů v XML

## Rozdíly oproti dnešním systémům

Existuje několik rozdílů, které jsou popsány dále. Jde zejména o zázračné odstranění priority operátorů nad kterou si čtenář XML souboru už nemusí lámat hlavu, pak třeba taky natrefí na operace s mnoha operandy, které jsou zobecněnou formou binárních operátorů. Dále se mnoho operací původně zapisovaných jako funkce zapisuje po novém v operátorovém tvaru, ale aby nebyl nový systém jenom o zjednodušování, nalezneme zde také poněkud složitější zápis paralelismu.

### *Neexistuje prioritita operátorů*

Algoritmy zapsané ve formě XML se odlišují od těch co jsou zapsány standardní syntaxí v tom, že u XML dokumentu se nemusí definovat priority operátorů. Tato vlastnost vyplývá ze struktury XML dokumentu, která je stromová a pokud v ní chceme reprezentovat výraz (nebo algoritmus výpočtu), je nejvhodnější do XML ukládat sémantický strom. V tomhle stromu jsou jednotlivé podvýrazy reprezentovány větvemi stromu podle toho jak byly ve zdrojovém textu závorky nebo jak byla definována prioritita operátorů v lexikálním analyzátoru.

### *Zobecnění operátorů na n-ární*

Většina binárních operátorů reprezentuje operaci, která je ve své podstatě n-ární. Tyto operátory můžeme identifikovat dle toho, že nerozlišují role operandů (tj. třeba sčítání i násobení sem patří, ale indexace pole už ne). Zobecnění spočívá v tom, že se binární sémantický strom výrazu přepíše na n-ární tak, že pokud obsahuje uzel s operací stejnou operaci jako svůj operand, tak se operandy druhého uzlu přesunou do uzlu prvního. Příkladem - původní vstup:

```
<add>
  <var>a</var>
  <add>
    <var>b</var>
    <var>c</var>
  </add>
</add>
```

V zobecněné formě operátoru můžeme pak zapsat operaci následovně:

```
<add>
  <var>a</var>
  <var>b</var>
  <var>c</var>
</add>
```

Zápis v zobecněné formě je výhodnější z hlediska čistější sémantiky programu, protože u uvedeného příkladu jde o sečtení tří proměnných a nepotřebujeme znát které dvě se sčítají prioritně.

### *Funkce, operátor nebo metoda ?*

Známe alespoň dva případy, kde se operace reprezentují funkcemi namísto operátorů. Jsou to operace s řetězci (podřetězec, vyhledání pozice) a složitější početní operace (mocniny, odmocniny, trigonometrické funkce).

Po sémantické stránce by bylo výhodnější zapisovat tyto funkce jako operátory, protože je to zase čistější - nezáleží totiž na tom, jestli je sinus operátor nebo je implementován jako funkce, význam je stejný - spočtení sinu ze nějaké hodnoty.

Z tohoto důvodu budeme specifikovat dané operace pomocí zápisu pro operátory namísto volání funkce, které se tímto nesnažíme odstranit celkově, jenom si ho ponecháváme pro odůvodněné případy použití (zejména jde o uživatelské funkce).

Třetím přístupem k řešení problému „funkce vs. operátor“ je použití objektivně orientovaného programování, kde jsou typy reprezentovány pomocí tříd které obsahují příslušné metody pro práci s hodnotami.

## Zobecnění přes pole

V našem obecném zápisu sémantiky výrazů můžeme zavést pravidlo, že pokud operátor neočekává pole jako parametr, ale přece je tam hodnota typu pole, tak se provede paralelizace operace následovně:

- typ výsledku bude pole s prvky typu jaký by byl původně očekáván
- parametr, který není pole bude konstantní přes všechny iterace
- pokud je více parametrů zapsáno jako pole na místě očekávané skalární hodnoty, tak musí sedět počet prvků aby se dali vytvořit n-tice pro jednotlivé iterace

Příklad pro spočtení délky 4 řetězců:

```
<!-- původní (sekvenční) algoritmus pro
      array(strlen($a),strlen($b),strlen($c),strlen($d)) -->
<array>
  <strlen>
    <var>a</var>
  </strlen>
  <strlen>
    <var>b</var>
  </strlen>
  <strlen>
    <var>c</var>
  </strlen>
  <strlen>
    <var>d</var>
  </strlen>
</array>

<!-- nový (paralelní) algoritmus pro
      array(strlen($a),strlen($b),strlen($c),strlen($d)) -->
<strlen>
  <var>a</var>
  <var>b</var>
  <var>c</var>
  <var>d</var>
</strlen>
```

