# A Factor-Graph Approach to the Context-Tree Weighting Method

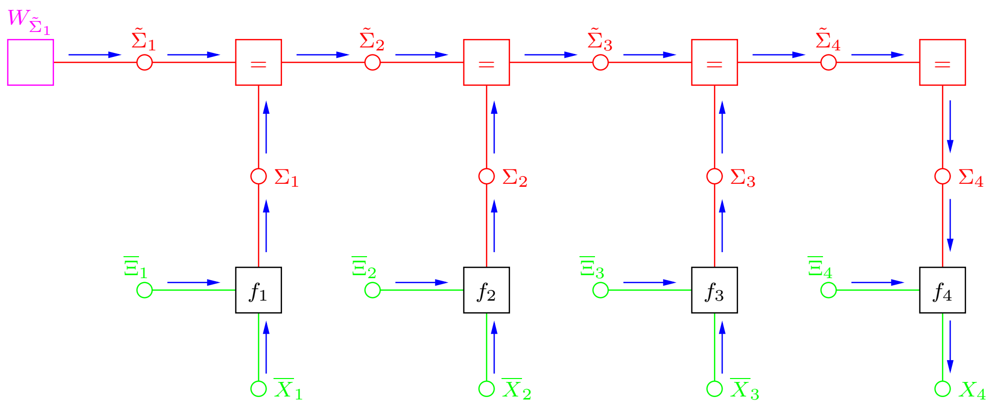
Pascal O. Vontobel
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

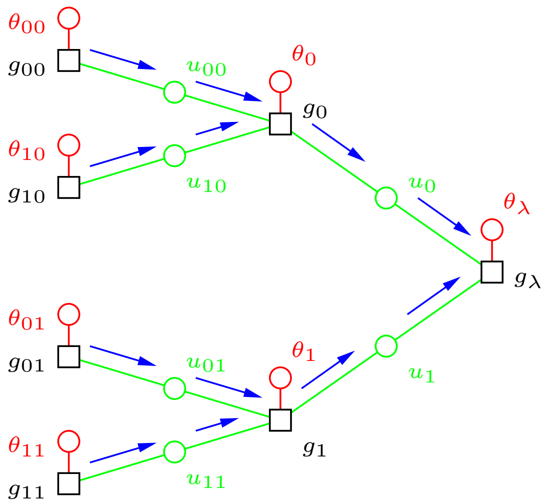
Talk at U of U, Salt Lake City, UT, USA March 22, 2004

# Table of Contents

# Preview

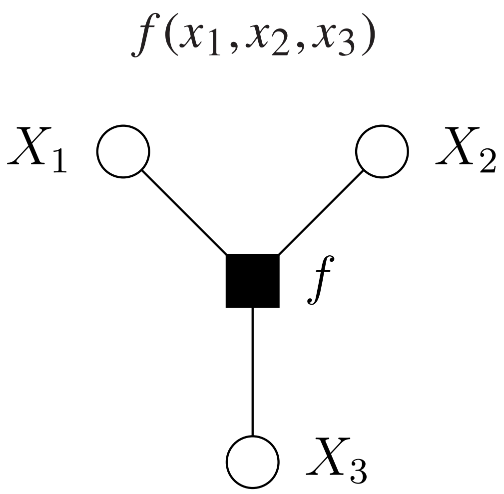
- Two different factor graphs will play the key roles in this talk.
- The sum-product algorithm will deliver the desired quantities.



## **Factor Graphs** (Part 1)

By a factor graph we can represent a multivariate function:




- For each variable we draw a variable node (empty circles) and for each function we draw a function node (filled squares).
- There is an edge between a variable node and a function node if the corresponding variable is an argument of the corresponding function.
- Therefore, a factor graph is a bipartite graph: edges are only allowed between vertices of different types.

