Kontext

tedy co bylo předneseno a bude použito...

- Automaty různých typů (tedy různé síly): konečný a zásobníkový automat a Turingův stroj,
- gramatiky (různých druhů jako z udělání disponující stejnou silou jako některé automaty),
- fakta, že jazyk můžeme buďto rozpoznat automatem, nebo nagenerovat gramatikou.
- První cíl: Pochopit, jak funguje překladač.
- Vstupní (zdrojový) text umíme rozsekat na atomy konečným automatem.

Kontext II

- Programovací jazyk jsme schopni popsat bezkontextovou gramatikou,
- bezkontextová gramatika nám umožňuje popsat program jako strom (podobný tomu, co se učil v 1. ročníku u aritmetických výrazů).
- Podle tohoto (tzv. syntaktického) stromu bychom byli schopni program interpretovat.
- Ale jak syntaktický strom postavit? Deterministický zásobníkový automat je příliš slabý a nedeterministický neumíme snadno naprogramovat.
- Proto vyrobíme automaty rozpoznávající jazyky "někde mezi".

Pohled inženýrský

- Existují automaty, které jsou schopné rozpoznávat regulární výrazy (flex),

Pohled inženýrský

- Existují automaty, které jsou schopné rozpoznávat regulární výrazy (flex),
- existují automaty schopné rozpoznat i některé bezkontextové gramatiky (bison).

Pohled inženýrský

- Existují automaty, které jsou schopné rozpoznávat regulární výrazy (flex),
- existují automaty schopné rozpoznat i některé bezkontextové gramatiky (bison).
- Použijeme tudíž tuto osvědčenou dvojici nástrojů. Prvním rozbijeme vstup na tokeny, druhý provede syntaktickou analýzu (ze které dovedeme postavit syntaktický strom).

Pohled inženýrský

- Existují automaty, které jsou schopné rozpoznávat regulární výrazy (flex),
- existují automaty schopné rozpoznat i některé bezkontextové gramatiky (bison).
- Použijeme tudíž tuto osvědčenou dvojici nástrojů. Prvním rozbijeme vstup na tokeny, druhý provede syntaktickou analýzu (ze které dovedeme postavit syntaktický strom).
- Při troše štěstí (u netypových jazyků) je sémantická analýza prázdná a máme hotovo.

Pohled inženýrský

- Existují automaty, které jsou schopné rozpoznávat regulární výrazy (flex),
- existují automaty schopné rozpoznat i některé bezkontextové gramatiky (bison).
- Použijeme tudíž tuto osvědčenou dvojici nástrojů. Prvním rozbijeme vstup na tokeny, druhý provede syntaktickou analýzu (ze které dovedeme postavit syntaktický strom).
- Při troše štěstí (u netypových jazyků) je sémantická analýza prázdná a máme hotovo.
- Při troše smůly u bisonem nehneme.

Pohled inženýrský

- Existují automaty, které jsou schopné rozpoznávat regulární výrazy (flex),
- existují automaty schopné rozpoznat i některé bezkontextové gramatiky (bison).
- Použijeme tudíž tuto osvědčenou dvojici nástrojů. Prvním rozbijeme vstup na tokeny, druhý provede syntaktickou analýzu (ze které dovedeme postavit syntaktický strom).
- Při troše štěstí (u netypových jazyků) je sémantická analýza prázdná a máme hotovo.
- Při troše smůly u bisonem nehneme.
- Flex bychom byli schopni oběhnout (vlastním automatem), bisona zatím ne.

Vnitřnosti překladače

- Lexikální analýza: Rozbije vstup na tokeny. Lze ji implementovat opakovaným voláním funkce, která pokaždé vrátí údaj o dalším tokenu, na konci vstupu vrátí nárazníkový literál.

Vnitřnosti překladače

- Lexikální analýza: Rozbije vstup na tokeny. Lze ji implementovat opakovaným voláním funkce, která pokaždé vrátí údaj o dalším tokenu, na konci vstupu vrátí nárazníkový literál.
- Syntaktická analýza: Ze vstupních tokenů postaví syntaktický strom (tedy sdělí, jak program syntakticky vznikl).

