Počítačové zpracování přirozeného jazyka

Dvojúrovňová morfologie

Daniel Zeman

http://ufal.mff.cuni.cz/course/popj1/

Dvojúrovňová (mor)fonologie

- Kimmo Koskenniemi: disertační práce (1983).
- Lze zkoušet programem **`pc-kimmo`** (je zdarma na http://www.sil.org/pckimmo/).
- Lauri Karttunen (Xerox Grenoble): překladač dvojúrovňových pravidel do konečných převodníků (two-level compiler, finite state technology, dokumentace na http://www.xrce.xerox.com/).
- Morfologická „klasika“.

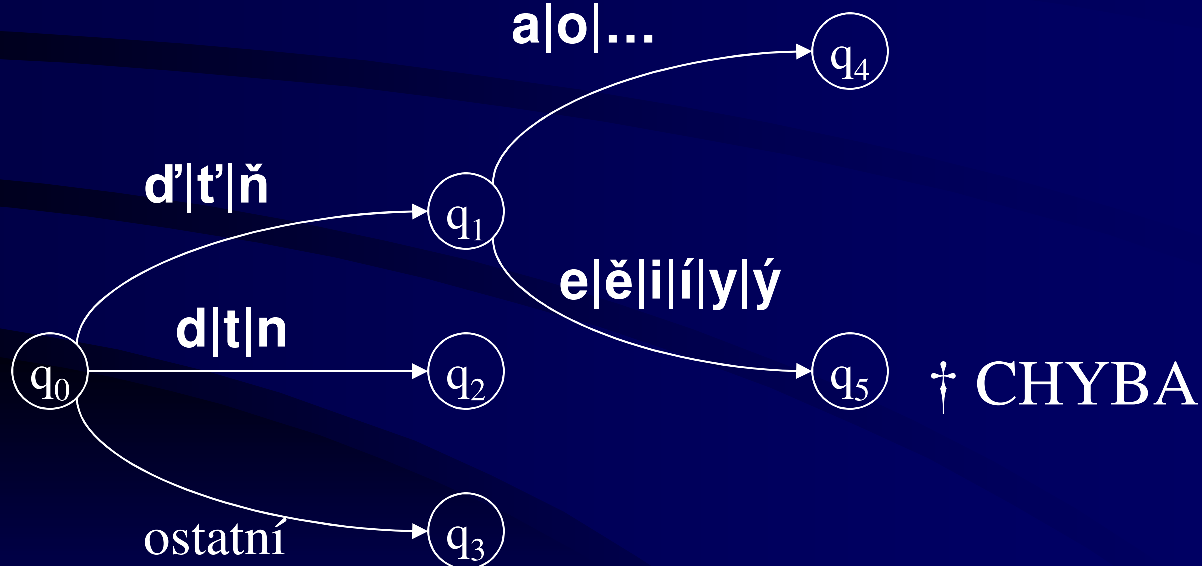
Konečný automat

- Pětice (A, Q, P, q_0, F).
  - A … konečná abeceda vstupních symbolů.
  - Q … konečná množina stavů automatu.
  - P … přechodová funkce (množina pravidel) $A \times Q \to Q$.
  - $q_0 \in Q$ … počáteční stav automatu.
  - $F \subseteq Q$ … množina koncových stavů automatu.

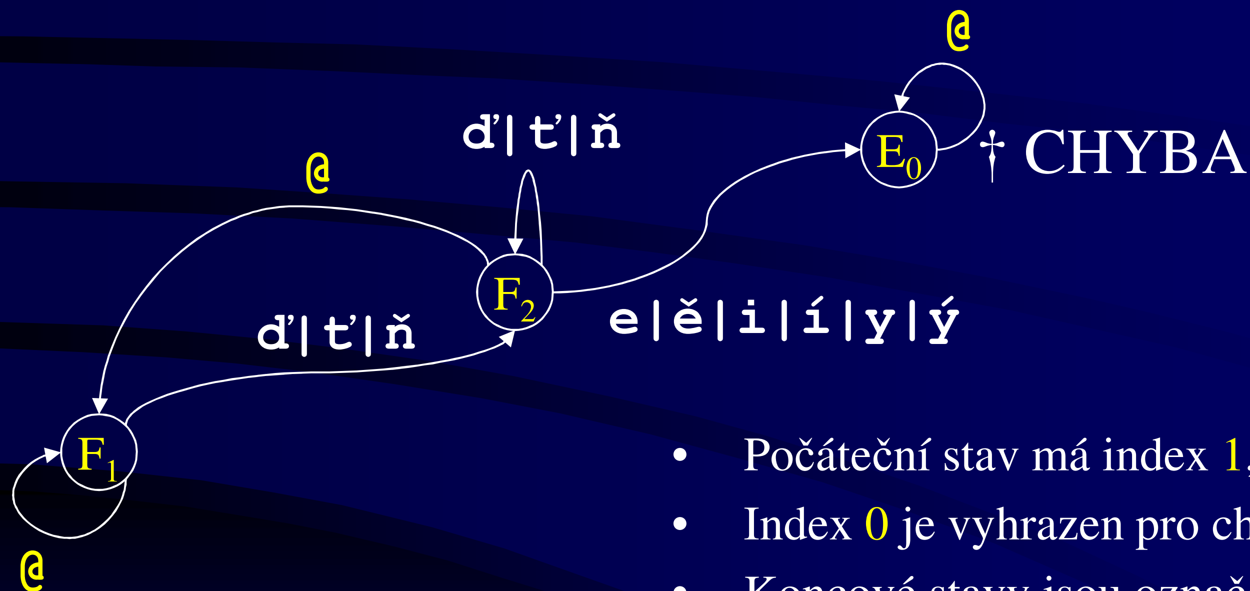
- Slovo je přijato jako správné, pokud ho přečteme jako vstup a skončíme v koncovém stavu.
- S koncovým stavem lze svázat další akci (výstupní info).

Příklad konečného automatu

- Kontroluje, zda je správně napsáno dě, tě, ně…
- Neumí výjimky („ťin“)



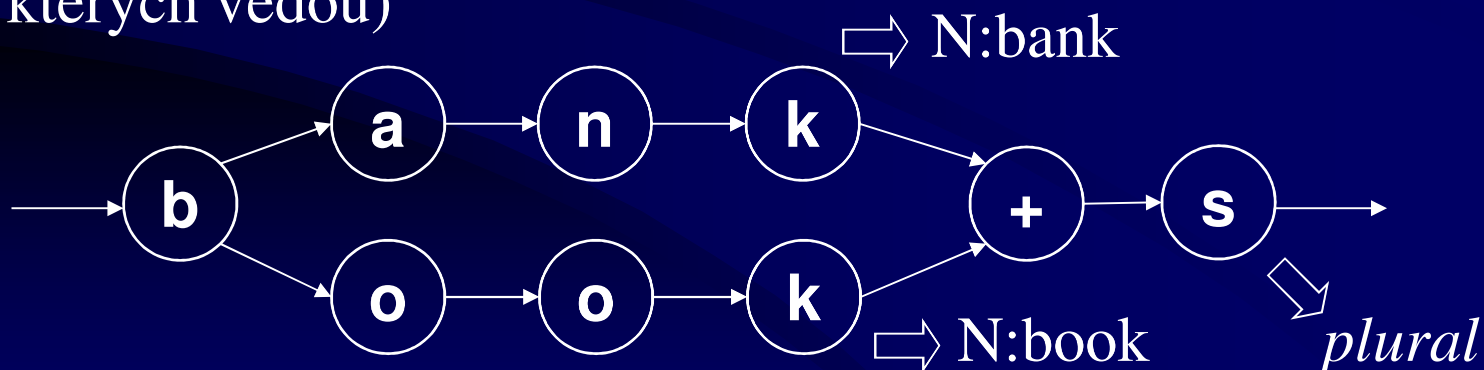
Příklad konečného automatu (dotažený, nová notace)



- Počáteční stav má index 1, ne 0 (zde tedy F_1).
- Index 0 je vyhrazen pro chybový stav.
- Koncové stavy jsou označené písmenem F.
- Zavináč („@“) znamená „ostatní“, tedy znaky, které nejsou na jiných šipkách se stejným počátkem.

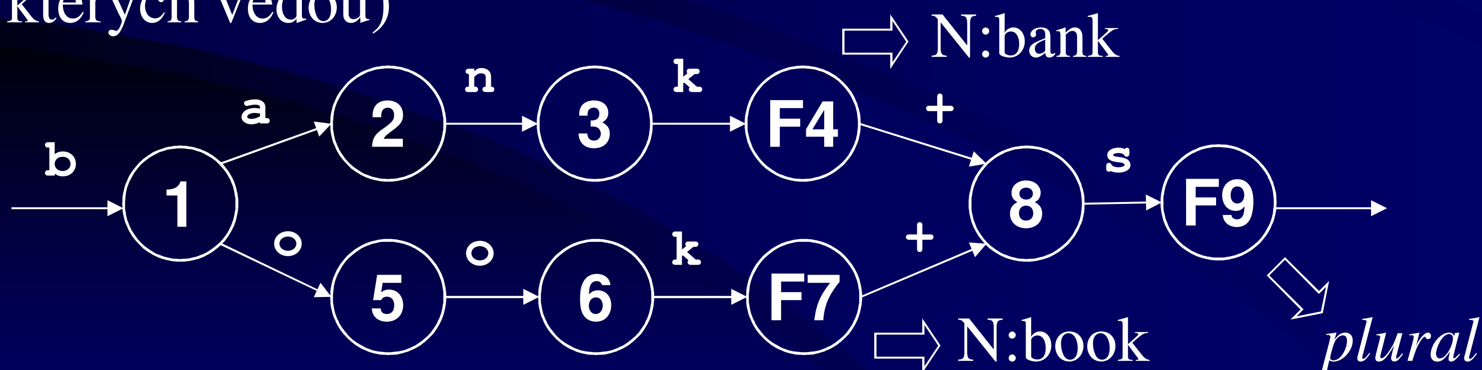
Slovník

- Implementován jako konečný automat (trie) *[trí]*.
- Zkompilován ze seznamu řetězců a slovníkových odkazů.
- Podslovníky pro kmeny, předpony, přípony, koncovky.
- Poznámky (glosy) na konci každého podslovníku.
- Příklad: (hrany jsou ohodnoceny stejně jako uzly, do kterých vedou)



Slovník

- Implementován jako konečný automat (trie) *[trí]*.
- Zkompilován ze seznamu řetězců a slovníkových odkazů.
- Podslovníky pro kmeny, předpony, přípony, koncovky.
- Poznámky (glosy) na konci každého podslovníku.
- Příklad: (hrany jsou ohodnoceny stejně jako uzly, do kterých vedou)



Příklady slovníků

- Anglické kmeny podstatných jmen (typicky současně celá slova): *book, bank, car, cat, donut…*
- Viz též **`pc-kimmo / englex`**.
- České kmeny (ne vždy celé lemma): *pán, hrad, muž, stroj, (před)sed, soudc, žen, růž, píseň, kost, měst, moř, kuř, staven*
- České předpony: *do, na, od, po, pře, před, při, se, z, za…*
  odpo, dopři, pona… *nej, ne* *dvoj, troj…*