General references for factor graphs are:

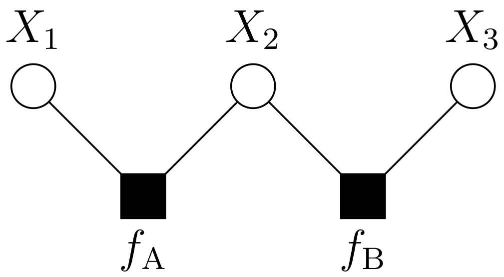

- F. R. Kschischang, B. J. Frey and H.-A. Loeliger, "Factor graphs and the sum-product algorithm", IEEE Trans. on Inform. Theory, IT–47, Feb. 2001.
- H.-A. Loeliger, "An introduction to factor graphs", IEEE Signal Processing Magazine, Jan. 2004.

## **Factor Graphs** (Part 2)

We assume, that we know more about the internal structure of the function $f(x_1, x_2, x_3)$, e.g.

$$f(x_1, x_2, x_3) = f_A(x_1, x_2) \cdot f_B(x_2, x_3).$$

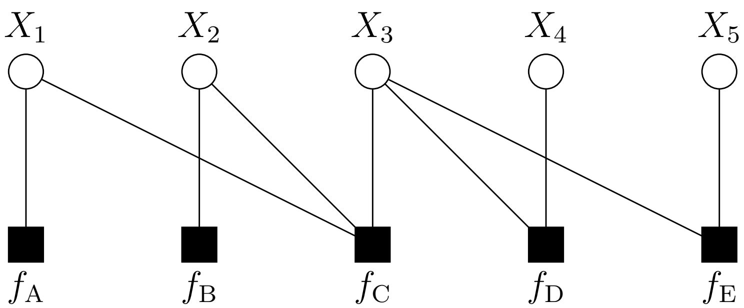
Then we can take advantage of this fact and the factor graph represents this structure.

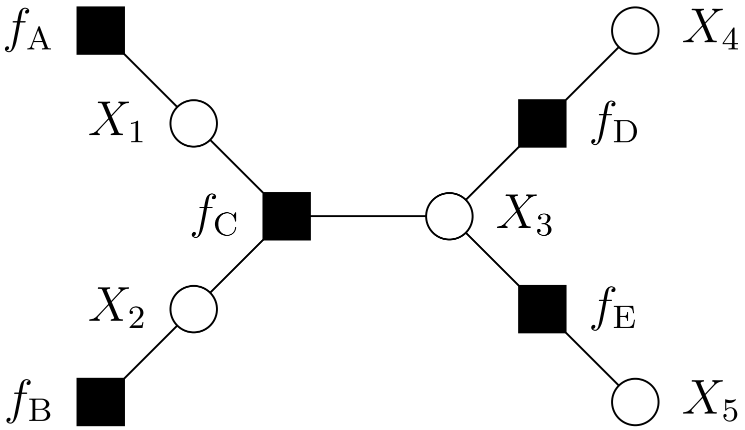



*Note:* We call $f(.,.,.)$ the global function and $f_A(.,.)$ and $f_B(.,.)$ the local functions.

## **Factor Graphs** (Part 3)

$$f(x_1,x_2,x_3,x_4,x_5) = f_A(x_1) \cdot f_B(x_2) \cdot f_C(x_1,x_2,x_3) \cdot f_D(x_3,x_4) \cdot f_E(x_3,x_5)$$

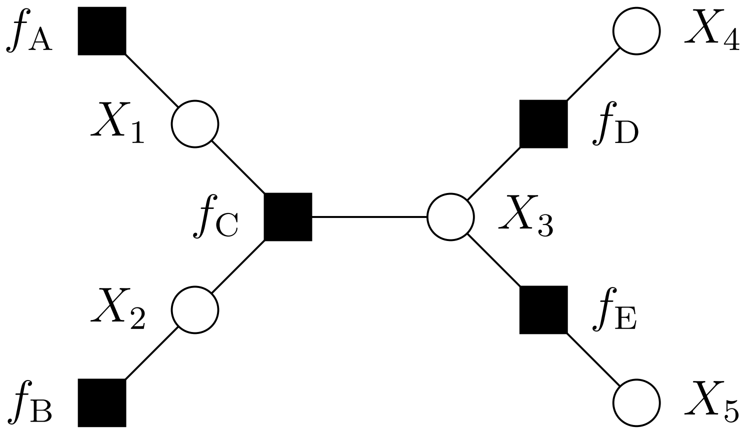



*Note:* One and the same function can be represented by graphs with different structures: some are more pleasing than others.

