Predicting Pseudoknots Without Hacking in C

Master's Thesis Project

Maik Riechert

HTWK Leipzig

4th October 2012
10. Herbstseminar der Bioinformatik

Problem

Input: RNA primary structure

CAAUUUCUGAAAAUUUCAC

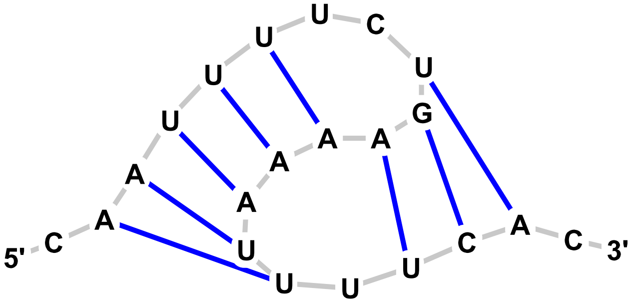
(from tobacco etch virus)

Problem

Input: RNA primary structure

CAAUUUCUGAAAUUUCAC
(from tobacco etch virus)

Output: “best” secondary structure (including pseudoknots)

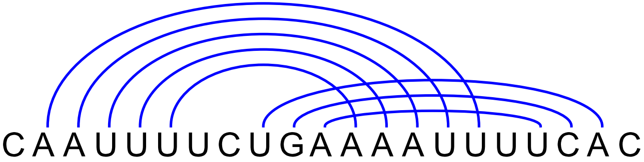


Problem

Input: RNA primary structure

CAAUUUUCUGAAAAUUUUCAC
(from tobacco etch virus)

Output: “best” secondary structure (including pseudoknots)

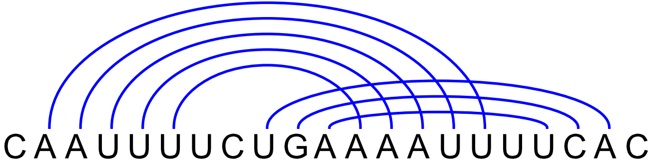


Problem

Input: RNA primary structure

CAAUUUUCUGAAAAUUUUCAC
(from tobacco etch virus)

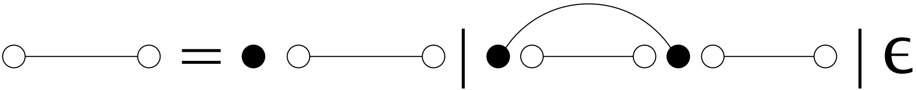
Output: “best” secondary structure (including pseudoknots)



w = . (((((. . [[[ ))))) .]]] .

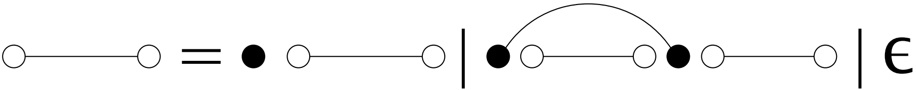
Decomposing secondary structures with grammars

Context-free grammars for nested secondary structures



Decomposing secondary structures with grammars

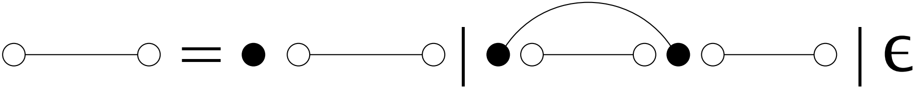
Context-free grammars for nested secondary structures



$S = \mathbf{.}\ S \mid \mathbf{(}S\mathbf{)}S \mid \epsilon$

Decomposing secondary structures with grammars

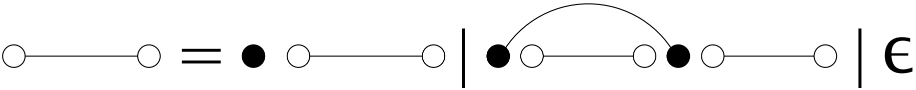
Context-free grammars for nested secondary structures



$$S = \texttt{.}\ S \mid PS \mid \epsilon \qquad P = \texttt{(}S\texttt{)}$$

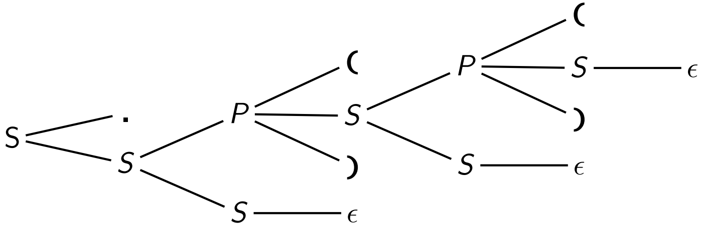
Decomposing secondary structures with grammars