Příklady slovníků

- České koncovky podstatných jmen:
  - *0, a, e, u, ovi, i, o, em, ou, i, ové, y, ů, ům, ech, ích*
  - *a, e, 0, y, i, u, o, ou, í, ám, ím, em, ách, ích, ech, ami, emi, mi*
  - *o, e, í, a, ete, u, i, eti, em, etem, ím, ata, 0, at, ům, ím, atům, ech, ích…*
- České koncovky přídavných jmen:
  - *ý, ého, ému, ým, í, ých, é, ými, á, ou, ém*
  - *í, ího, ímu, ím, ích, ími*
  - *(ej+, ěj+) š + í, ího, …*
- České koncovky sloves:
  - *(n+) u, eš, e, eme, ete, ou*
ím, íš, í, íme, íte, í
ám, áš, á, áme, áte, ají
  - *(e+, u+) (j+) 0, me, te*
  - *l, en, t*
    - *0, a, o, i, y, y, a*

Třídy pokračování

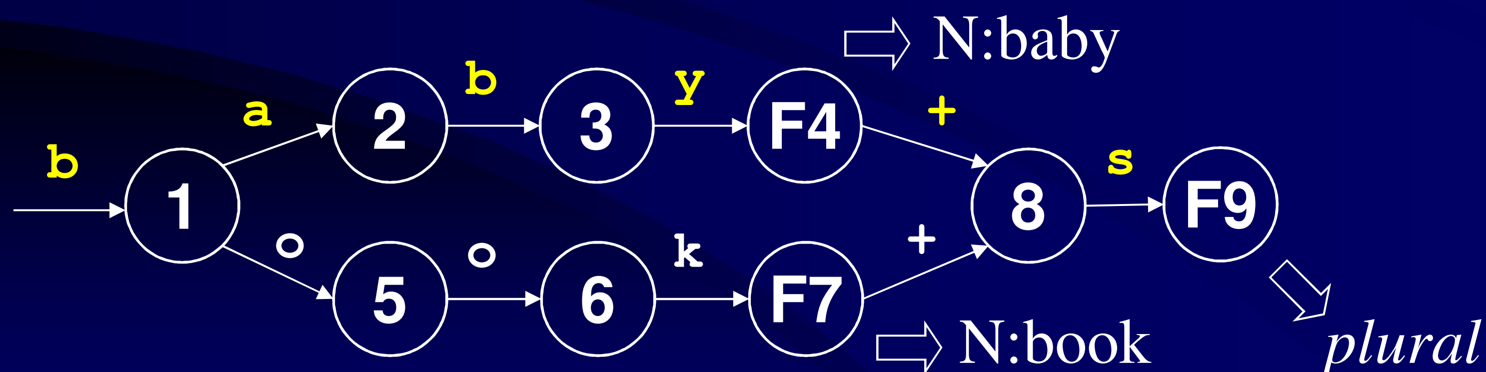
- Na rozdíl od trie slovník není strom, ale DAG (orientovaný acyklický graf).
- Slovník zná pro každé heslo **třídu pokračování** (continuation class, alternation).
- Třída pokračování je množina podslovníků, do nichž lze přejít na konci tohoto podslovníku (po přijetí hesla).
- Například z podslovníku jmenných kmenů lze přejít do podslovníků pro koncovky pádů.
- Tříd pokračování jmenných kmenů je tolik, kolik je jmenných vzorů (viz příklad v **`pc-kimmo`**).

Ukázka v PC Kimmo

- http://ufal.mff.cuni.cz/~zeman/vyuka/podklady/pckimmo-cs.zip
- Slovník českých podstatných jmen
- Slovník koncovek podle vzoru *žena*
- `t cs`
- `set grammar off`
- `r žena`
- `r ženy`
- Pro každou interpretaci +*y* samostatné heslo, abychom mohli mít různé glosy

Problém jménem fonologie

- Připojení koncovky někdy způsobí hláskové změny!
- Množné číslo od *baby* (mimino) není **babys*, ale *babies*!
  - *Ve skutečnosti zrovna tahle změna není dána pravidly fonologie, ale pravopisu. Pro nás to ale bude totéž.*



Dva v jednom: morfologie a fonologie

- Integrace morfologie a fonologie je možná a snadná.
- Dvojúrovňová je vlastně spíše fonologie než morfologie.
- Morfologie (morfematika): Propojené slovníky realizované konečnými automaty (FSA) (právě jsme viděli).
- Fonologie: dvojúrovňová. Sada pravidel realizovaných konečnými převodníky (FST). Příklad pravidla:

```
b a b y + 0 s
b a b i 0 e s
```

Dvojúrovňová pravidla

- **Horní a dolní jazyk**
  - Hornímu se zde též říká **lexikální**.
  - Dolnímu se zde též říká **povrchový**.
- Dvouřádkový zápis se zkracuje pomocí dvojteček:

```
b a b y + 0 s
b a b i 0 e s

b:b a:a b:b y:i +:0 0:e s:s
```

- Znak + obvykle označuje rozhraní dvou morfémů.
- Znak 0 obvykle označuje prázdné místo (jeho protějšek nemá na této rovině žádnou realizaci).
- Další zvláštní znaky PC-Kimma: #, @.

Konečný převodník

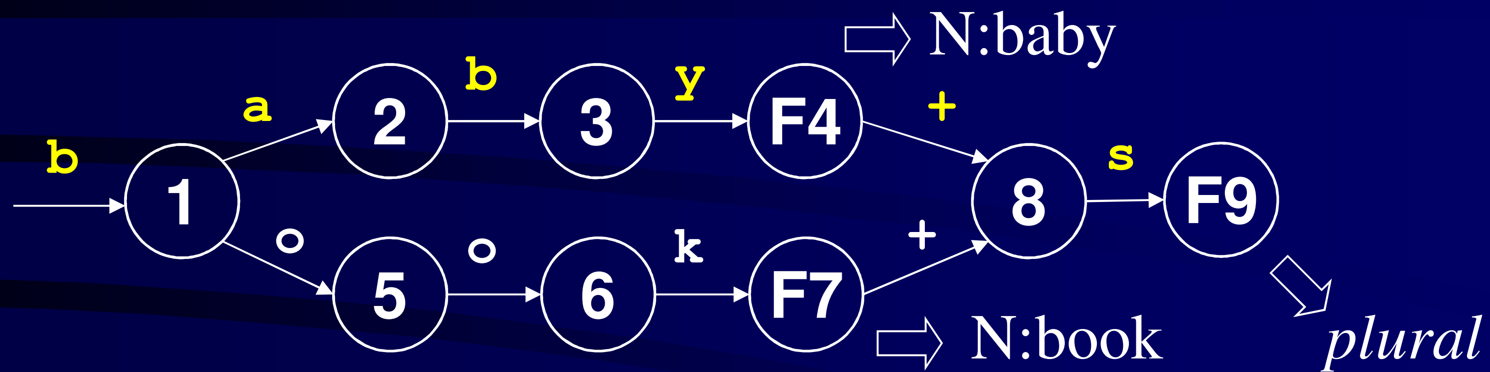
Horní jazyk

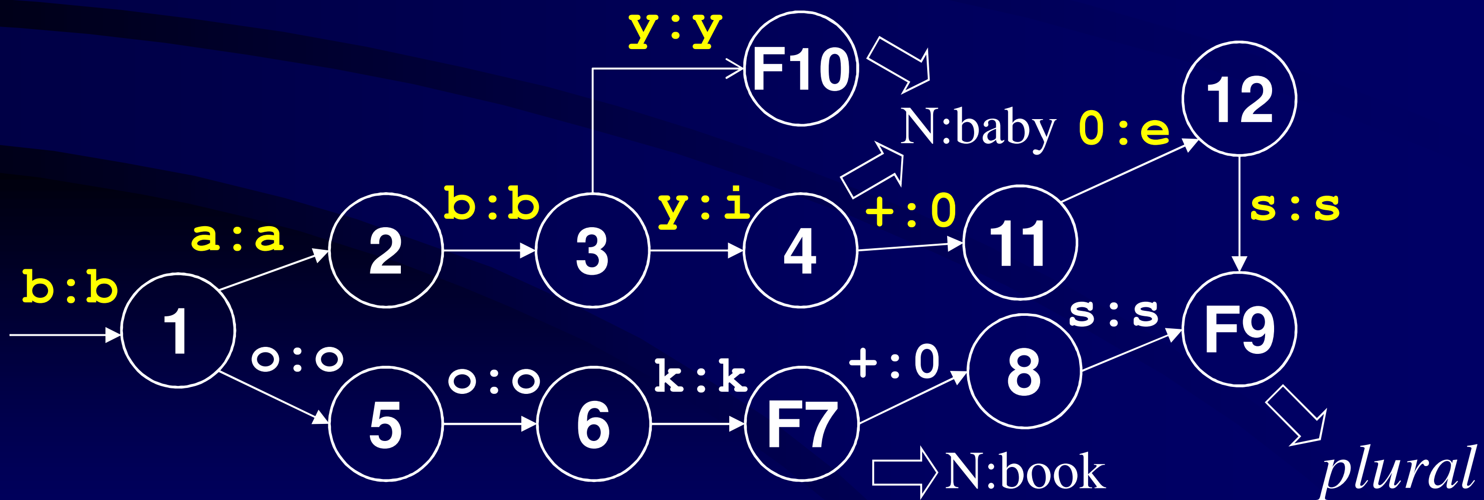
- Převodník je speciální případ automatu.
  - Symboly jsou dvojice (r:s) z konečných abeced R a S.