## **The Sum-Product Algorithm** (Part 1)

Let us consider again the following factor graph (which is a tree).




The global function is

$$f(x_1,x_2,x_3,x_4,x_5) = f_A(x_1)\cdot f_B(x_2)\cdot f_C(x_1,x_2,x_3)\cdot f_D(x_3,x_4)\cdot f_E(x_3,x_5).$$

# The Sum-Product Algorithm (Part 2)

The global function is

$$f(x_1,x_2,x_3,x_4,x_5) = f_{\mathrm{A}}(x_1)\cdot f_{\mathrm{B}}(x_2)\cdot f_{\mathrm{C}}(x_1,x_2,x_3)\cdot f_{\mathrm{D}}(x_3,x_4)\cdot f_{\mathrm{E}}(x_3,x_5)$$

We would like to calculate the marginals

$$\begin{aligned}\tilde{f}_1(x_1) &= \sum_{x_2,x_3,x_4,x_5} f(x_1,x_2,x_3,x_4,x_5)\\ &= \sum_{x_2,x_3,x_4,x_5} f_{\mathrm{A}}(x_1)\cdot f_{\mathrm{B}}(x_2)\cdot f_{\mathrm{C}}(x_1,x_2,x_3)\cdot f_{\mathrm{D}}(x_3,x_4)\cdot f_{\mathrm{E}}(x_3,x_5)\end{aligned}$$

or

$$\begin{aligned}\tilde{f}_3(x_3) &= \sum_{x_1,x_2,x_4,x_5} f(x_1,x_2,x_3,x_4,x_5)\\ &= \sum_{x_1,x_2,x_4,x_5} f_{\mathrm{A}}(x_1)\cdot f_{\mathrm{B}}(x_2)\cdot f_{\mathrm{C}}(x_1,x_2,x_3)\cdot f_{\mathrm{D}}(x_3,x_4)\cdot f_{\mathrm{E}}(x_3,x_5),\end{aligned}$$

etc.

# **The Sum-Product Algorithm** (Part 3)

$$\tilde{f}_1(x_1) = \sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} f_{\mathrm{A}}(x_1)\cdot f_{\mathrm{B}}(x_2)\cdot f_{\mathrm{C}}(x_1,x_2,x_3)\cdot f_{\mathrm{D}}(x_3,x_4)\cdot f_{\mathrm{E}}(x_3,x_5)$$