Context-free grammars for nested secondary structures



$S = \mathbf{.}\ S \mid PS \mid \epsilon \qquad P = \mathbf{(}S\mathbf{)}$

Derivation tree = decomposed secondary structure



Deriving (and decomposing) sec. structures with grammars

Context-free grammars for primary structures

$S = BS \mid PS \mid \epsilon$

$P = \mathbf{a}S\mathbf{u} \mid \mathbf{u}S\mathbf{a} \mid \mathbf{g}S\mathbf{c} \mid \mathbf{c}S\mathbf{g} \mid \mathbf{g}S\mathbf{u} \mid \mathbf{u}S\mathbf{g}$

$B = \mathbf{a} \mid \mathbf{u} \mid \mathbf{g} \mid \mathbf{c}$

Deriving (and decomposing) sec. structures with grammars

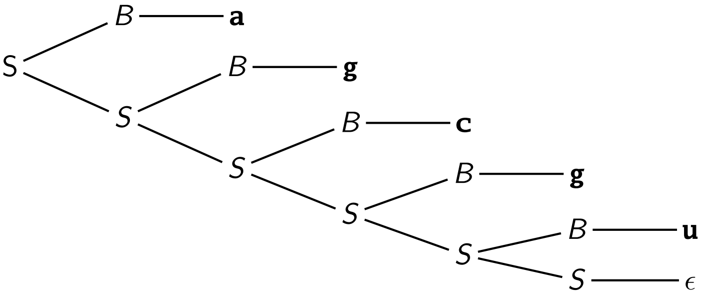
Context-free grammars for primary structures

$S = BS \mid PS \mid \epsilon$

$P = \mathbf{a}S\mathbf{u} \mid \mathbf{u}S\mathbf{a} \mid \mathbf{g}S\mathbf{c} \mid \mathbf{c}S\mathbf{g} \mid \mathbf{g}S\mathbf{u} \mid \mathbf{u}S\mathbf{g}$

$B = \mathbf{a} \mid \mathbf{u} \mid \mathbf{g} \mid \mathbf{c}$

Derivation tree = implicit (and decomposed) secondary structure

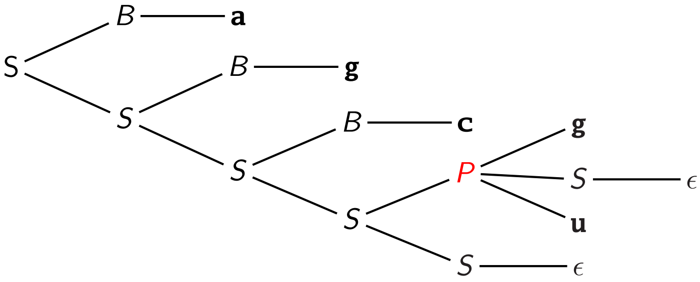


Deriving (and decomposing) sec. structures with grammars

Context-free grammars for primary structures

$S = BS \mid PS \mid \epsilon$
$P = \mathbf{a}S\mathbf{u} \mid \mathbf{u}S\mathbf{a} \mid \mathbf{g}S\mathbf{c} \mid \mathbf{c}S\mathbf{g} \mid \mathbf{g}S\mathbf{u} \mid \mathbf{u}S\mathbf{g}$
$B = \mathbf{a} \mid \mathbf{u} \mid \mathbf{g} \mid \mathbf{c}$

Derivation tree = implicit (and decomposed) secondary structure

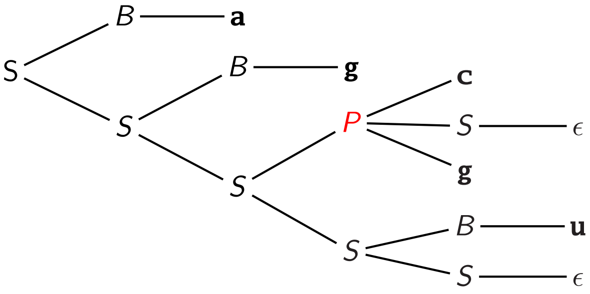


Deriving (and decomposing) sec. structures with grammars

Context-free grammars for primary structures

$S = BS \mid PS \mid \epsilon$
$P = \mathbf{a}S\mathbf{u} \mid \mathbf{u}S\mathbf{a} \mid \mathbf{g}S\mathbf{c} \mid \mathbf{c}S\mathbf{g} \mid \mathbf{g}S\mathbf{u} \mid \mathbf{u}S\mathbf{g}$
$B = \mathbf{a} \mid \mathbf{u} \mid \mathbf{g} \mid \mathbf{c}$