Dolní jazyk

- Kontrola (~ konečný automat)
  - vstup: posloupnost dvojic
  - výstup: ano / ne (přijmout / nepřijmout)
- Analýza:
  - vstup: posloupnost $s \in S$ (2ú morfologie: povrchový zápis)
  - výstup: posloupnost $r \in R$ (2ú morfologie: lexikální zápis) + přídavné informace ze slovníku
- Syntéza:
  - jako analýza, ale prohozené role $S \leftrightarrow R$

Automat vs. převodník

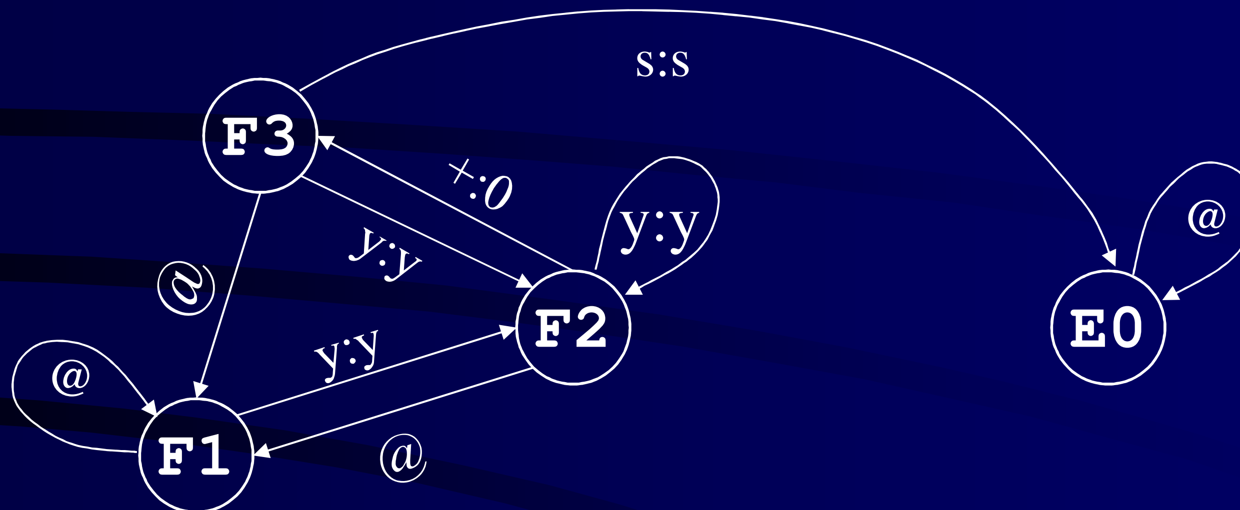




Převodník pro jedno pravidlo

- Jestliže na lexikální rovině po *y* následuje +*s*, musí se na povrchové rovině místo *y* objevit *i*.
  `y:i <= _ +:0 s:s`
  - Opačnou implikaci v tuto chvíli nevyžadujeme. Je možné, že *y* se mění na *i* i jinde z jiných důvodů.
- Současně chceme, aby se ve stejném kontextu před *s* vkládalo *e*:
  `0:e <= y:i +:0 _ s:s`
- Vytvoříme konečný převodník, který bude převádět lexikální rovinu na povrchovou podle těchto pravidel.
  - Přesněji: převodník je automat, který jen *kontroluje,* že převádíme roviny správně.

Příklad převodníku: *baby+s*



y:i <= _ +:0 s:s

N: nekoncový stav

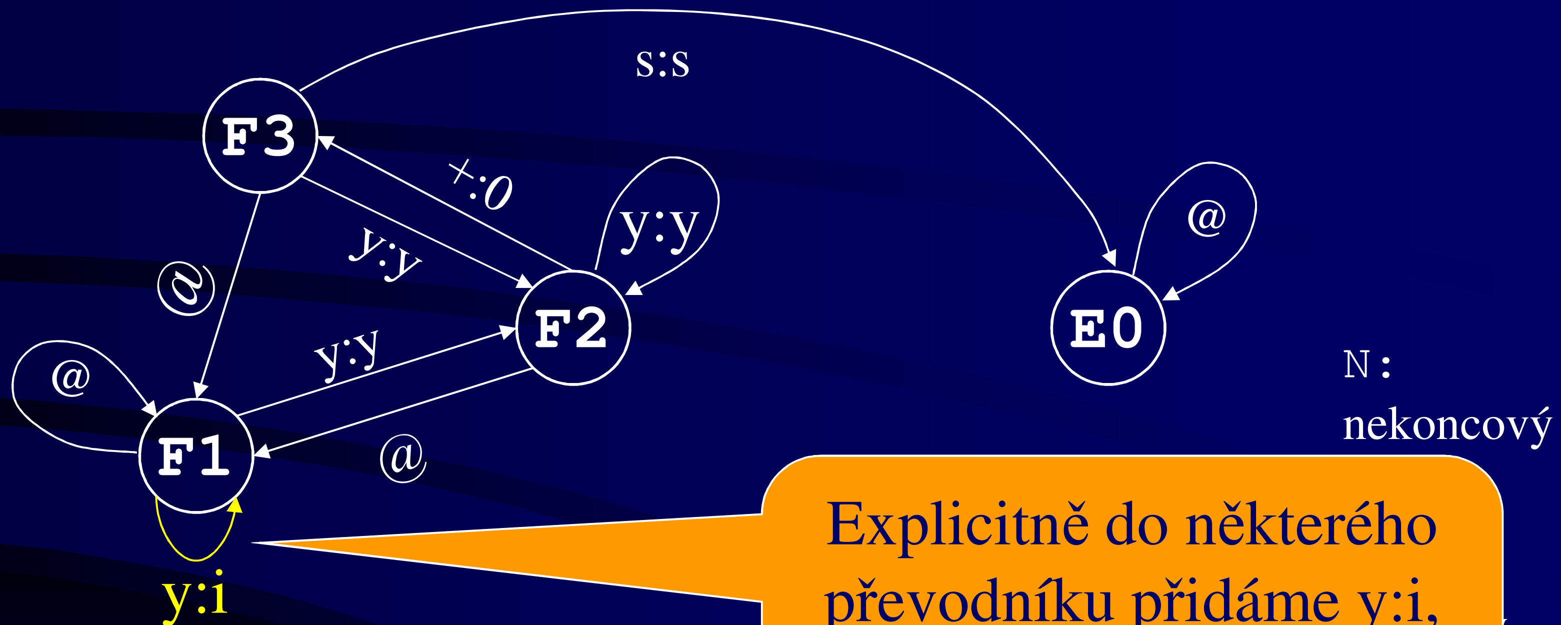
F: koncový stav

E: chybový stav

Jak získáme vstup převodníku?

- Konečný automat prostě kontroloval vstup.
- U převodníku čteme jen půl vstupu (povrch).
- Kde vezmeme druhou, lexikální polovinu?
- Známe ji předem!
  - Typické písmeno odpovídá samo sobě, např. i:i
  - Některá vznikají i fonologicky, např. y:i
  - Dopředu tedy víme, že povrchovému *i* může ve slovníku odpovídat buď *y*, nebo *i*.
  - Necháme zkontrolovat obě možnosti. Pokud projdou obě, analyzované slovo je víceznačné.

Příklad převodníku:
baby+s



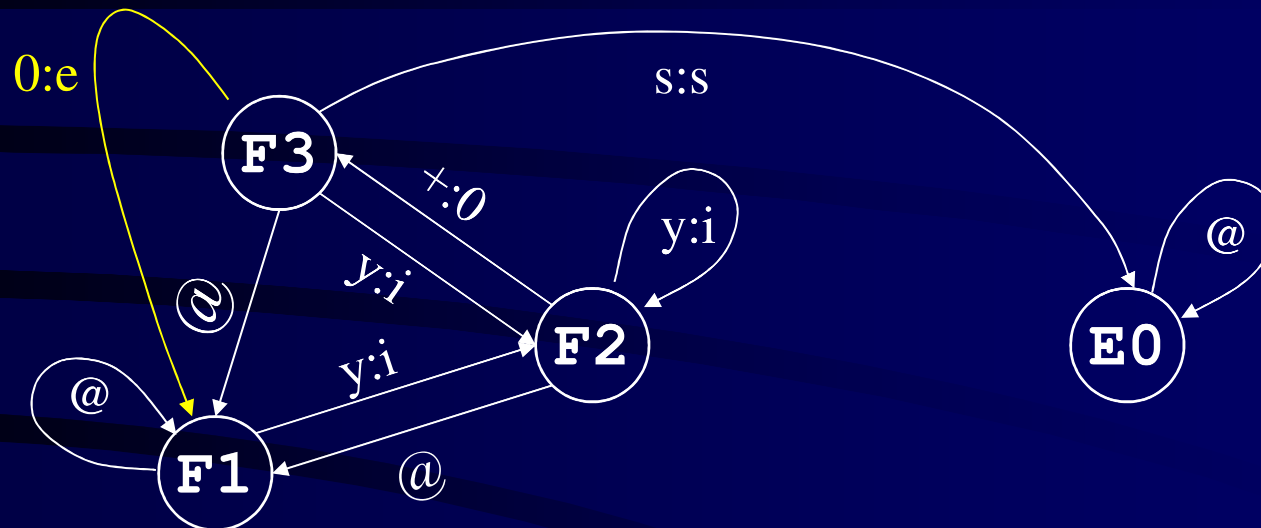
y:i <= _ +:0 s:s

Explicitně do některého převodníku přidáme y:i, aby systém o této možnosti věděl.

N: nekoncový

chybový stav

Příklad převodníku: *baby+s*



0:e <= y:i +:0 _ s:s

N: nekoncový stav

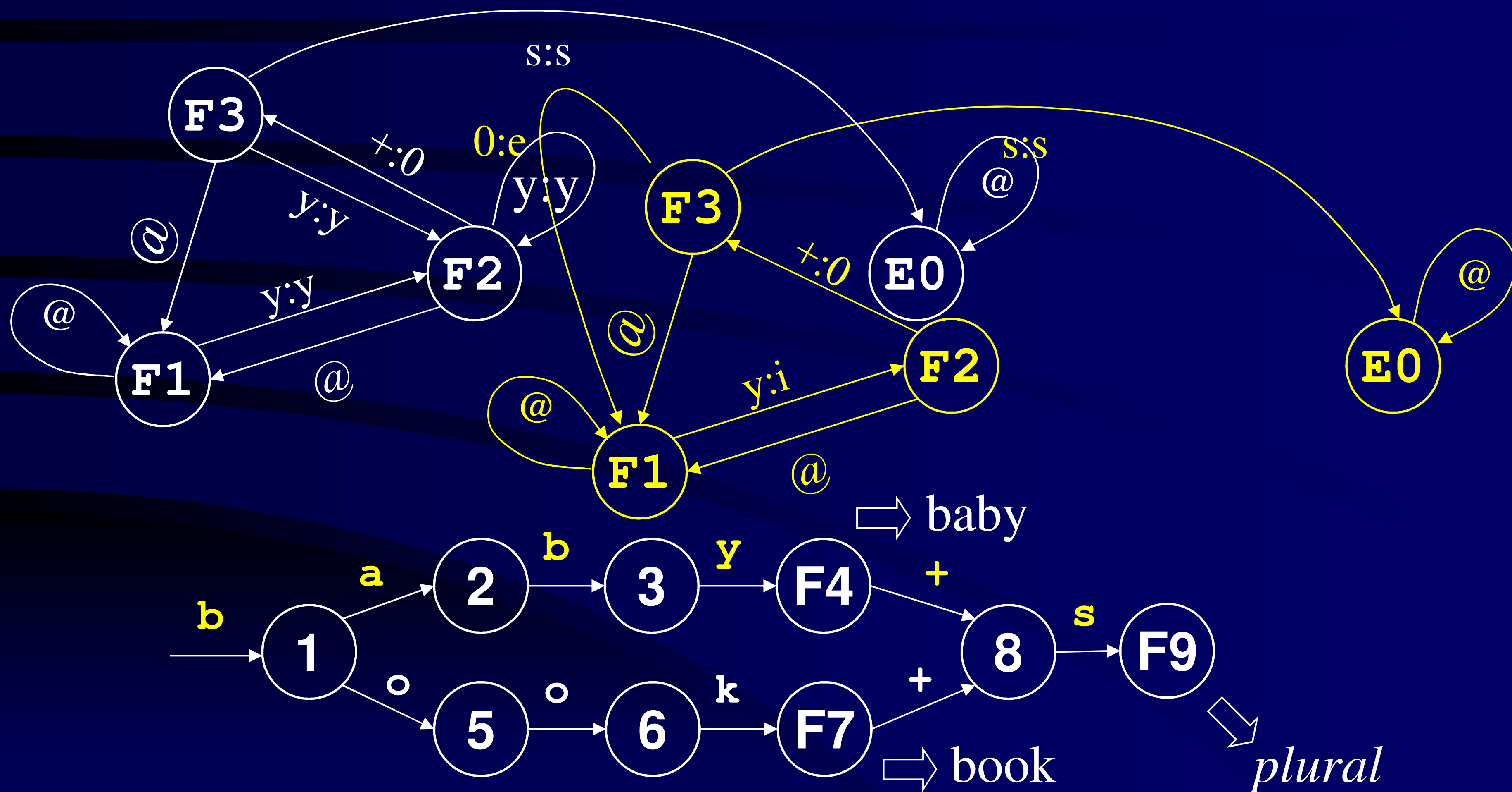
F: koncový stav

E: chybový stav

Jak to celé funguje

- Paralelní FST (včetně slovníkového FSA) mohou být zkompilovány do jednoho gigantického FST.
- Převodník ve skutečnosti sám nepřevádí, ale kontroluje.
- Převodník je však pro nás zdrojem informace, co lze na co převést (tj. co můžeme zkusit a nechat si to zkontrolovat).
  - Kromě explicitních převodních pravidel předpokládáme pro všechna x také základní převodní pravidlo x:x.