Příklad pro spočtení délky skupiny řetězců:

```
<!--
  {
    $tmp = array();
    foreach($pole as $key => $value) $tmp[$key] = strlen($value);
    return $tmp;
  }
-->
<strlen>
  <var>pole</var>
</strlen>
```

Zobecnění podle těchto pravidel zavádí prvky paralelního programování do algoritmů, jelikož je možné jednotlivé iterace provádět současně (pokud to teda daná architektura umožňuje).

## Logické operátory

Logické operátory jsou operátory pro práci s pravdivostní hodnotou. Základní logické operace jsou následovny:

- logický součin, AND
- logický součet, OR
- logický exkluzivní součet, XOR
- logická negace, NOT

Operace logického součinu, součtu i exkluzivního součtu jsou n-ární operace, negace je unární. Vstupní operandy musí být binárního typu (boolean) a výsledek bude znova binární hodnota.

### Zkrácené vyhodnocování

Pro úplnost můžeme specifikovat pomocný atribut, který dovoluje ovládat chování překladače (generátoru kódu) v souvislosti se zkráceným vyhodnocováním. Výchozí hodnota je nastavena na povolení zkráceného vyhodnocování, tudíž následující dva zápisy jsou zhodné:

```
<!-- short evaluation: ON -->
<and>
:

<and eval="short">
:
```

Vypnutí zkráceného vyhodnocování (a zapnutí úplného) se provede takhle:

```
<!-- short evaluation: OFF -->
<and eval="full">
:
```

Platnost explicitního nastavení zkráceného nebo úplného je pro všechny pod-uzly výpočetního stromu výrazu (až do následujícího předefinování). Vrácení na výchozí hodnotu lze provést pomocí:

```
<!-- evaluation: Automatic -->
<and eval="auto">
:
```

Pozn.: Zkrácené vyhodnocování se týká jen operací AND a OR.

## Operátor výběru

Tento operátor je taky známý jako ternární operátor „?:“, u nás nese identifikaci jako CHOOSE. Výsledkem je jedna ze dvou hodnot (druhý nebo třetí operand) v závislosti na logické hodnotě prvního operandu.

Příklad:

```
<!-- ($a < $b) ? $a : $b ... menší hodnota -->
<choose>
<lt>
  <var>a</var>
  <var>b</var>
</lt>
<var>a</var>
<var>b</var>
</choose>
```

Vstupy - první operand musí nést pravdivostní hodnotu, druhý a třetí by měl být stejného typu, typu, který zároveň určuje typ výsledku.

V případě že typy nesedí, se musí vyhlásit varování nebo chyba, v závislosti na tom, jestli jsou typy slčitelné nebo ne (celé číslo jde sloučit s číslem v plovoucí desetinné tečce - výsledkem je obecnější typ, tj. plovoucí desetinná tečka).

## Přiřazovací operátor

Pro přiřazení hodnoty do proměnné se používá přiřazovací operátor označený jako ASSIGN. Očekává 2 nebo více operandů, ze kterých mají první a poslední speciální roli:

- poslední - je hodnota která se přiřazuje do ostatních operandů
- první - tato proměnná bude vrácena jako výsledek operace

Do všech operandů kromě posledního musí být možnost zapisovat hodnotu, jinak se musí vyhlásit chyba při překladu. Příklad:

```
<!-- $b = $a = 3; ... binárně -->
<assign>
  <var>b</var>
  <assign>
    <var>a</var>
    <int xmlns="Const">3</int>
  </assign>
</assign>

<!-- $b = $a = 3; ... n-árně -->
<assign>
  <var>b</var>
  <var>a</var>
  <int xmlns="Const">3</int>
</assign>
```

## Relační operátory

Relační operátory porovnávají dvě nebo více hodnot. Tři základní relační operátory jsou:

- menší než, LT
- větší než, GT
- rovný, EQ

Ke všem těmto operátorům pak existují jejich komplementy:

- větší nebo rovný, GTE
- menší nebo rovný, LTE
- nerovný, NEQ

Operandy mohou být obecně různých typů, ale v rámci jedné operace musí být typ zachován, konkrétněji teda existují relační operace mezi číselnými, znakovými a řetězcovými hodnotami. Výsledkem relačních operací je vždy pravdivostní hodnota která vyjadřuje pravdivost dané relace, daného tvrzení. Relační operátory mohou být zobecněny na n-ární, protože následní dva výrazy jsou shodné a lze mezi nimi dělat převody:

$$A < B < C$$
$$(A < B) \wedge (B < C)$$

Příklad na aplikaci zjednodušovacího pravidla ukazuje vznik jednodušeji zapsané operace:

```
<!-- původní strom -->
<and>
  <lt>
    <var>a</var>
    <var>b</var>
  </lt>
  <lt>
    <var>b</var>
    <var>c</var>
  </lt>
</and>

<!-- zjednodušený strom -->
<lt>
  <var>a</var>
  <var>b</var>
  <var>c</var>
</lt>
```

## Aritmetické operátory

Aritmetické operátory reprezentují početní operace, které jsou:

- záporná hodnota, NEG
- sčítání, ADD
- odečítání, SUB
- násobení, MUL
- dělení, (obecně DIV)
  - celočíselné, IDIV (nebo div out="int")
  - v pohyblivé řádové čárce, FDIV (nebo div out="float")
- zbytek po celočíselném dělení, MOD

Vstupními hodnotami, operandy, jsou ve všech případech jediné hodnoty číselného typu. Operátor záporné hodnoty je unární operátor, sčítání a násobení jdou zobecnit na n-ární a zbytek (odečítání, dělení a zbytek po dělení) jsou binární operátory, a nepůjde je zobecnit protože rozlišují role operandů. Výsledek je znova číselného typu a podle vstupních operandů a operace může být buď celočíselný nebo v pohyblivé řádové čárce.

V aritmetických operacích existují struktury, které lze syntakticky upravit se zachováním sémantiky, např. z:

$$- A + B$$

na:

$$B - A$$

V zápisu jako XML je to pak úprava z:

```
<add>
  <neg>
    <var>A</var>
  </neg>
  <var>B</var>
</add>
```

na:

```
<sub>
  <var>B</var>
  <var>A</var>
</sub>
```

## Operátory inkrementu a dekrementu

Existují 4 typy těchto operátorů, které se rozlišují podle toho jestli:

- inkrementují (zvyšují) hodnotu
- dekrementují (znižují) hodnotu

A nezávisle na změně hodnoty pak:

- vracejí původní hodnotu (post operátory)
- vracejí novou - změněnou - hodnotu (pre operátory)

Identifikátory našich operátorů jsou (spojení s PRE jsou jenom aliasy):

- INC (PREINC)
- POSTINC
- DEC (PREDEC)
- POSTDEC

Operátory je možné aplikovat nejen na číselné typy ale i výčtové a jsou unární nebo binární - v případě že je uveden druhý operand, označuje velikost inkrementu resp. dekrementu a vztahují se na něj nějaká omezení (třeba že celočíselný operand nebo operand výčtového typu nelze inkrementovat necelým, float, číslem).

## Bitové operátory

Mezi operace které souvisí s jednotlivými bity celého čísla se řadí:

- bitový součin, BAND
- bitový součet, BOR
- bitová negace, BNOT
- bitový exkluzivní součet, BXOR
- bitový posun vlevo, BSL
- bitový posun vpravo, BSR
- bitová rotace vlevo, BRL
- bitová rotace vpravo, BRR

Operace pro rotaci bitů mají jeden až dva volitelné parametry - a to:

- počet bitů které se rotují
  - umožňuje rotovat dokolečka třeba jenom 4 bity
- posunutí bitů
  - umožňuje rotovat třeba horní 4 bity z bajtu

Operace BNOT je unární, BAND, BOR, BXOR jsou n-ární. BSL a BSR požadují vstupní hodnotu a číslo vyjadřující počet bitů o které se hodnota posouvá, BRL a BRR nápodobně, s možností specifikace šířky nebo šířky a začátku oblasti rotace.

Pozn.: pro bitové operace se doporučuje alokovat UNSIGNED celočíselnou proměnnou se zadanou velikostí, aby se zajistila sémantická jednoznačnost. V jiném případě je vhodné generovat alespoň varování u překladu.