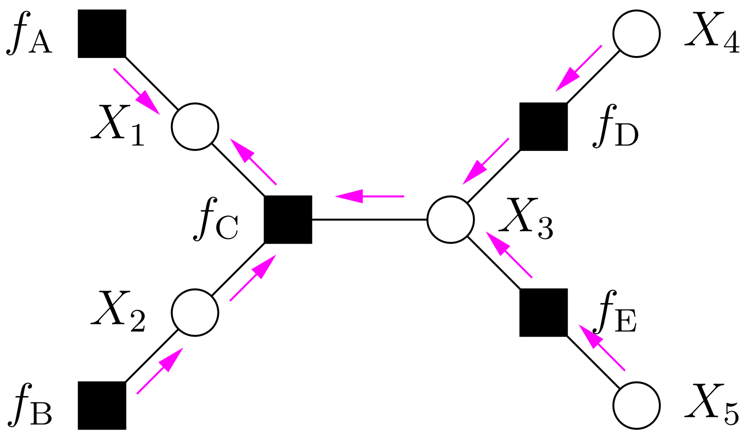
$$= \underbrace{f_{\mathrm{A}}(x_1)}_{\mu_{f_{\mathrm{A}}\to X_1}(x_1)} \underbrace{\sum_{x_2}\sum_{x_3} f_{\mathrm{C}}(x_1,x_2,x_3) \underbrace{\underbrace{f_{\mathrm{B}}(x_2)}_{\mu_{f_{\mathrm{B}}\to X_2}(x_2)}}_{\mu_{X_2\to f_{\mathrm{C}}}(x_2)} \underbrace{\underbrace{\sum_{x_4} f_{\mathrm{D}}(x_3,x_4)\cdot \underbrace{1}_{\mu_{X_4\to f_{\mathrm{D}}}(x_4)}}_{\mu_{f_{\mathrm{D}}\to X_3}(x_3)} \underbrace{\sum_{x_5} f_{\mathrm{E}}(x_3,x_5)\cdot \underbrace{1}_{\mu_{X_5\to f_{\mathrm{E}}}(x_5)}}_{\mu_{f_{\mathrm{E}}\to X_3}(x_3)}}_{\mu_{f_{X_3\to \mathrm{C}}}(x_3)}}_{\mu_{f_{\mathrm{C}}\to X_1}(x_1)}$$

Similar manipulations can be performed for calculating $\tilde{f}_2(x_2)$, $\tilde{f}_3(x_3)$, $\tilde{f}_4(x_4)$, $\tilde{f}_5(x_5)$.

The objects $\mu_{X_i\to f_j}(x_i)$ or $\mu_{f_j\to X_i}(x_i)$, which can be associated with the edge between the vertices $X_i$ and $f_j$, are called messages and are functions of $x_i$. The domain of these functions correspond to the alphabet of $X_i$.

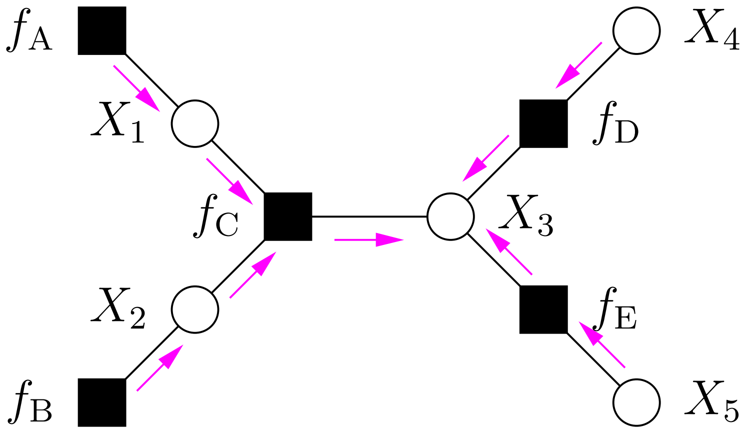
## The Sum-Product Algorithm (Part 4)

Messages necessary for calculating $\tilde{f}_1(x_1)$.



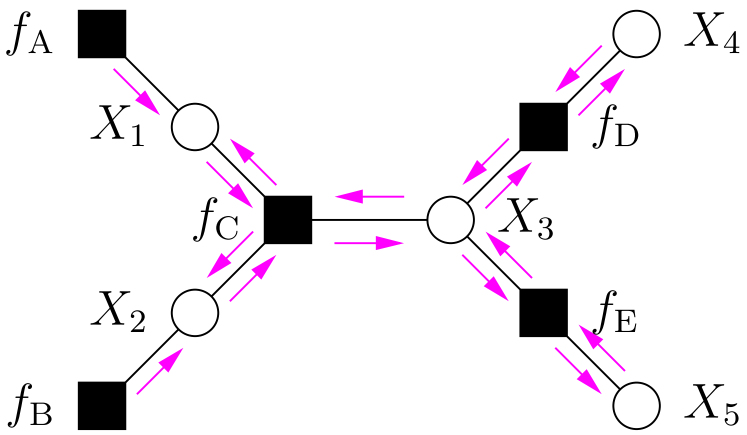
## **The Sum-Product Algorithm** (Part 5)

Messages necessary for calculating $\tilde{f}_3(x_3)$.