Slovník a pravidla dohromady



Dvojúrovňová morfologie: analýza

1. Inicializovat množinu cest P = { }.
2. Číst po jednom vstupní symboly.
3. Pro každý symbol x generovat všechny lexikální symboly, které mohou odpovídat prázdnému symbolu (x:0).
4. Všechny cesty v P prodloužit o všechny odpovídající páry (x:0).
5. Zkontrolovat všechna nová prodloužení cest proti fonologickému převodníku a lexikálnímu automatu. Odstranit nepovolené prefixy cest (rozpracovaná řešení).

Dvojúrovňová morfologie: analýza (pokračování)

6. Opakovat 4–5 až do dosažení maximálního možného počtu po sobě jdoucích nul.
7. Generovat všechny možné lexikální symboly (ze všech převodníků) pro aktuální symbol. Vytvořit dvojice.
8. Rozšířit každou cestu v P o všechny takové dvojice.
9. Zkontrolovat všechny cesty v P (následující krok ve FST/FSA). Odstranit všechny nemožné cesty.
10. Opakovat od bodu 3, až než skončí vstup.
11. Posbírat glosy ze slovníku ze všech přeživších cest.

Příklad analýzy

- Každý znak odpovídá sám sobě
- Navíc: **y:i**, **+:0**, **0:e**
- Vstup: *babies*
- Zkus smazané slovníkové + (**+:0**) … slovník zakáže (žádné slovo takhle nezačíná)
- Zkus **b:b** … OK (ani slovník, ani převodníky neprotestují)
- **b:b +:0** … slovníková chyba
- **b:b a:a** … OK
- **b:b a:a +:0** … slov. chyba
- **b:b a:a b:b** … OK
- **b:b a:a b:b +:0** … sl. chyba
- **b:b a:a b:b i:i** … sl. chyba
  b:b a:a b:b y:i … OK

- … **b:b y:i +:0** … OK
  … **b:b y:i +:0 +:0** … chyba
- … **y:i e:e** … chyba
  … **y:i 0:e** … OK
  … **y:i +:0 e:e** … chyba
  … **y:i +:0 0:e** … OK
- … **0:e +:0** … OK
  … **0:e +:0 +:0** … chyba
  … **+:0 0:e +:0** … chyba
- … **0:e s:s** … chyba
  … **+:0 0:e s:s** … OK
  … **0:e +:0 s:s** … OK
- … **+:0 0:e s:s +:0** … chyba
  … **0:e +:0 s:s +:0** … chyba
- Jednu z hypotéz by mohly zablokovat převodníky, kdybychom je navrhli lépe (⇔)

České příklady

- Spojením kmene a koncovky se k sobě může dostat například ď a e, které se u sebe normálně nesmí vyskytnout.

```
k á ď + e
k á ď 0 e
```

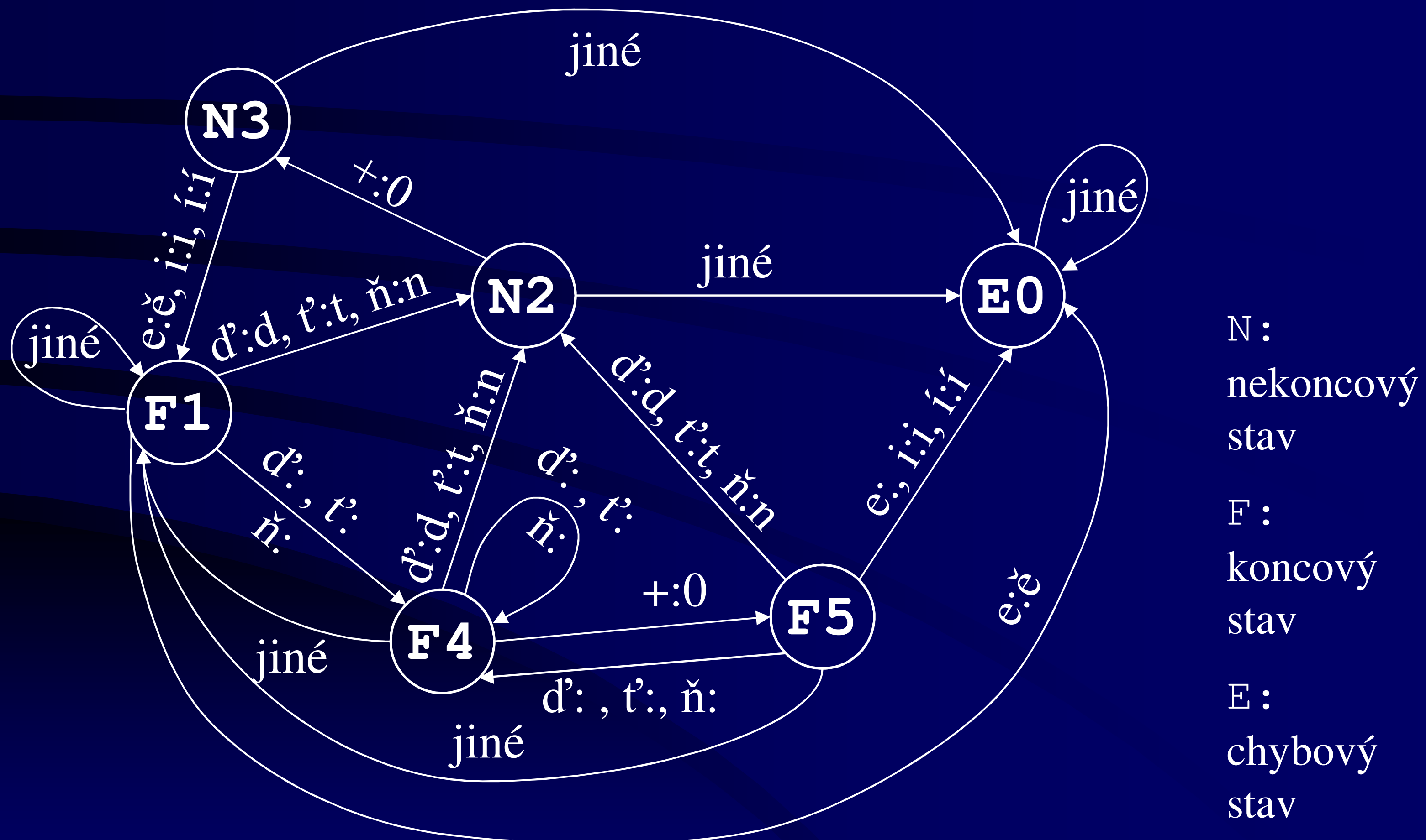
- Potřebujeme pravidlo, které v takovém a podobném případě zajistí správný přepis ďe → dě.

```
k á ď + e
k á d 0 ě
```

Příklad převodníku: ď, ť, ň na hranici morfémů

- `ď:d +:0 e:ě` je v pořádku, ostatní možnosti ne.
- Předpoklad: ďe, ďi se mohlo objevit pouze na hranici morfémů (uvnitř to můžeme napsat rovnou správně).
- Neřešíme ďě. Znak ě se do koncovky dostane leda pravidlem o změně kmenové souhlásky, jinak ne:
  - (brzda brzďe, žena žeňe, máta máťe, máma mámňe, bába bábje, matka matce, váha váze, sprcha sprše, kůra kůře, mula mule, vosa vose, lůza lůze)
- Dále neřešíme ďy (mohlo by se stát aplikováním vzoru na podstatné jméno končící na –ďa).