## Operace pro práci s výčtovými typy

Jsou to vlastně operace přetypování - v praxi totiž potřebujeme převádět:

- výčtový typ na pořadové číslo hodnoty, ORD
- číselnou hodnotu na hodnotu z výčtového typu, ENUM

V našem systému je výsledný typ operace ENUM určen podle typu požadované hodnoty v místě použití.

Příklad použití:

```
<!-- definice výčtového typu -->
<typedef id="text/justify" xmlns="Type">
  <enum xmlns="Type">
    <option>LEFT</option>
    <option>RIGHT</option>
    <option>CENTER</option>
  </enum>
</typedef>

<!-- použití operátoru enum pro určení hodnoty CENTER -->
<pad>
  <string xmlns="Const">Short</string>
  <string xmlns="Const"> </string>
  <int xmlns="Const">80</int>
  <enum>2</enum>
  <true xmlns="Const" />
</pad>
```

## Řetězcové operátory

Jak již bylo v úvodu naznačeno, operace s řetězci jsou v dnešních programovacích jazycích reprezentovány pomocí funkcí, v některých objektově orientovaných jazycích (C++, JavaScript nebo JScript) pak pomocí metod třídy reprezentující řetězcový typ.

V našem systému jsou řetězcové typy zahrnuty do základních datových typů, obecně to nejsou objekty a tudíž metody odpadají. Zůstává možnost použít funkce nebo operátory a volíme cestu operátorů - protože je potřeba standardizovat operace, zahrnout je do základní sémantiky.

Operátory které jsou určeny pro práci s řetězci jsou:

- konkatenace, spojení řetězců, CONCAT
- podřetězec, SUBSTR
- pozice podřetězce STRPOS
- pozice podřetězce z konce, STRRPOS
- délka řetězce, STRLEN (LENGTH)
- indexace znaku, INDEX
- reverzace, otočení řetězce REVERSE
- opakování, REPEAT
- nahrazení, REPLACE, IREPLACE
- doplnění na určitou délku, PAD
- převod na malá písmena, LOWERCASE
- převod na velká písmena, UPPERCASE
- uříznutí prázdných znaků, TRIM, LTRIM, RTRIM

Aritu a typy vstupů a výstupů definuje následovní tabulka:

Operátor	Arita	Vstupy	Výstup
concat	n	string <sup>n</sup>	string
substr	2-3	string <i>source</i> int <i>start</i> [int] [ <i>length</i> ]	string
strpos	2	string <i>source</i>	optional int
strlen/length	1	string	int
index	2	string <i>source</i> int <i>index</i>	char
reverse	1	string	string
repeat	2	string <i>piece</i> int <i>count</i>	string
replace, ireplace	3	string <i>source</i> string <i>pattern</i> string <i>replacement</i>	string
pad	5	string <i>source</i> string <i>fill</i> int <i>length</i> enum <i>justify</i> (L,R,C) bool <i>cutLonger?</i>	string
lowercase, uppercase	1	string	string
trim, rtrim, ltrim	1	string	string

## Operátory pro práci s poli

Pro práci s poli existuje v našem systému několik různých operátorů - některé z nich se opět odvodili od existujících funkcí nebo metod:

- indexace, INDEX
- dotaz na velikost pole, LENGTH/COUNT
- sestavení pole, ARRAY
- vytvoření z řetězce, EXPLODE
- zřetězení pole, IMplode/JOIN

Operace EXPLODE sice není čistě operátorem který má operand typu pole, ale pole naopak vrací, takže je zahrnuta v této skupině jako její inverzní funkce IMplode. Zvláštností u IMplode je, že je to unární i binární operace - pokud se vynechá 2. parametr, tak se pole řetězců spojí bez spojovacího řetězce.

Příklady použití:

```
<!-- $pole[$i] -->
<index>
  <var>pole</var>
  <var>i</var>
</index>

<!-- count($pole) -->
<count>
  <var>pole</var>
</count>

<!-- array($a,$b,$c,$d) -->
<array>
  <var>a</var>
  <var>b</var>
  <var>c</var>
  <var>d</var>
</array>

<!-- explode('.', $ipstr) -->
<explode>
  <var>ipstr</var>
  <string xmlns="Const">.</string>
</explode>

<!-- implode('.', $pole) -->
<implode>
  <var>pole</var>
  <string xmlns="Const">.</string>
</implode>
```

Pozn.: EXPLODE má volitelně ještě třetí parametr, který říká kolik maximálně prvků může obsahovat výstupní pole.