Vnitřnosti překladače

- Lexikální analýza: Rozbije vstup na tokeny. Lze ji implementovat opakovaným voláním funkce, která pokaždé vrátí údaj o dalším tokenu, na konci vstupu vrátí nárazníkový literál.
- Syntaktická analýza: Ze vstupních tokenů postaví syntaktický strom (tedy sdělí, jak program syntakticky vznikl).
- Sémantická analýza: K syntaktickému stromu dodá informace o tom, co operace znamenají, případně strom přestaví.

Vnitřnosti překladače

- Lexikální analýza: Rozbije vstup na tokeny. Lze ji implementovat opakovaným voláním funkce, která pokaždé vrátí údaj o dalším tokenu, na konci vstupu vrátí nárazníkový literál.
- Syntaktická analýza: Ze vstupních tokenů postaví syntaktický strom (tedy sdělí, jak program syntakticky vznikl).
- Sémantická analýza: K syntaktickému stromu dodá informace o tom, co operace znamenají, případně strom přestaví.
- Generátor interkódu: Ze sémantického stromu postaví předobraz binárního kódu.

Vnitřnosti překladače

- Lexikální analýza: Rozbije vstup na tokeny. Lze ji implementovat opakovaným voláním funkce, která pokaždé vrátí údaj o dalším tokenu, na konci vstupu vrátí nárazníkový literál.
- Syntaktická analýza: Ze vstupních tokenů postaví syntaktický strom (tedy sdělí, jak program syntakticky vznikl).
- Sémantická analýza: K syntaktickému stromu dodá informace o tom, co operace znamenají, případně strom přestaví.
- Generátor interkódu: Ze sémantického stromu postaví předobraz binárního kódu.
- Optimalizátor: Optimalizuje interkód.

Vnitřnosti překladače

- Lexikální analýza: Rozbije vstup na tokeny. Lze ji implementovat opakovaným voláním funkce, která pokaždé vrátí údaj o dalším tokenu, na konci vstupu vrátí nárazníkový literál.
- Syntaktická analýza: Ze vstupních tokenů postaví syntaktický strom (tedy sdělí, jak program syntakticky vznikl).
- Sémantická analýza: K syntaktickému stromu dodá informace o tom, co operace znamenají, případně strom přestaví.
- Generátor interkódu: Ze sémantického stromu postaví předobraz binárního kódu.
- Optimalizátor: Optimalizuje interkód.
- Generátor kódu.

Vnitřnosti překladače

- Lexikální analýza: Rozbije vstup na tokeny. Lze ji implementovat opakovaným voláním funkce, která pokaždé vrátí údaj o dalším tokenu, na konci vstupu vrátí nárazníkový literál.
- Syntaktická analýza: Ze vstupních tokenů postaví syntaktický strom (tedy sdělí, jak program syntakticky vznikl).
- Sémantická analýza: K syntaktickému stromu dodá informace o tom, co operace znamenají, případně strom přestaví.
- Generátor interkódu: Ze sémantického stromu postaví předobraz binárního kódu.
- Optimalizátor: Optimalizuje interkód.
- Generátor kódu.
- První tři jsou zvané front-end, druhé tři back-end. My se omezíme na front-end.

Všechno ostatní

mimo syntaktickou analýzu

- Všechny kroky se provádějí nějakým automatem (tedy hledáme typické formace a něco s nimi děláme).

Všechno ostatní

mimo syntaktickou analýzu

- Všechny kroky se provádějí nějakým automatem (tedy hledáme typické formace a něco s nimi děláme).
- Lexikální analýza je nyní jasná.

Všechno ostatní

mimo syntaktickou analýzu

- Všechny kroky se provádějí nějakým automatem (tedy hledáme typické formace a něco s nimi děláme).
- Lexikální analýza je nyní jasná.
- Sémantická závisí na jazyku (při vhodném návrhu syntaktické analýzy může být prázdná).