Příklad převodníku: ď, ť, ň na hranici morfémů

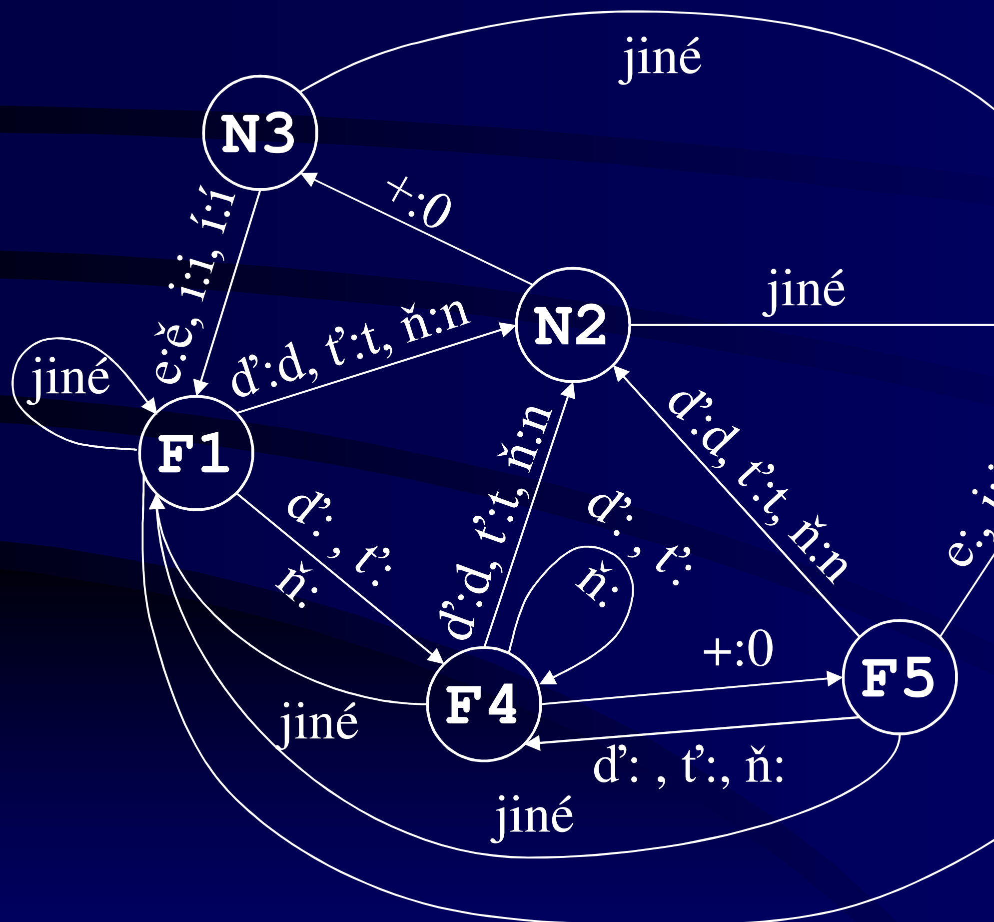


N: nekoncový stav

F: koncový stav

E: chybový stav

Příklad převodníku: ď, ť, ň na hranici m

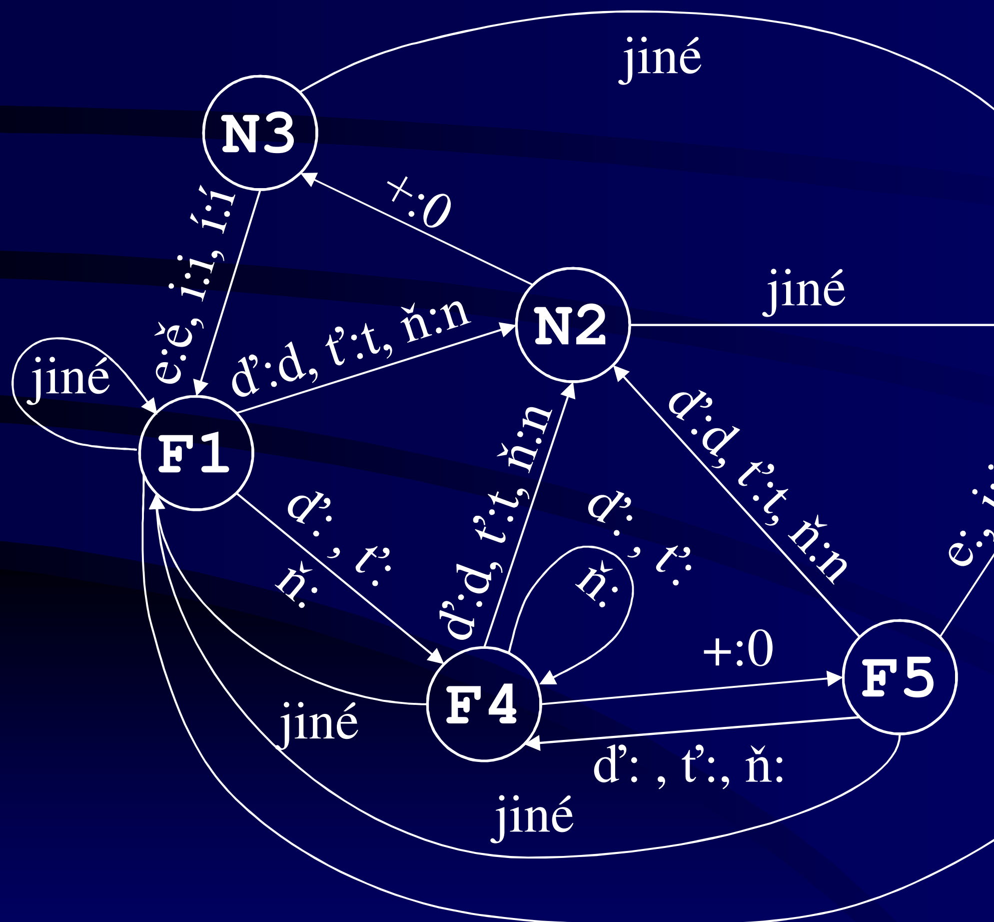


Možné převody:

- **ď:d**
- **ť:t**
- **ň:n**
- **+:0**
- **e:ě**
- **i:i**
- **í:í**

E: chybový stav

Příklad převodníku: ď, ť, ň na hranici m

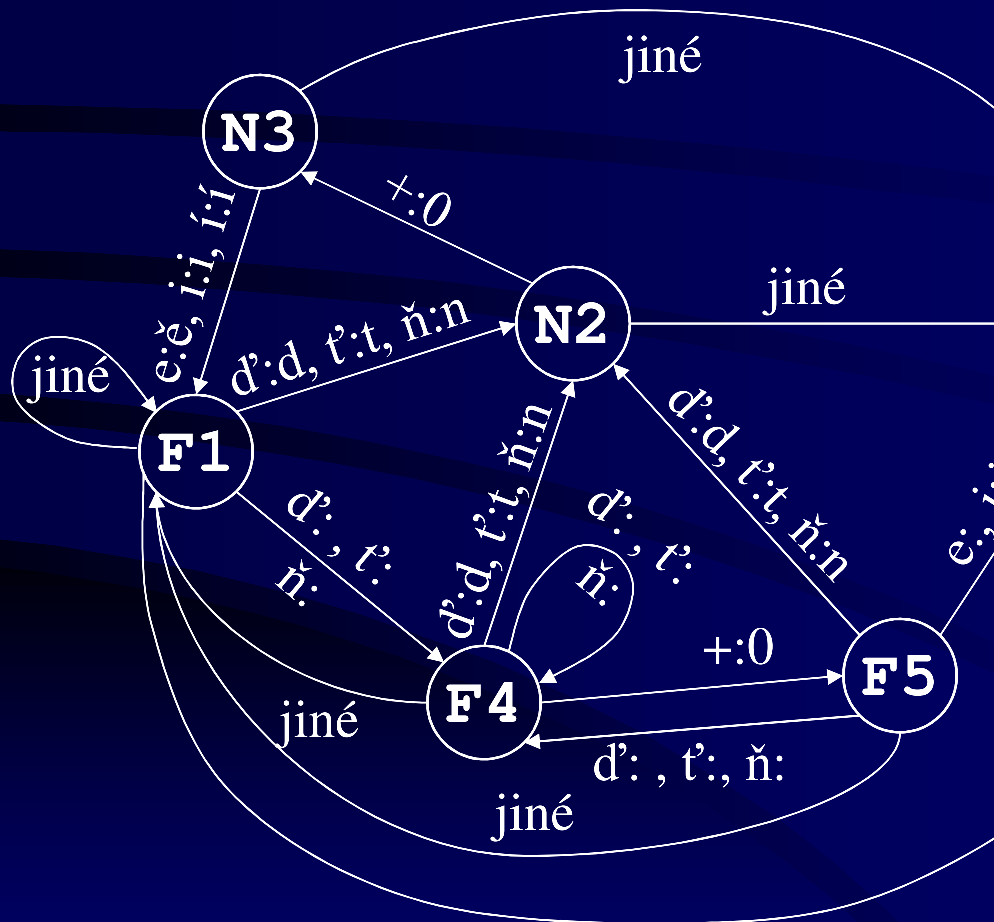


Možné převody:

- **ď:d @**
- **ť:t @**
- **ň:n @**
- **+:0**
- **e:ě @**
- **i:i**
- **í:í**
- **@:@**

stav

Příklad převodníku: ď, ť, ň na hranici m

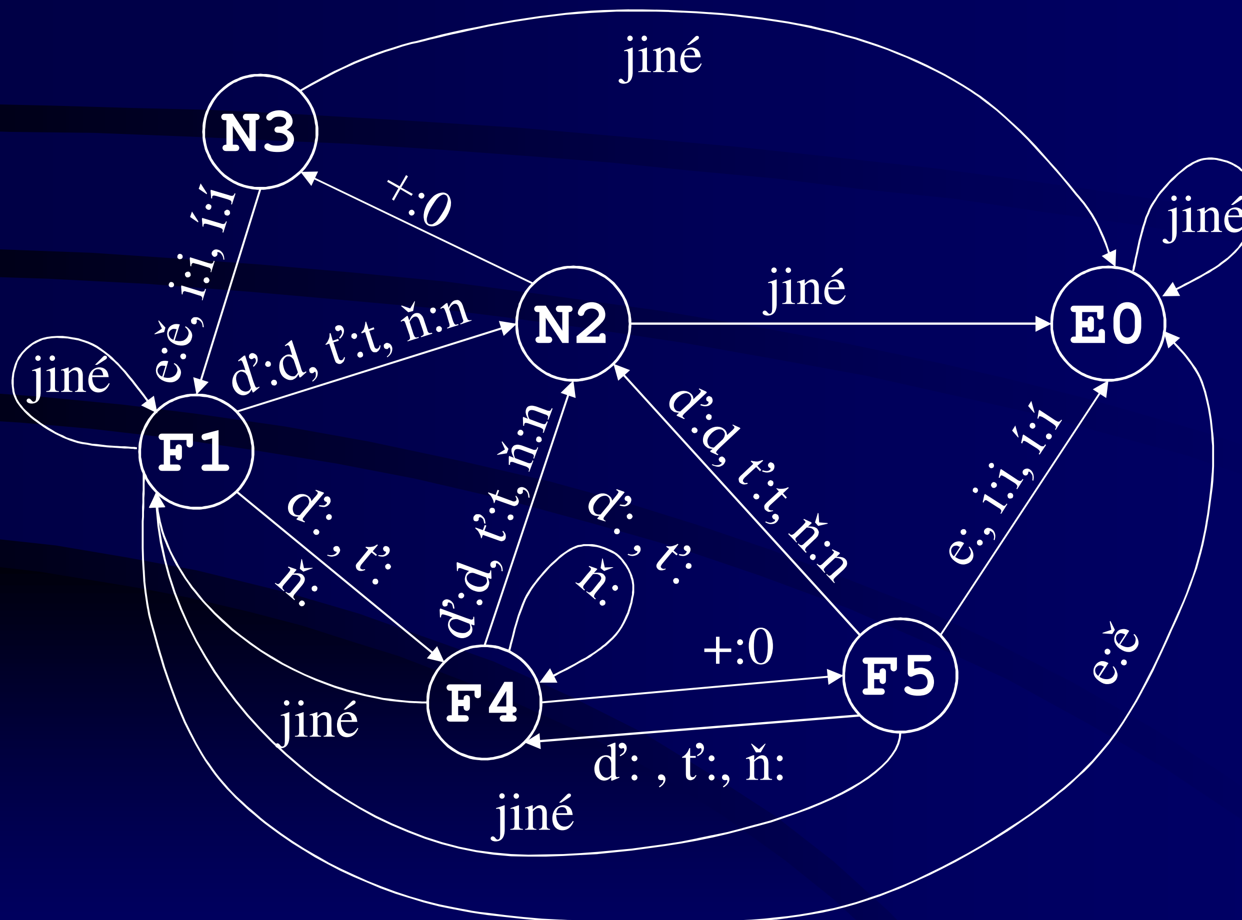


Přidat **abecedu:**

- **ď:d ď:ď d:d**
- **ť:t ť:ť t:t**
- **ň:n ň:ň n:n**
- **+:0**
- **e:ě e:e ě:ě**
- **i:i**
- **í:í**
- **x:x …**