Derivation tree = implicit (and decomposed) secondary structure

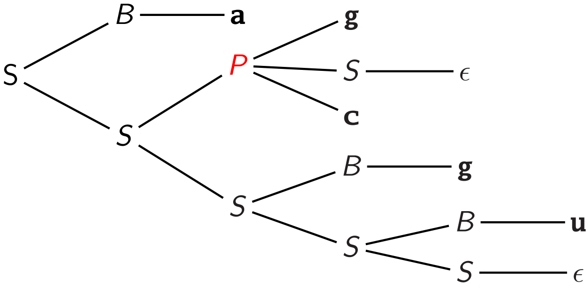


Deriving (and decomposing) sec. structures with grammars

Context-free grammars for primary structures

$S = BS \mid PS \mid \epsilon$
$P = \mathbf{a}S\mathbf{u} \mid \mathbf{u}S\mathbf{a} \mid \mathbf{g}S\mathbf{c} \mid \mathbf{c}S\mathbf{g} \mid \mathbf{g}S\mathbf{u} \mid \mathbf{u}S\mathbf{g}$
$B = \mathbf{a} \mid \mathbf{u} \mid \mathbf{g} \mid \mathbf{c}$

Derivation tree = implicit (and decomposed) secondary structure

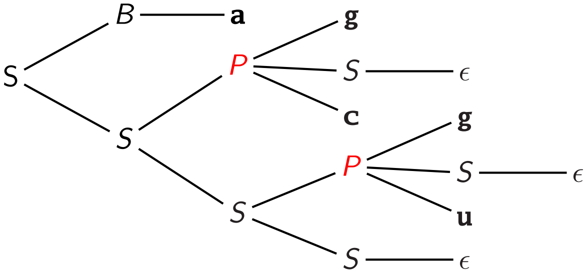


Deriving (and decomposing) sec. structures with grammars

Context-free grammars for primary structures

$S = BS \mid PS \mid \epsilon$
$P = \mathbf{a}S\mathbf{u} \mid \mathbf{u}S\mathbf{a} \mid \mathbf{g}S\mathbf{c} \mid \mathbf{c}S\mathbf{g} \mid \mathbf{g}S\mathbf{u} \mid \mathbf{u}S\mathbf{g}$
$B = \mathbf{a} \mid \mathbf{u} \mid \mathbf{g} \mid \mathbf{c}$

Derivation tree = implicit (and decomposed) secondary structure



Deriving (and decomposing) sec. structures with grammars

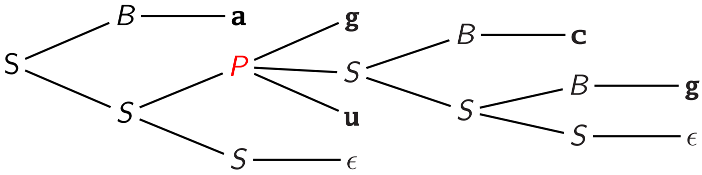
Context-free grammars for primary structures

$S = BS \mid PS \mid \epsilon$

$P = \mathbf{a}S\mathbf{u} \mid \mathbf{u}S\mathbf{a} \mid \mathbf{g}S\mathbf{c} \mid \mathbf{c}S\mathbf{g} \mid \mathbf{g}S\mathbf{u} \mid \mathbf{u}S\mathbf{g}$