Všechno ostatní

mimo syntaktickou analýzu

- Všechny kroky se provádějí nějakým automatem (tedy hledáme typické formace a něco s nimi děláme).
- Lexikální analýza je nyní jasná.
- Sémantická závisí na jazyku (při vhodném návrhu syntaktické analýzy může být prázdná).
- Back-end závisí na konkrétní architektuře a jeho návrh by vyžadoval znalost Assembleru. Mimochodem mezikódem může být i sémantický strom.

Všechno ostatní

mimo syntaktickou analýzu

- Všechny kroky se provádějí nějakým automatem (tedy hledáme typické formace a něco s nimi děláme).
- Lexikální analýza je nyní jasná.
- Sémantická závisí na jazyku (při vhodném návrhu syntaktické analýzy může být prázdná).
- Back-end závisí na konkrétní architektuře a jeho návrh by vyžadoval znalost Assembleru. Mimochodem mezikódem může být i sémantický strom.
- Pokud napíšeme front-end a místo generování kódu interpretujeme strom, mluvíme o interpretu.

Syntaktická analýza

. . . aneb jak postavit syntaktický strom?

- Syntaktická analýza zpravidla probíhá jednoprůchodově tzv. zleva doprava.

Syntaktická analýza

. . . aneb jak postavit syntaktický strom?

- Syntaktická analýza zpravidla probíhá jednoprůchodově tzv. zleva doprava.
- Strom můžeme stavět od kořene, nebo od listů (jinak se v tom zapleteme).

Syntaktická analýza

. . . aneb jak postavit syntaktický strom?

- Syntaktická analýza zpravidla probíhá jednoprůchodově tzv. zleva doprava.
- Strom můžeme stavět od kořene, nebo od listů (jinak se v tom zapleteme).
- Máme k dispozici gramatiku sestávající z přepisovacích pravidel.

Syntaktická analýza

. . . aneb jak postavit syntaktický strom?

- Syntaktická analýza zpravidla probíhá jednoprůchodově tzv. zleva doprava.
- Strom můžeme stavět od kořene, nebo od listů (jinak se v tom zapleteme).
- Máme k dispozici gramatiku sestávající z přepisovacích pravidel.
- Stavba od kořene (tzv. LL-analýza): Je třeba vědět, které pravidlo v daném okamžiku použijeme.

Syntaktická analýza

. . . aneb jak postavit syntaktický strom?

- Syntaktická analýza zpravidla probíhá jednoprůchodově tzv. zleva doprava.
- Strom můžeme stavět od kořene, nebo od listů (jinak se v tom zapleteme).
- Máme k dispozici gramatiku sestávající z přepisovacích pravidel.
- Stavba od kořene (tzv. LL-analýza): Je třeba vědět, které pravidlo v daném okamžiku použijeme.
- Kde jsme se (už loni) s LL-analýzou setkali?

Syntaktická analýza

. . . aneb jak postavit syntaktický strom?

- Syntaktická analýza zpravidla probíhá jednoprůchodově tzv. zleva doprava.
- Strom můžeme stavět od kořene, nebo od listů (jinak se v tom zapleteme).
- Máme k dispozici gramatiku sestávající z přepisovacích pravidel.
- Stavba od kořene (tzv. LL-analýza): Je třeba vědět, které pravidlo v daném okamžiku použijeme.
- Kde jsme se (už loni) s LL-analýzou setkali?
- Stavba od listů (tzv. LR-analýza): Tak dlouho skládáme vstup na zásobník, až nám po sobě jdoucí tokeny začnou dávat smysl.

Syntaktická analýza

. . . aneb jak postavit syntaktický strom?