stav

Příklad převodníku: ď, ť, ň na hranici morfémů



N: nekoncový stav

F: koncový stav

E: chybový stav

Zápis převodníku pomocí matice

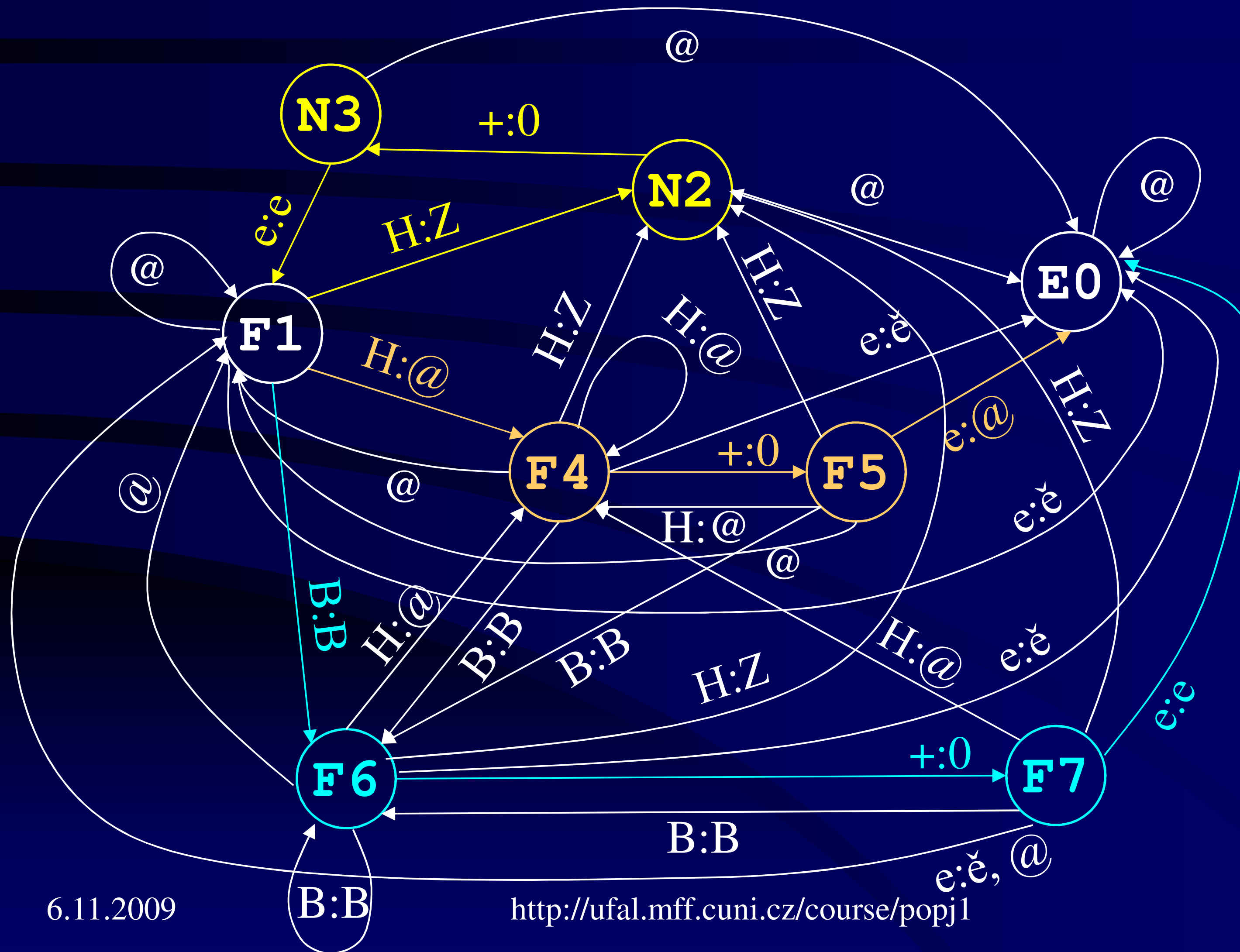
```
RULE "[ď:d | ň:n | ť:t] <=> _ +:0 [e:ě | i:i
                                  | í:í]" 5 12

     ď ň ť ď ň ť + e i í e @
     d n t @ @ @ 0 ě i í @ @
  1: 2 2 2 4 4 4 1 0 1 1 1 1
  2. 0 0 0 0 0 0 3 0 0 0 0 0
  3. 0 0 0 0 0 0 0 1 1 1 0 0
  4: 2 2 2 4 4 4 5 1 1 1 1 1
  5: 2 2 2 4 4 4 1 0 0 0 0 1
```

Změkčování *váha – váze*

- váha – váze
- sprcha – sprše
- matka – matce
- kůra – kůře
- vláda – vládě
- máta – mátě
- žena – ženě
- Olga – Olze
- bába – bábě
- karafa – karafě
- máma – mámě
- chrpa – chrpě
- jíva – jívě
- Naďa – Nadě
- Jíťa – Jítě
- Áňa – Áně

Změkčování *váha – váze*



H:Z = g:z | h:z | ch:š | k:c | r:ř

B:B = b:b | f:f | m:m | p:p | v:v | w:w | q:q | d:d | t:t | n:n | ď:d | ť:t | ň:n

Ukázka v PC Kimmo

- `r ženě matce Bláže Nadě…`
- Oddělit vzory *žena* a *růže* pomocí tříd pokračování
- `r Nadi`
- `g Naď+y`

Příklady dvojúrovňových pravidel v češtině

- Změny kmenových souhlásek.

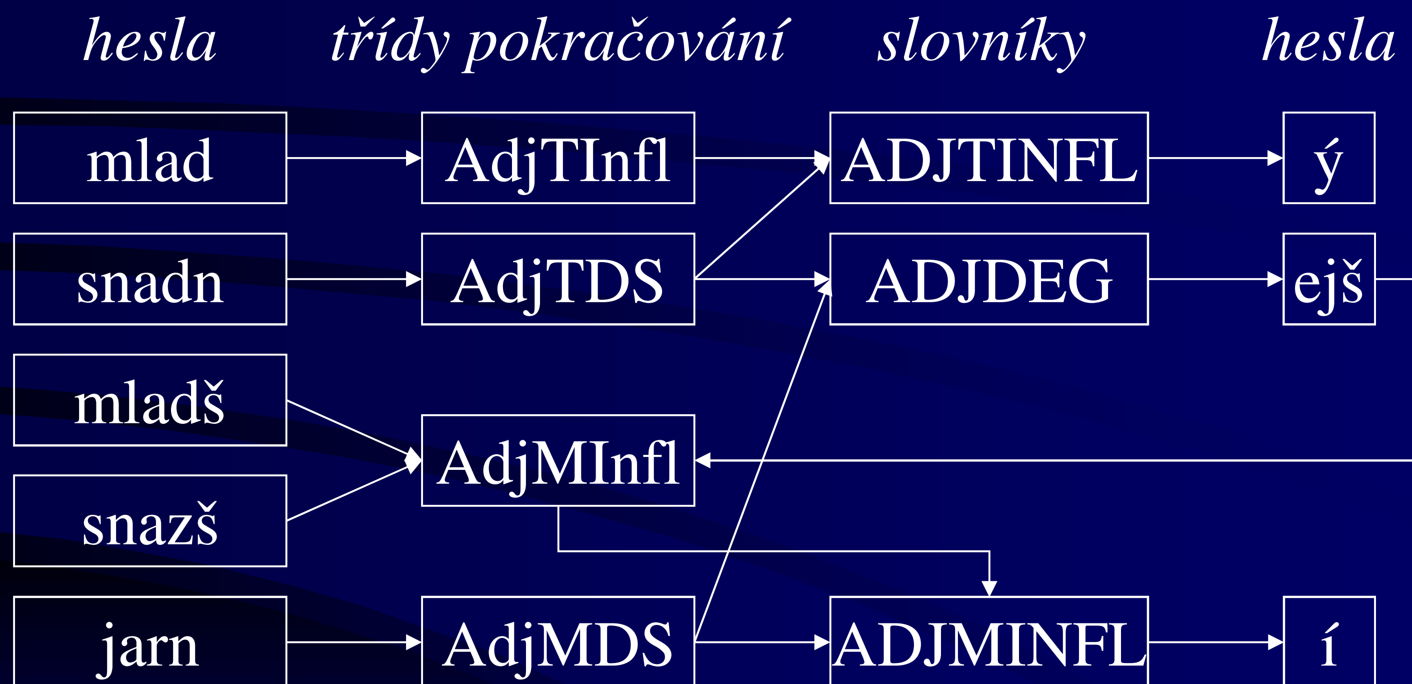
```
m a t E K + e
m a t 0 c 0 e
```

- Vkládání nebo mazání *e*.

```
m a t E K
m a t e k
```

- Přechody mezi přítomným, minulým a infinitivním kmenem sloves.
  - Změkčování kmenové souhlásky v rozkazovacím způsobu.

PC Kimmo: česká přídavná jména



Ukázka v PC Kimmo

- Přídavná jména: nepravidelné stupňování
- `r mladý mladší *mladější`
- `r snadný snazší snadnější`
- `r jarní jarnější`
- I zde zafungovalo změkčování:
  jarn + ejš + í = jarnější
- `r nejmladší *nejmladý`

Dlouhé závislosti

- Nevýhoda DÚM:
  - Závislosti na velkou vzdálenost se zachycují těžkopádně!

Příklad z němčiny

- Přehlásky v němčině (zjednodušeno)

  u ↔ ü jestliže (ne právě když) následuje c h e r (Buch → Bücher)

  pravidlo: **`u:ü`** ⇐

  `_ c:c h:h e:e r:r`

  FST:

  Buch:

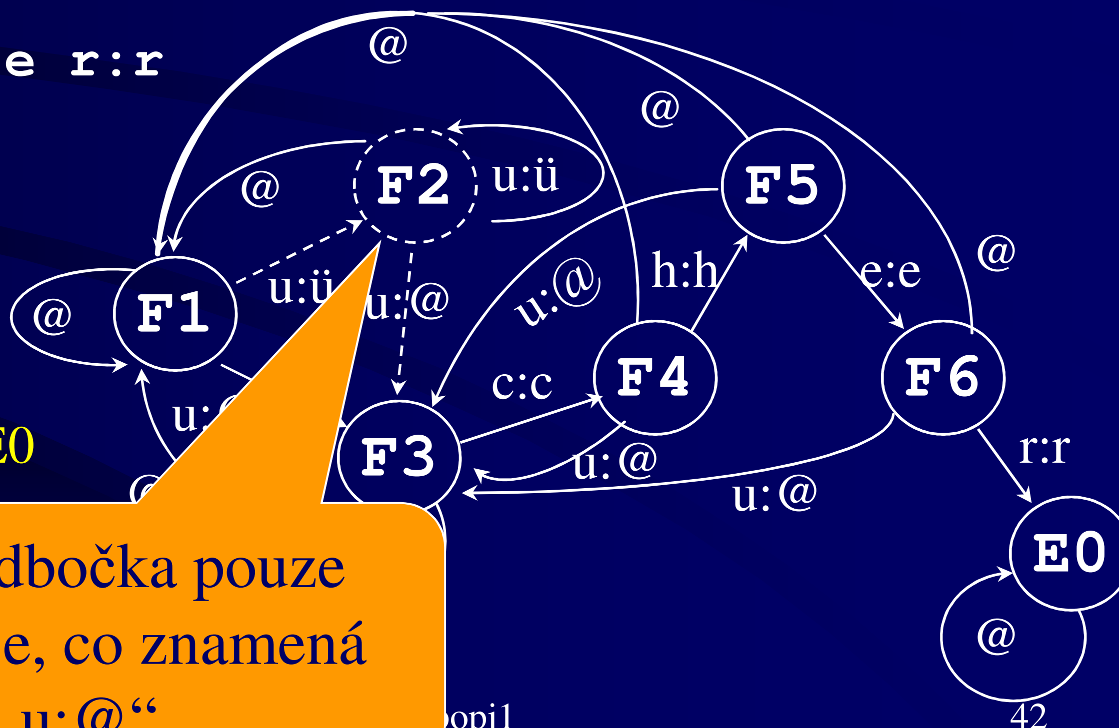
  F1 F3 F4 F5

  Bucher:

  F1 F3 F4 F5 F6 E0

  Buck:

  F1



Tahle odbočka pouze vymezuje, co znamená „u:@“

Příklad z němčiny

- *Buch / Bücher, Dach / Dächer, Loch / Löcher*
- Kontext by měl navíc obsahovat +:0 a možná i testovat konec (#)
  - Jinak by se *Sucherei* (hledání) považovalo za chybu!
  - Musíme poznat nejen že jde o příponu, ale také množného čísla.
  - Protipříklady:
    - *Kocher* (vařič), zde přípona *er* pouze odvozuje od slovesa *kochen* (vařit). *Kocher* je stejný v jednotném i množném čísle! Nechceme, aby se pletl s *Köcher* (toulec), ani aby se *Kocher* bez přehlásky považovalo za chybu!
    - *Besucher* (návštěvník), odvozeno od *Besuch* (návštěva), stejné jednotné i množné číslo, **Besücher* neexistuje!
- Závislosti na větší vzdálenost se zachycují těžkopádně.
  - Např. *Kraut / Kräuter* má jiné mezilehlé symboly, takže je to jakoby jiné pravidlo.

Dvojúrovňovost a slovník

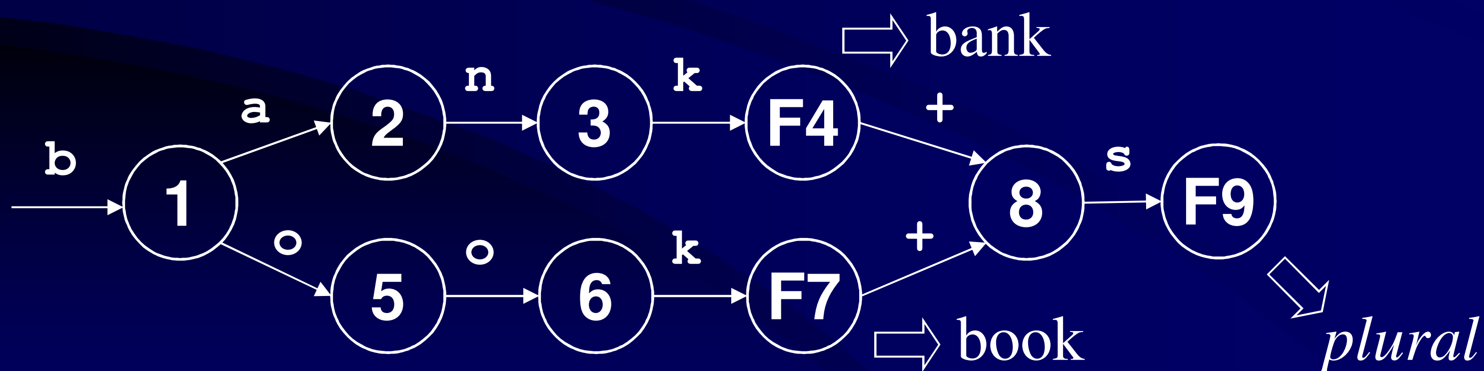
- Slovník obsahuje pouze symboly na lexikální (slovníkové) rovině.
  - Vztah k povrchové rovině je vyjádřen jen převodníky.
- Zato jsou tu *glosy* (výstup analýzy).
- Celkem tedy systém obsahuje 3 vrstvy!
  - Povrchová rovina (SL):
    - *book*
  - Lexikální rovina (LL, slovo rozdělené na morfémy):
    - *book+s*
  - Glosy (lemma, slovní druh, značka, cokoliv)
    - *N(book)+plural*

Analýza a syntéza

- Analýza je přechod z povrchové roviny na lexikální.
  - books => book+s   book +*plural*
- Syntéza je přechod z lexikální roviny na povrchovou.
  - Typickým vstupem jsou spíše glosy než morfémy.
  - book +*plural* => book+s => books

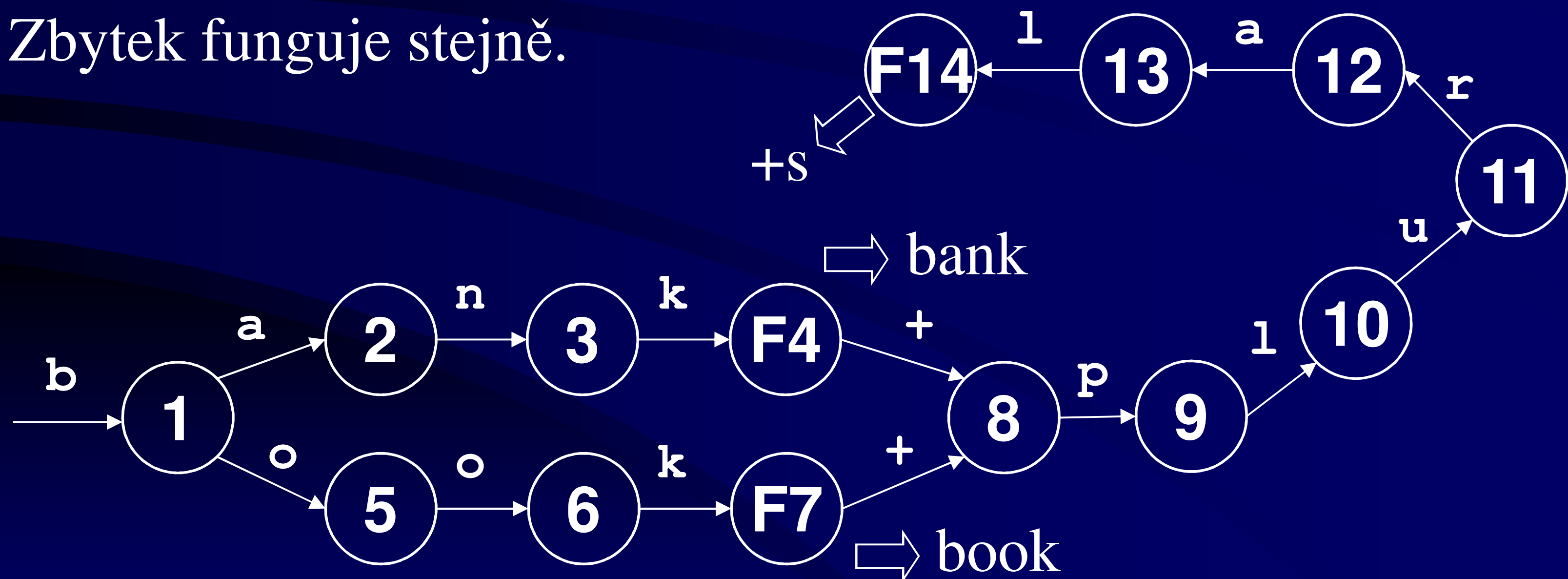
Slovník pro analýzu

- Implementován jako konečný automat (trie).
- Zkompilován ze seznamu řetězců a slovníkových odkazů.
- Podslovníky pro kmeny, předpony, přípony, koncovky.
- Poznámky (glosy) na konci každého podslovníku.



Slovník pro syntézu

- Prohodíme povrchovou a lexikální rovinu (glosy).
- Opět lze automaticky zkompilovat ze stejného seznamu jako slovník pro analýzu.
- Zbytek funguje stejně.



Syntéza v PC Kimmo verze 2

- Původně jen „generování“:
  - `g žen+e`
- Syntéza (nová v PCK v. 2):
  - `l synthesis-lexicon cs.lex`
  - `s N(žena) :žena+SG+LOC`

Jak naplnit slovník?

- Máme anotovaný korpus (lemmata, značky)?
  - Vznikl s pomocí existujícího morfologického analyzátoru
  - My k tomu analyzátoru nemáme přístup a chceme svůj vlastní
- Jednoduché:
  - Všechna slova podle vzoru *žena*:
    - Vytáhnout slova se značkou **`NNFS1.*`** a lemmatem končícím na *–a*
    - Některá slova se nevyskytla v 1. pádě jednotného čísla. Nechceme-li o ně přijít, opakujeme postup pro jiný charakteristický tvar vzoru *žena*.

Jak naplnit slovník?

- PDT 1.0 obsahuje 141 798 výskytů **`NNF`**
- 26 346 jedinečných (všechny tvary)
- 10 844 má lemma končící na *–a*
- 2109 z nich je **`NNFS1`**
- V tom jsou ještě zvlášť započítána velká písmena na začátku věty a lemmata s přídavnými informacemi (`zmije_,a`)
- Od kmenů nejsou odděleny případné předpony *(pře-stavba)*

Jak naplnit slovník?

- Máme pouze neanotovaný korpus
  - Jaký vzor má dosud nepokryté slovo?
  - Hypotéza: slovo *města* patří pod vzor *žena*.
  - Vygenerovat PC Kimmem všechny tvary tohoto slova podle daného vzoru, pak hledat jejich výskyty v korpusu.
  - Problém 1: co je kmen a co koncovka?
    - Zkusit postupně m+ěsta, mě+sta, měs+ta, měst+a, města+λ
  - Problém 2: co s hláskovými změnami? Co když zkoumám slovo *matce*?
    - Zkusit postupně všechny kombinace hláskových změn, které dovoluje příslušný soubor s pravidly pro PC Kimmo

Dvojúrovňová gramatika

- Nadstavba nad Kimmem (Lauri Karttunen, Xerox)
- Způsob, jak popsat pravidlo, pro které chceme převodník
- Tři části:
  - Dvojice horní-dolní symbol = změna
  - Kontext změny
  - Vztah mezi změnou a kontextem (operátor)
- Příklad: v tomto pravém kontextu *musíme* ď změnit na d
- Zápis:

```
ď:d <= _ +:0 e:@
```

- (Bez dalších pravidel jsme tím ovšem povolili ď:d i v jiných kontextech.)

Dvojúrovňová gramatika

- **`x:y <= lk _ pk`**
  Jestliže se x vyskytuje mezi levým kontextem lk a pravým kontextem pk, pak musí být realizováno jako y. V daném kontextu je x realizováno jako y *vždy*.
- **`x:y => lk _ pk`**
  Jako y je x realizováno *pouze* v tomto kontextu.
- **`x:y <=> lk _ pk`**
  Jako y je x realizováno *právě* v tomto kontextu.
- **`x:y /<= lk _ pk`**
  V tomto kontextu není x *nikdy* realizováno jako y.