$B = \mathbf{a} \mid \mathbf{u} \mid \mathbf{g} \mid \mathbf{c}$

Derivation tree = implicit (and decomposed) secondary structure

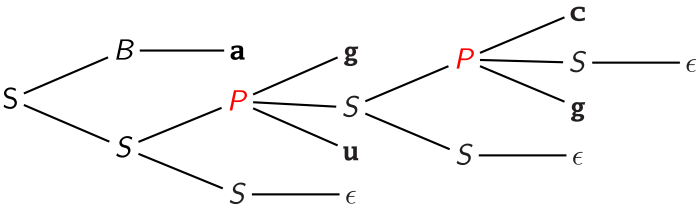


Deriving (and decomposing) sec. structures with grammars

Context-free grammars for primary structures

$S = BS \mid PS \mid \epsilon$
$P = \mathbf{a}S\mathbf{u} \mid \mathbf{u}S\mathbf{a} \mid \mathbf{g}S\mathbf{c} \mid \mathbf{c}S\mathbf{g} \mid \mathbf{g}S\mathbf{u} \mid \mathbf{u}S\mathbf{g}$
$B = \mathbf{a} \mid \mathbf{u} \mid \mathbf{g} \mid \mathbf{c}$

Derivation tree = implicit (and decomposed) secondary structure

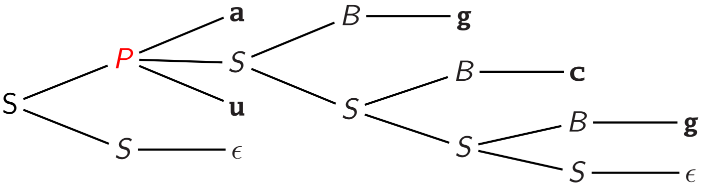


Deriving (and decomposing) sec. structures with grammars

Context-free grammars for primary structures

$S = BS \mid PS \mid \epsilon$
$P = \mathbf{a}S\mathbf{u} \mid \mathbf{u}S\mathbf{a} \mid \mathbf{g}S\mathbf{c} \mid \mathbf{c}S\mathbf{g} \mid \mathbf{g}S\mathbf{u} \mid \mathbf{u}S\mathbf{g}$
$B = \mathbf{a} \mid \mathbf{u} \mid \mathbf{g} \mid \mathbf{c}$

Derivation tree = implicit (and decomposed) secondary structure

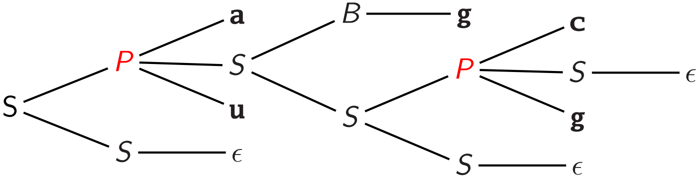


Deriving (and decomposing) sec. structures with grammars

Context-free grammars for primary structures

$S = BS \mid PS \mid \epsilon$
$P = \mathbf{a}S\mathbf{u} \mid \mathbf{u}S\mathbf{a} \mid \mathbf{g}S\mathbf{c} \mid \mathbf{c}S\mathbf{g} \mid \mathbf{g}S\mathbf{u} \mid \mathbf{u}S\mathbf{g}$
$B = \mathbf{a} \mid \mathbf{u} \mid \mathbf{g} \mid \mathbf{c}$

Derivation tree = implicit (and decomposed) secondary structure

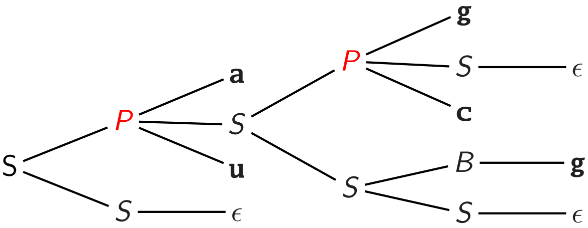


Deriving (and decomposing) sec. structures with grammars

Context-free grammars for primary structures

$S = BS \mid PS \mid \epsilon$
$P = \mathbf{a}S\mathbf{u} \mid \mathbf{u}S\mathbf{a} \mid \mathbf{g}S\mathbf{c} \mid \mathbf{c}S\mathbf{g} \mid \mathbf{g}S\mathbf{u} \mid \mathbf{u}S\mathbf{g}$
$B = \mathbf{a} \mid \mathbf{u} \mid \mathbf{g} \mid \mathbf{c}$

Derivation tree = implicit (and decomposed) secondary structure



Dynamic Programming (Cocke–Younger–Kasami, '60s)

- Nonterminal N represented by 2-dimensional table
- $N_{i,j}$ represents optimal value of all derivation trees with N as root for subword $[i,j]$
- Tables computed in cubic time

Dynamic Programming (Cocke–Younger–Kasami, '60s)

- Nonterminal N represented by 2-dimensional table
- $N_{i,j}$ represents optimal value of all derivation trees with N as root for subword $[i,j]$
- Tables computed in cubic time