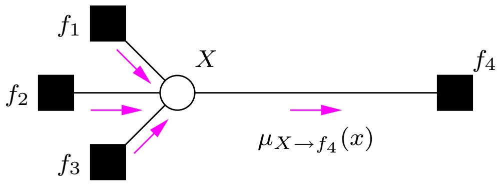
## The Sum-Product Algorithm (Part 6)

Messages necessary for calculating $\tilde{f}_1(x_1)$, $\tilde{f}_2(x_2)$, $\tilde{f}_3(x_3)$, $\tilde{f}_4(x_4)$, and $\tilde{f}_5(x_5)$.

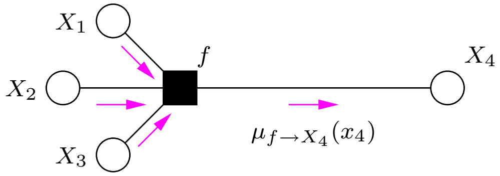



- Messages are sent along edges and processing (taking products and doing summations) is done in the vertices.
- We see that messages can be "reused" in the sense that many partial calculations are the same; so it suffices to perform them only once.

## The Sum-Product Algorithm (Part 7)




$$\mu_{X\to f_4}(x) = \mu_{f_1\to X}(x)\cdot\mu_{f_2\to X}(x)\cdot\mu_{f_3\to X}(x)$$

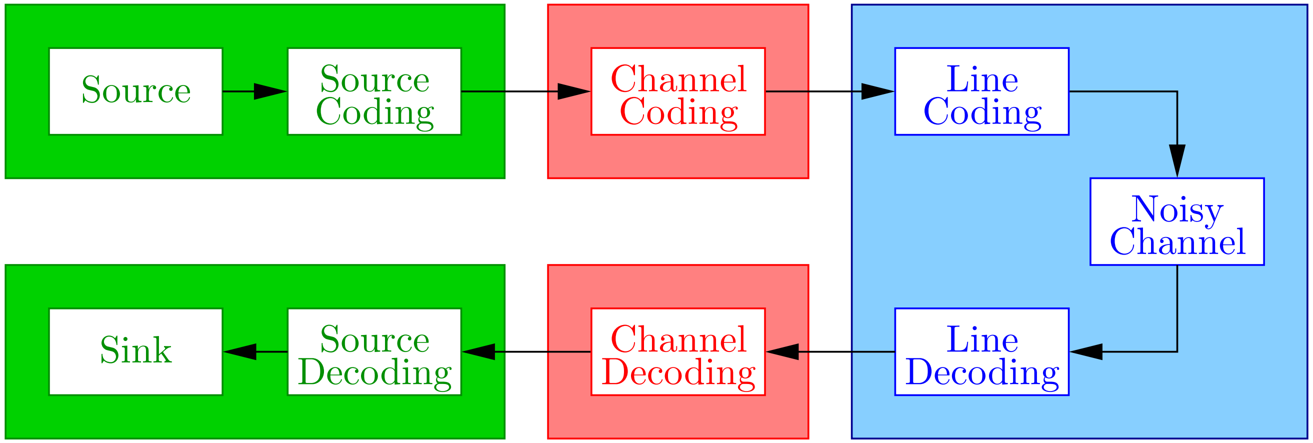



$$\mu_{f\to x_4}(x_4) = \sum_{x_1}\sum_{x_2}\sum_{x_3} f(x_1,x_2,x_3,x_4)\cdot\mu_{X_1\to f}(x_1)\cdot\mu_{X_2\to f}(x_2)\cdot\mu_{X_3\to f}(x_3)$$

If the factor graph has no loops then it is obvious what messages have to be calculated when.

If the factor graphs has loops one has to decide what update schedule to take. (This second point is not of relevance for this talk.)