- Syntaktická analýza zpravidla probíhá jednoprůchodově tzv. zleva doprava.
- Strom můžeme stavět od kořene, nebo od listů (jinak se v tom zapleteme).
- Máme k dispozici gramatiku sestávající z přepisovacích pravidel.
- Stavba od kořene (tzv. LL-analýza): Je třeba vědět, které pravidlo v daném okamžiku použijeme.
- Kde jsme se (už loni) s LL-analýzou setkali?
- Stavba od listů (tzv. LR-analýza): Tak dlouho skládáme vstup na zásobník, až nám po sobě jdoucí tokeny začnou dávat smysl.
- Kde jsme loni viděli tohle?

Množiny FIRST a FOLLOW

FIRST

- Abychom mohli stavět strom shora dolů, hodí se ke každému symbolu gramatiky množina FIRST(A), která řekne, na které první terminály lze přepsat daný symbol.

Množiny FIRST a FOLLOW

FIRST

- Abychom mohli stavět strom shora dolů, hodí se ke každému symbolu gramatiky množina FIRST(A), která řekne, na které první terminály lze přepsat daný symbol.
- Pro terminály je to jasné, FIRST(α)=$\{\alpha\}$.

Množiny FIRST a FOLLOW

FIRST

- Abychom mohli stavět strom shora dolů, hodí se ke každému symbolu gramatiky množina FIRST(A), která řekne, na které první terminály lze přepsat daný symbol.
- Pro terminály je to jasné, FIRST(α)=$\{\alpha\}$.
- Pro neterminály tyto množiny postupně rozšiřujeme:

Množiny FIRST a FOLLOW

FIRST

- Abychom mohli stavět strom shora dolů, hodí se ke každému symbolu gramatiky množina FIRST(A), která řekne, na které první terminály lze přepsat daný symbol.
- Pro terminály je to jasné, FIRST(α)=$\{\alpha\}$.
- Pro neterminály tyto množiny postupně rozšiřujeme:
- Pro každé pravidlo $A \rightarrow \alpha...$ do FIRST(A) přidej α.

Množiny FIRST a FOLLOW

FIRST

- Abychom mohli stavět strom shora dolů, hodí se ke každému symbolu gramatiky množina FIRST(A), která řekne, na které první terminály lze přepsat daný symbol.
- Pro terminály je to jasné, FIRST(α)=$\{\alpha\}$.
- Pro neterminály tyto množiny postupně rozšiřujeme:
- Pro každé pravidlo $A \rightarrow \alpha\ldots$ do FIRST(A) přidej α.
- Pro každé pravidlo $A \rightarrow B\ldots$ do FIRST(A) přidej FIRST(B).

Množiny FIRST a FOLLOW

FIRST

- Abychom mohli stavět strom shora dolů, hodí se ke každému symbolu gramatiky množina FIRST(A), která řekne, na které první terminály lze přepsat daný symbol.
- Pro terminály je to jasné, FIRST(α)=$\{\alpha\}$.
- Pro neterminály tyto množiny postupně rozšiřujeme:
- Pro každé pravidlo $A \rightarrow \alpha...$ do FIRST(A) přidej α.
- Pro každé pravidlo $A \rightarrow B...$ do FIRST(A) přidej FIRST(B).
- Pro pravidlo $A \rightarrow \lambda...$ do FIRST(A) přidej λ.

Množiny FIRST a FOLLOW

second part

- Množina FOLLOW říká, co může jít za dotyčným neterminálem, tedy:

Množiny FIRST a FOLLOW

second part

- Množina FOLLOW říká, co může jít za dotyčným neterminálem, tedy:
- Pro každé pravidlo $A \rightarrow B\alpha...$ přidej α do FOLLOW(B).

Množiny FIRST a FOLLOW

second part

- Množina FOLLOW říká, co může jít za dotyčným neterminálem, tedy:
- Pro každé pravidlo $A \rightarrow B\alpha...$ přidej α do FOLLOW(B).
- Pro každé pravidlo $A \rightarrow BC...$ přidej do FOLLOW(B) množinu FIRST(C).