Recurrences for optimizing base pairs

(grammar: $S = \texttt{.}\ S \mid \texttt{(}S\texttt{)}S \mid \epsilon$)

$$
\begin{aligned}
S_{i,j} = \max(& \\
& \{S_{i+1,j} \mid j-i \geq 1 \wedge w_i \in \{\text{`g'}, \text{`c'}, \text{`a'}, \text{`u'}\}\} \ \cup \\
& \{1 + S_{i+1,k-1} + S_{k,j} \mid j-i \geq 2 \wedge i+1 \leq k \leq j \wedge \dots\} \ \cup \\
& \{0 \mid j-i = 0\} \\
)&
\end{aligned}
$$

Software for CYK parsing / evaluation

Fixed set of grammars

- RNAfold
- Mfold
- UNAFold
- RNAstructure
- Pknots
- ...

explicit DP
=
arrays, indices, recurrences
=
many lines of C code

Software for CYK parsing / evaluation

Fixed set of grammars

- RNAfold
- Mfold
- UNAFold
- RNAstructure
- Pknots
- …

Arbitrary grammars

- ADP
- ADPfusion
- ADPC
- GAPC

explicit DP
=
arrays, indices, recurrences
=
many lines of C code

implicit DP
=
grammars, algebras
=
concise high-level code

Algebraic Dynamic Programming (Giegerich, 2000)

```
s = (\ b s -> s)     <<< b -~~ s |||
    (\ p s -> 1+p+s) <<< p +~~ s |||
    (\ _   -> 0)     <<< empty
    ... max

p = (\ _ s _ -> s) <<< char 'a' -~~ s ~~- char 'u' |||
    (\ _ s _ -> s) <<< char 'u' -~~ s ~~- char 'a' |||
    ⋮

b = (\ _ -> 0) <<< char 'a' |||
    (\ _ -> 0) <<< char 'u' |||
    ⋮
```

- Grammar with evaluation functions (= algebra)
- Haskell program
- `s`, `p`, `b` compute their dynamic programming table

Multiple context-free grammars (Seki et al., 1991)

(for RNA secondary structures: (Kato et al., 2005))

Pseudoknots = crossings = not context-free

$L = \{a_1^i b_1^j a_2^i b_2^j \mid i, j \geq 0\}$
e.g. **((([[)))]]**

Multiple context-free grammars (Seki et al., 1991)

(for RNA secondary structures: (Kato et al., 2005))

Pseudoknots = crossings = not context-free

$L = \{a_1^i b_1^j a_2^i b_2^j \mid i, j \geq 0\}$
e.g. **((([[)))]]**

Nonterminals as vectors (or: introducing limited context-sensitivity)

$$S = \texttt{.}\ S \mid \texttt{(}S\texttt{)}S \mid \epsilon \mid M_1 S N_1 S M_2 S N_2 S$$

$$\begin{pmatrix} M_1 \\ M_2 \end{pmatrix} = \begin{pmatrix} M_1 \texttt{(} \\ \texttt{)} M_2 \end{pmatrix} \Bigg| \begin{pmatrix} \texttt{(} \\ \texttt{)} \end{pmatrix}$$

$$\begin{pmatrix} N_1 \\ N_2 \end{pmatrix} = \begin{pmatrix} N_1 \texttt{[} \\ \texttt{]} N_2 \end{pmatrix} \Bigg| \begin{pmatrix} \texttt{[} \\ \texttt{]} \end{pmatrix}$$

(inlined style (Wild, 2010))

Multiple context-free grammars, continued

Input word

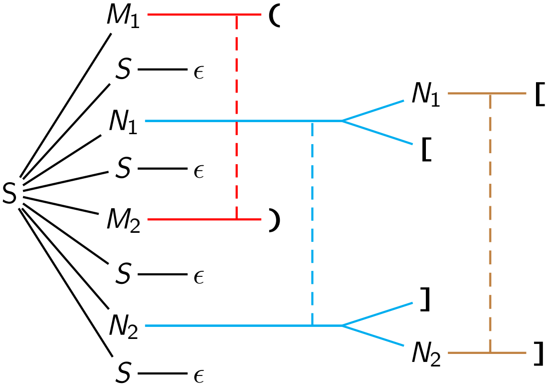
([[)]]