Morphological and Syntactic Analysis

Multi-Level Finite State Rules

Daniel Zeman

http://ufal.mff.cuni.cz/~zeman/

XFST

- Xerox Finite State Toolkit
  - xfst, lexc, tokenize, lookup
  - Binaries and API for multiple operating systems
  - Kenneth R. Beesley, Lauri Karttunen: *Finite State Morphology*. CSLI Publications, 2003
- http://www.fsmbook.com/
  - http://www.stanford.edu/~laurik/.book2software/
  - http://cs.haifa.ac.il/~shuly/teaching/06/nlp/xfst-tutorial.pdf
  - http://cs.haifa.ac.il/~shuly/teaching/06/nlp/fst2.pdf
- Current version uses UTF8 by default.
- Some support for reduplication (!)
  - At compile time, morpheme *m* can be replaced by regex *m*^2
  - It simulates having two entries in the lexicon: one for the normal form and one for the reduplicated one.

Foma

- Open-source finite-state toolkit
  - In contrast, xfst comes without sources and with some copyright restrictions
- Claims compatibility with Xerox tools
  - But also supports Perl-style regular expressions
- Now integrated in Apertium (open-source rule-based machine translation framework)
- Home: https://code.google.com/p/foma/
  - Publication: http://www.aclweb.org/anthology-new/E/E09/E09-2008.pdf

Foma vs. Kimmo

- Multiple levels
  - Sequence of ordered rewrite rules
  - Even lexicon supports two levels (TAG:suffix)
- Regular expressions
  - Instead of directly encoding transducers
  - Supports usual FSM algorithms (minimization etc.)
- Sequence of rules still compiled into one FST
  - We still have one upper and one lower language

Compiling Regular Expressions: regex

- `regex a+;`
- `regex c a t | d o g;`
- `regex ?* a ?*;`

- `regex [a:b | b:a]*;`
- `regex [c a t]:[k a t u a];`
- `regex b -> p, g -> k, d -> t || _ .#.;`

Foma Operators

- (space) … concatenation
- `|` … union
- `*` … Kleene star
- `&` … intersection
- `~` … complement
- Single- and multi-character symbols
  - Supports Unicode
- `0` … empty string (epsilon)
- `?` … any symbol (similar to "." in Perl, grep etc.)
- `( a )` … "a" is optional (as "a?" in Perl)

Rozdíl mezi dvojtečkou a šipkou

- Dvojtečka ovlivňuje konkrétní pozici nebo posloupnost pozic.
- Dvojtečky se používají v regulárních výrazech, které omezují množinu slov patřících do jazyka.
- Regulární výrazy s šipkou vedou na převodníky, které přijímají libovolný řetězec, ale pokud v něm narazí na hledaný znak, nahradí ho.
- Šipka se implementuje pomocí dvojtečky.

Testing Automata against Words

```
foma[0]: regex ?* a ?*;
261 bytes. 2 states, 4 arcs, Cyclic.
foma[1]: down
apply down> ab
ab
apply down> bbx
???
apply down> CTRL+D
foma[1]:
```

Labeling FSMs: define

```
foma[0]: define V [a|e|i|o|u];
defined V: 317 bytes. 2 states, 5 arcs, 5 paths.
foma[0]: define StartsWithVowel [V ?*];
defined StartsWithVowel: 429 bytes. 2 states, 11 arcs, Cyclic.
foma[0]:
```

Rewrite Rules

```
foma[0]: regex a -> b;
290 bytes. 1 states, 3 arcs, Cyclic.
foma[1]: down
apply down> a
b
apply down> axa
bxb
apply down> CTRL+D
```

Accepts any input.

Changes *a* to *b*.

Conditional Replacement

```
foma[0]: regex a -> b || c _ d ;
526 bytes. 4 states, 16 arcs, Cyclic.
foma[1]: down cadca
cbdca
foma[1]:
```

Multiple Contexts

```
foma[0]: regex a -> b || c _ d, e _ f;
890 bytes. 7 states, 37 arcs, Cyclic.
foma[1]: down
apply down> cadeaf
cbdebf
apply down> a
a
apply down> CTRL+D
```

Parallel Rules
End-of-Word Symbol

```
foma[0]: regex b -> p, g -> k, d -> t ||
_ .#. ;
634 bytes. 3 states, 20 arcs, Cyclic.
foma[1]: down
apply down> cab
cap
apply down> dog
dok
apply down> dad
dat
```

Composition of Rules

```
foma[0]: define Rule1 a -> b || c _ ;
defined Rule1: 384 bytes. 2 states, 8 arcs, Cyclic.
foma[0]: define Rule2 b -> c || _ d ;
defined Rule2: 416 bytes. 3 states, 10 arcs, Cyclic.
foma[0]: regex Rule1 .o. Rule2;
574 bytes. 4 states, 19 arcs, Cyclic.
foma[1]: down
apply down> cad
ccd
apply down> ca
cb
apply down> ad
ad
```

Review

- `regex` regular-expression;
  - compile regular expression and put it on the stack
- `define` name regular-expression;
  - name a FST/FSM using regex; do not put it on the stack
- `view` (`view net`)
  - (Linux only) display the compiled regex from stack graphically in a window
- `net` (`print net`)
  - textual net description
- `down` <word> (apply down)
  - run a lexical word through a transducer (generation)
- `up` <word> (apply up)
  - run a surface word through a transducer (analysis)
- `words` (print words)
  - print all the words an automaton accepts
- `lower-words`
  - only lower side of an FST
- `upper-words`
  - only upper side of an FST

Lexicon in lexc Format

- Create the file, then load it to Foma

```
LEXICON Root
cat   Suff;
dog   Suff;
horse Suff;

LEXICON Suff
s #;
  #;
```

Load Lexicon to Foma

```
foma[0]: read lexc simple.lexc
Root…3, Suff…2
Building lexicon…Determinizing…Minimizing…Done!
575 bytes. 13 states, 15 arcs, 8 paths.
foma[1]: print words
horse horses dog dogs cat cats
foma[1]: define Lexicon;
```

Or alternatively:

```
foma[0]: define Lexicon [c a t|d o g|…] (s);
```

Example English lexc File

```
Multichar_Symbols
+N +V +PastPart
+Past +PresPart +3P
+Sg +Pl
LEXICON Root
Noun ;
Verb ;
LEXICON Noun
cat  Ninf;
city Ninf;
```

- `LEXICON Ninf`
- `+N+Sg:0  #;`
- `+N+Pl:^s #;`

  ! ^ is our morpheme boundary

Put It All Together

- Lexical string = `city+N+Pl`
- Lexicon transducer: `city+N+Pl` → `city^s`
- *y* → *ie* rule: `city^s` → `citie^s`
- Remove ^: `citie^s` → `cities`
- Surface string = `cities`

Put It All Together

```
foma[0]: read lexc english.lexc
foma[1]: define Lexicon;
foma[0]: define YRepl y -> i e || _ "^" s;
foma[0]: define Cleanup "^" -> 0;
foma[0]: regex Lexicon .o. YRepl .o. Cleanup;
foma[1]: lower-words
cat cats city cities …
```

Irregular Forms

```
LEXICON Verb
beg Vinf;
make+V+PastPart:made #;  ! bypass Vinf
make+V #;
…
```

Priority Union

```
foma[1]: define Grammar;
foma[0]: define Exceptions [m a k e "+V"
"+PastPart"]:[m a d e];
foma[0]: regex [Exceptions .P. Grammar];
foma[1]: down
apply down> make+V+PastPart
made
apply down> CTRL+D
```

Alternate Forms

- English: *cactus*+N+Pl → *cactuses, cacti*

```
foma[0]: define Parallel [c a c t u s
"+N" "+Pl"]:[c a c t i];
foma[1]: regex Parallel | Grammar;
…
```

Long-Distance Dependencies

- Constraining co-occurrence of morphemes
- Create a filter before or after lexical level
- Usual format ~$[PATTERN];
- "The language does not contain PATTERN."

```
define SUPFILT ~$[ "[Sup]" ?+ "[Pos]" ];
define MORPH SUPFILT .o. LEX .o. RULES;
```

Flag Diacritics

- Invisible symbols to control co-occurrence:
  - U … unify features @U.*feature.value*@
  - P … positive set @P.*feature.value*@
  - N … negate @N.*feature.value*@
  - R … require feat/val @R.*feature(.value)*@
  - D … disallow feat/val @D.*feature(.value)*@
  - C … clear feature @C.*feature*@
  - E … require equal feat/val @E.*feature.value*@

Flag Diacritics to Control Czech Superlatives

- `Multichar_Symbols Sup+ +Pos +Comp @P.SUP.ON@ @D.SUP@`
- `LEXICON AdjSup`
  `@P.SUP.ON@Sup+:@P.SUP.ON@nej^ Adj;`
- `LEXICON AhardDeg`
  `@D.SUP@+Pos:@D.SUP@ Ahard;`
  `+Comp:^ejš Asoft;`

Non-interactive Runs

```
foma[1]: save stack en.bin
Writing to file en.bin.
foma[1]: exit

$ echo begging | flookup en.bin
begging beg+V+PresPart

$ echo beg+V+PresPart | flookup -i en.bin
beg+V+PresPart begging
```

Czech Lexicon Example

- **Multichar_Symbols** +NF +Masc +Fem +Neut +Sg +Pl +Nom +Gen +Dat +Acc +Voc +Loc +Ins
- **LEXICON Root**
  ```
  Noun;
  Adj;
  AdjSup;
  ```
- **LEXICON Noun**
  ```
  žena:žen   NFzena;
  matka:matk NFzena;
  ```
- **LEXICON NFzena**
  ```
  +NF+Sg+Nom:^a    #;
  +NF+Sg+Gen:^y    #;
  +NF+Sg+Dat:^e    #;
  …
  ```

Czech Rules Example

- `# matk + ^0 --> matek`
  `define NFPlGenEInsertion [t k]->[t e k] || _ "^" λ;`
- `# matke -> matce, žene -> žeňe`
  `define NFSgDatPalatalization k->c, n->ň || _ "^" e;`
- `# ďe ťe ňe -> dě tě ně`
  `define DeTeNe [ď "^" e]->[d "^" ě], [ť "^" e]->[t "^" ě], [ň "^" e]->[n "^" ě];`
- `# Finally erase temporary symbols.`
  `define Surface "^" -> 0, λ -> 0;`
- `read lexc cs.lexc`
  `define Lexicon;`
  `regex Lexicon .o. NFPlGenEInsertion .o. NFSgDatPalatalization .o. DeTeNe .o. Surface;`

Foma: Czech Demo

- `cd ~/nastroje/foma/cs`
- `../foma -l cs/cs.foma`

Unsorted Notes

- Rozdíl mezi dvojtečkou a šipkou?
  - Šipka se implementuje pomocí dvojtečky.
  - Dvojtečka ovlivňuje konkrétní pozici nebo posloupnost pozic.
  - Regexy s šipkou vedou na převodníky, které přijímají libovolný řetězec, ale pokud v něm narazí na hledaný znak, nahradí ho.
  - Dvojtečky se používají v regexech, které omezují množinu slov patřících do jazyka.
- Proč označují hranici morfému znakem „^“? Proč mi nefunguje „+“?
- Můj malý český příklad
- Okopírovat z Linuxu obrázek nějaké sítě (třeba té české)