Množiny FIRST a FOLLOW

second part

- Množina FOLLOW říká, co může jít za dotyčným neterminálem, tedy:
- Pro každé pravidlo $A \rightarrow B\alpha...$ přidej α do FOLLOW(B).
- Pro každé pravidlo $A \rightarrow BC...$ přidej do FOLLOW(B) množinu FIRST(C).
- Pro každé pravidlo $A \rightarrow ...C$ přidej do FOLLOW(C) množinu FOLLOW(A).

Množiny FIRST a FOLLOW

second part

- Množina FOLLOW říká, co může jít za dotyčným neterminálem, tedy:
- Pro každé pravidlo $A \rightarrow B\alpha...$ přidej α do FOLLOW(B).
- Pro každé pravidlo $A \rightarrow BC...$ přidej do FOLLOW(B) množinu FIRST(C).
- Pro každé pravidlo $A \rightarrow ...C$ přidej do FOLLOW(C) množinu FOLLOW(A).
- Pokud $\lambda \in$ FIRST(A) a B se po aplikaci posloupnosti nějakých pravidel může objevit bezprostředně za A, pak do FIRST(A) přidej i FIRST(B).

Množiny FIRST a FOLLOW

second part

- Množina FOLLOW říká, co může jít za dotyčným neterminálem, tedy:
- Pro každé pravidlo $A \rightarrow B\alpha...$ přidej α do FOLLOW(B).
- Pro každé pravidlo $A \rightarrow BC...$ přidej do FOLLOW(B) množinu FIRST(C).
- Pro každé pravidlo $A \rightarrow ...C$ přidej do FOLLOW(C) množinu FOLLOW(A).
- Pokud $\lambda \in$ FIRST(A) a B se po aplikaci posloupnosti nějakých pravidel může objevit bezprostředně za A, pak do FIRST(A) přidej i FIRST(B).
- Pokud se B může objevit na konci, přidej \$ do FOLLOW(B).

Množiny FIRST a FOLLOW

second part

- Množina FOLLOW říká, co může jít za dotyčným neterminálem, tedy:
- Pro každé pravidlo $A \rightarrow B\alpha...$ přidej α do FOLLOW(B).
- Pro každé pravidlo $A \rightarrow BC...$ přidej do FOLLOW(B) množinu FIRST(C).
- Pro každé pravidlo $A \rightarrow ...C$ přidej do FOLLOW(C) množinu FOLLOW(A).
- Pokud $\lambda \in$ FIRST(A) a B se po aplikaci posloupnosti nějakých pravidel může objevit bezprostředně za A, pak do FIRST(A) přidej i FIRST(B).
- Pokud se B může objevit na konci, přidej \$ do FOLLOW(B).
- ...

Pověstná gramatika kolegy Yaghoba

Příklad

- $E \rightarrow E + T$
- $E \rightarrow T$
- $T \rightarrow T * F$
- $T \rightarrow F$
- $F \rightarrow (E)$
- $F \rightarrow \mathrm{id}$
- Eliminace levé rekurze:
- $E \rightarrow TE'$, $E' \rightarrow +TE'$, $E \rightarrow \lambda$,
- $T \rightarrow FT'$, $T' \rightarrow *FT'$, $T' \rightarrow \lambda$,
- $F \rightarrow (E)$, $F \rightarrow id$.

FIRST a FOLLOW

v pověstné gramatice kolegy Yaghoba

- FIRST$(E, T, F)=\{(, id\}$, FOLLOW$(E, E')=\{), \$\}$
- FIRST$(E')=\{+, \lambda\}$, FOLLOW$(T, T')=\{+,), \$\}$
- FIRST$(T')=\{*, \lambda\}$, FOLLOW$(F)=\{+, *,), \$\}$

LL(1) automat

- Má výhled na 1 symbol kupředu,

LL(1) automat

- Má výhled na 1 symbol kupředu,
- staví strom shora dolů, tedy které pravidlo použijeme, se rozhoduje podle množiny FIRST.

LL(1) automat

- Má výhled na 1 symbol kupředu,
- staví strom shora dolů, tedy které pravidlo použijeme, se rozhoduje podle množiny FIRST.
- Pravidlo s λ se použije, pokud není symbol ve výhledu ve FIRST pro žádné pravidlo.