Multiple context-free grammars, continued

Input word

`([[)]]`

Derivation tree



Dynamic programming for multiple context-free grammars

- Nonterminal N with dimension $d \geq 1$ represented by $2d$-dimensional table
- For $d = 1$, $N_{i,j}$ represents optimal value of all derivation trees with N as root for subword $[i, j]$

Dynamic programming for multiple context-free grammars

- Nonterminal N with dimension $d \geq 1$ represented by $2d$-dimensional table
- For $d = 1$, $N_{i,j}$ represents optimal value of all derivation trees with N as root for subword $[i, j]$
- For $d = 2$, $N_{i,j,k,l}$ represents optimal value of all derivation trees with N as root for subwords $[i, j]$ and $[k, l]$

Dynamic programming for multiple context-free grammars

- Nonterminal N with dimension $d \geq 1$ represented by $2d$-dimensional table
- For $d = 1$, $N_{i,j}$ represents optimal value of all derivation trees with N as root for subword $[i, j]$
- For $d = 2$, $N_{i,j,k,l}$ represents optimal value of all derivation trees with N as root for subwords $[i, j]$ and $[k, l]$
- For $d = 3$, $N_{i,j,k,l,m,n}$ ...

Dynamic programming for multiple context-free grammars

- Nonterminal N with dimension $d \geq 1$ represented by $2d$-dimensional table
- For $d = 1$, $N_{i,j}$ represents optimal value of all derivation trees with N as root for subword $[i, j]$
- For $d = 2$, $N_{i,j,k,l}$ represents optimal value of all derivation trees with N as root for subwords $[i, j]$ and $[k, l]$
- For $d = 3$, $N_{i,j,k,l,m,n}$ ...

1. Writing explicit DP code without errors is hard.
2. It becomes even harder for $\geq$ 4-dimensional tables.
3. Don't do it over and over from scratch!

Dynamic programming for multiple context-free grammars

- Nonterminal N with dimension $d \geq 1$ represented by $2d$-dimensional table
- For $d = 1$, $N_{i,j}$ represents optimal value of all derivation trees with N as root for subword $[i, j]$
- For $d = 2$, $N_{i,j,k,l}$ represents optimal value of all derivation trees with N as root for subwords $[i, j]$ and $[k, l]$
- For $d = 3$, $N_{i,j,k,l,m,n}$ ...

1. Writing explicit DP code without errors is hard.
2. It becomes even harder for $\geq$ 4-dimensional tables.
3. Don't do it over and over from scratch!

Solution: **adp-multi** (myself, 2012)

adp-multi: extending ADP with MCFGs

```
rewritePair [p1,p2,s1,s2] = [p1,s1,p2,s2]
rewriteKnot [k11,k12,k21,k22,s1,s2,s3,s4]
          = [k11,s1,k21,s2,k12,s3,k22,s4]

s = nil  <<< EPS                                  >>>| id |||
    left <<< b ~~~| s                             >>>| id |||
    pair <<< p ~~~| s ~~~| s                      >>>| rewritePair |||
    knot <<< k ~~~  k ~~~| s ~~~| s ~~~| s ~~~| s >>>| rewriteKnot
    ... h

b = base <<< ’a’ >>>| id |||
    base <<< ’u’ >>>| id |||
    ⋮

p = basepair <<< (’a’,’u’) >>>|| id2 |||
    basepair <<< (’u’,’a’) >>>|| id2 |||
    ⋮

rewriteKnot1 [p1,p2,k1,k2] = ([k1,p1],[p2,k2])

k = knot1 <<< p ~~~|| k >>>|| rewriteKnot1 |||
    knot2 <<< p         >>>|| id2
```

```
nil _      = 0
left _ b   = b
pair _ b c = b + c
knot _ _ c d e f
 = 1 + c + d + e + f
knot1 _ _  = 0
knot2 _    = 0
basepair _ = 0
base _     = 0
h          = max
```

adp-multi: extending ADP with MCFGs

What is it?

- Extension of ADP method with multiple context-free grammars
- Prototype in Haskell based on original ADP implementation

Prototype means...

- High constant runtime factor (as original ADP prototype)
- Only 1+2-dimensional nonterminals so far (but easily extensible)
- Useful for experimentation (includes example grammars)

Future work

- Support all dimensions
- Integrate into ADPfusion

Thanks and acknowledgments

- Johannes Waldmann
- Peter F. Stadler
- Christian Höner zu Siederdissen
- #haskell @ irc://chat.freenode.net

More info and source code

`http://adp-multi.ruhoh.com`