LL(1) automat

- Má výhled na 1 symbol kupředu,
- staví strom shora dolů, tedy které pravidlo použijeme, se rozhoduje podle množiny FIRST.
- Pravidlo s λ se použije, pokud není symbol ve výhledu ve FIRST pro žádné pravidlo.
- Implementace: Neterminály odpovídají funkcím,

LL(1) automat

- Má výhled na 1 symbol kupředu,
- staví strom shora dolů, tedy které pravidlo použijeme, se rozhoduje podle množiny FIRST.
- Pravidlo s λ se použije, pokud není symbol ve výhledu ve FIRST pro žádné pravidlo.
- Implementace: Neterminály odpovídají funkcím,
- každá funkce se podívá, které pravidlo (pro přepsání na dotyčný terminál) použít. Pokud nic nevyhovuje, jde o syntaktickou chybu.

LL(1) automat

- Má výhled na 1 symbol kupředu,
- staví strom shora dolů, tedy které pravidlo použijeme, se rozhoduje podle množiny FIRST.
- Pravidlo s λ se použije, pokud není symbol ve výhledu ve FIRST pro žádné pravidlo.
- Implementace: Neterminály odpovídají funkcím,
- každá funkce se podívá, které pravidlo (pro přepsání na dotyčný terminál) použít. Pokud nic nevyhovuje, jde o syntaktickou chybu.
- Jedná se tedy o obyčejnou rekurzi (jako kdysi v úloze "Evaluace prefixní notace").

LL(1) automat

- Má výhled na 1 symbol kupředu,
- staví strom shora dolů, tedy které pravidlo použijeme, se rozhoduje podle množiny FIRST.
- Pravidlo s λ se použije, pokud není symbol ve výhledu ve FIRST pro žádné pravidlo.
- Implementace: Neterminály odpovídají funkcím,
- každá funkce se podívá, které pravidlo (pro přepsání na dotyčný terminál) použít. Pokud nic nevyhovuje, jde o syntaktickou chybu.
- Jedná se tedy o obyčejnou rekurzi (jako kdysi v úloze "Evaluace prefixní notace").
- Pokud vznikne nejednoznačnost, LL-analýzu nelze použít.

LL(1) – pokr.

- LL-jazykem je třeba Pascal, ale ne například C (natož C#).

LL(1) – pokr.

- LL-jazykem je třeba Pascal, ale ne například C (natož C#).
- Rekurzi lze eliminovat (nahradit tabulkou, která podle FIRST říká, které pravidlo se použije teď).

LL(1) – pokr.

- LL-jazykem je třeba Pascal, ale ne například C (natož C#).
- Rekurzi lze eliminovat (nahradit tabulkou, která podle FIRST říká, které pravidlo se použije teď).
- Pokud chceme obecný LL(k) automat, je třeba definovat množiny FIRST a FOLLOW ne jako množiny terminálů a speciálních symbolů, ale jako množiny k-tic (těchto symbolů), jelikož s nimi budeme porovnávat výhled (lookahead), ve kterém je k po sobě jdoucích symbolů.

LR-analýza

- Automat nějakou dobu symboly pouze strká na zásobník.

LR-analýza

- Automat nějakou dobu symboly pouze strká na zásobník.
- V jednu chvíli načítání zastaví a začne symboly na vrcholu zásobníku přepisovat.

LR-analýza

- Automat nějakou dobu symboly pouze strká na zásobník.
- V jednu chvíli načítání zastaví a začne symboly na vrcholu zásobníku přepisovat.
- Tyto dva kroky automat všelijak střídá.

LR-analýza

- Automat nějakou dobu symboly pouze strká na zásobník.
- V jednu chvíli načítání zastaví a začne symboly na vrcholu zásobníku přepisovat.
- Tyto dva kroky automat všelijak střídá.
- Na zásobníku nám tak vznikají podstromečky syntaktického stromu, které všelijak slepujeme dohromady.

LR-analýza

- Automat nějakou dobu symboly pouze strká na zásobník.
- V jednu chvíli načítání zastaví a začne symboly na vrcholu zásobníku přepisovat.
- Tyto dva kroky automat všelijak střídá.
- Na zásobníku nám tak vznikají podstromečky syntaktického stromu, které všelijak slepujeme dohromady.
- Funkce `shift (S)` znak v lookaheadu uloží na zásobník a za něj (taktéž na zásobník) automat uloží údaj o stavu `S`. Do lookaheadu je načten další terminál.

LR-analýza

- Automat nějakou dobu symboly pouze strká na zásobník.
- V jednu chvíli načítání zastaví a začne symboly na vrcholu zásobníku přepisovat.
- Tyto dva kroky automat všelijak střídá.
- Na zásobníku nám tak vznikají podstromečky syntaktického stromu, které všelijak slepujeme dohromady.
- Funkce `shift (S)` znak v lookaheadu uloží na zásobník a za něj (taktéž na zásobník) automat uloží údaj o stavu `S`. Do lookaheadu je načten další terminál.
- Funkce `reduction` podle pravidla $A \rightarrow B$ načte ze zásobníku $|B|$ symbolů, na zásobník přidá A a přejde do stavu určeného funkcí `GOTO(zás,A)`, kde `zás` je stav, který se nachází na zásobníku bezprostředně pod slovem B.

LR-analýza pokr.

- Rozšířením gramatiky nazveme gramatiku, která má v každém pravidle na pravé straně právě jedenkrát speciální symbol zvaný tečka.

LR-analýza pokr.

- Rozšířením gramatiky nazveme gramatiku, která má v každém pravidle na pravé straně právě jedenkrát speciální symbol zvaný tečka.
- Tečka bude určovat, kde v dotyčném pravidle v současném okamžiku jsme, tedy stavy budou odpovídat jednotlivým tečkovaným pravidlům (přesněji jejich podmnožinám) a funkce GOTO definuje přechod přes neterminál. Shift nás podobným způsobem přenese v daných pravidlech přes terminál.

LR-analýza pokr.

- Rozšířením gramatiky nazveme gramatiku, která má v každém pravidle na pravé straně právě jedenkrát speciální symbol zvaný tečka.
- Tečka bude určovat, kde v dotyčném pravidle v současném okamžiku jsme, tedy stavy budou odpovídat jednotlivým tečkovaným pravidlům (přesněji jejich podmnožinám) a funkce GOTO definuje přechod přes neterminál. Shift nás podobným způsobem přenese v daných pravidlech přes terminál.
- Automat bude též vybaven funkcemi `accept` a `error`. První říká, že strom je úspěšně postavený, druhá, že jej postavit nelze (pokoušíme se použít nedefinovaný přechod).

SLR(1) automat

- K definici SLR-automatu vytvoříme z množiny I uzávěr CLOSURE(I) tak, že pro každé pravidlo z CLOSURE(I) tvaru $A \rightarrow \alpha.B\beta$, kde B je neterminál přidáme do CLOSURE(I) všechna pravidla tvaru $B \rightarrow .X$.
- V první fázi vytvoříme uzávěr pro startovací pravidlo.
- Stavy budou indexovány množinami otečkovaných pravidel (které říkají, na kterém místě kterého pravidla můžeme být).
- Pro pravidlo $I : A \rightarrow \alpha.X\beta$ a symbol X definujeme přechod GOTO(I, X) jako uzávěr pravidla $I : A \rightarrow \alpha X.\beta$.
- Pro přechod přes terminál definujeme operaci shift, pro přechod přes neterminál definujeme GOTO (do dotyčného stavu), dojede-li tečka na konec pravidla, definujeme redukci (podle dotyčného pravidla).

Poznámky

- Podrobnosti viz kam.mff.cuni.cz/ perm/programovani/NPRM049/gramatiky.txt

Poznámky

- Podrobnosti viz kam.mff.cuni.cz/ perm/programovani/NPRM049/gramatiky.txt
- a www.ksi.mff.cuni.cz/public/NSWI098pub/html/pp-03-lasxa.ppt

Poznámky

- Podrobnosti viz kam.mff.cuni.cz/ perm/programovani/NPRM049/gramatiky.txt
- a www.ksi.mff.cuni.cz/public/NSWI098pub/html/pp-03-lasxa.ppt
- LR-automat tedy sestává ze zásobníku a tabulek pro shifty, redukce a GOTO.

Poznámky

- Podrobnosti viz kam.mff.cuni.cz/ perm/programovani/NPRM049/gramatiky.txt
- a www.ksi.mff.cuni.cz/public/NSWI098pub/html/pp-03-lasxa.ppt
- LR-automat tedy sestává ze zásobníku a tabulek pro shifty, redukce a GOTO.
- Přesný návrh se liší podle druhu automatu (LR(0), SLR(1), LALR(1), LR(1)).

Poznámky

- Podrobnosti viz kam.mff.cuni.cz/ perm/programovani/NPRM049/gramatiky.txt
- a www.ksi.mff.cuni.cz/public/NSWI098pub/html/pp-03-lasxa.ppt
- LR-automat tedy sestává ze zásobníku a tabulek pro shifty, redukce a GOTO.
- Přesný návrh se liší podle druhu automatu (LR(0), SLR(1), LALR(1), LR(1)).
- Pokus nastanou kolize, není gramatika toho správného typu a měli bychom ji přepsat.

Poznámky

- Podrobnosti viz kam.mff.cuni.cz/ perm/programovani/NPRM049/gramatiky.txt
- a www.ksi.mff.cuni.cz/public/NSWI098pub/html/pp-03-lasxa.ppt
- LR-automat tedy sestává ze zásobníku a tabulek pro shifty, redukce a GOTO.
- Přesný návrh se liší podle druhu automatu (LR(0), SLR(1), LALR(1), LR(1)).
- Pokus nastanou kolize, není gramatika toho správného typu a měli bychom ji přepsat.
- Místo přepsání zneužijeme některou z vlastností, tedy pokud je shift/reduction konflikt, rozhodujeme se pro shift.

Problémy

- Člověk zpravidla opíše z normy gramatiku, která není ani rozhodnutelná, natož SLR.

Problémy

- Člověk zpravidla opíše z normy gramatiku, která není ani rozhodnutelná, natož SLR.
- Nejednoznačná gramatika a konflikty.

Problémy

- Člověk zpravidla opíše z normy gramatiku, která není ani rozhodnutelná, natož SLR.
- Nejednoznačná gramatika a konflikty.
- Na některé konflikty existuje průmyslové řešení,

Problémy

- Člověk zpravidla opíše z normy gramatiku, která není ani rozhodnutelná, natož SLR.
- Nejednoznačná gramatika a konflikty.
- Na některé konflikty existuje průmyslové řešení,
- výhodou je rozmyslet si, co naše rozhodnutí udělá.

Problémy II

- Eliminace levé rekurze (při LL-analýze): Buďto navrhneme operátory asociativní zprava a strom přehazujeme při sémantické analýze, nebo uděláme sprostý trik v podobě analýzy od konce k začátku (pokud jsou všechny operátory asociativní zleva), anebo gramatiku kompletně rozkopeme.

Problémy II

- Eliminace levé rekurze (při LL-analýze): Buďto navrhneme operátory asociativní zprava a strom přehazujeme při sémantické analýze, nebo uděláme sprostý trik v podobě analýzy od konce k začátku (pokud jsou všechny operátory asociativní zleva), anebo gramatiku kompletně rozkopeme.
- Levá faktorizace: Když není jasné, které pravidlo pro redukci vybrat, gramatiku zkusíme přepsat tak, že pravidla změníme, aby stačilo rozhodnout se později.
  Příklad: $A \to \alpha\beta$, $A \to \alpha\gamma$ změníme na:
  $A \to \alpha B$, $B \to \beta$, $B \to \gamma$.