# Source Coding

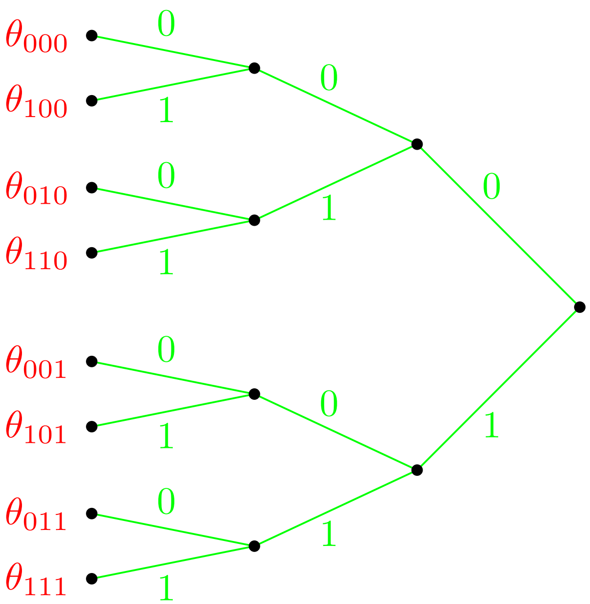



This talk focuses on the source coding and decoding part of a communication system. There are two main categories of source coding:

- lossless,
- lossy.

This talk will be about lossless source coding. In the following, we assume that thanks to channel coding/decoding we have an ideal channel. For simplicity, we will assume to have a binary source.

# Markov Source or Order $m$




Context tree of a Markov source of order $m = 3$ with parameters $\boldsymbol{\theta}$.

A Markov source of order $m$ with parameters $\boldsymbol{\theta}$ produces sequences with pmf

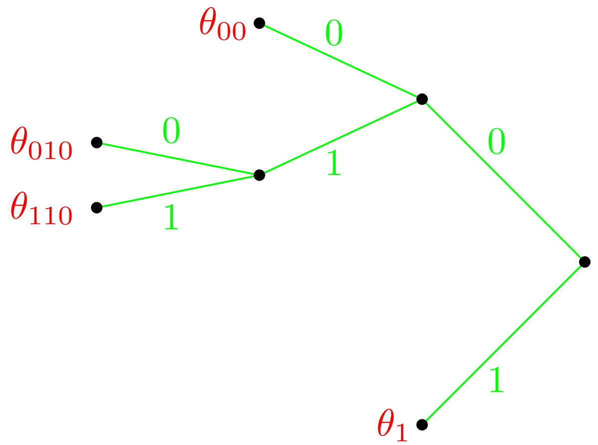
$$P(\mathbf{x}_1^n | \mathbf{x}_{1-m}^0) = \prod_{k=1}^{n} P(x_k | \mathbf{x}_{1-m}^{k-1}) = \prod_{k=1}^{n} P(x_k | \mathbf{x}_{k-m}^{k-1}),$$

where

$$P(x_k | \mathbf{x}_{k-m}^{k-1}) \triangleq \begin{cases} 1 - \theta_{\mathbf{x}_{k-m}^{k-1}} & \text{(for } x_k = 0\text{)} \\ \theta_{\mathbf{x}_{k-m}^{k-1}} & \text{(for } x_k = 1\text{)} \end{cases}.$$

For notational simplicity, we will omit the conditioning on $\mathbf{x}_{1-m}^{-1}$ in the following.

# Tree Source




Context tree $t$ of a tree source with parameters $\boldsymbol{\theta}$.

A tree source with context tree $t$ and parameters $\boldsymbol{\theta}$ produces sequences with pmf

$$P(\mathbf{x}_1^n | \mathbf{x}_{1-m}^0) = \prod_{k=1}^{n} P(x_k | \mathbf{x}_{1-m}^{k-1})$$

$$= \prod_{k=1}^{n} P(x_k | s(\mathbf{x}_{1-m}^{k-1})),$$

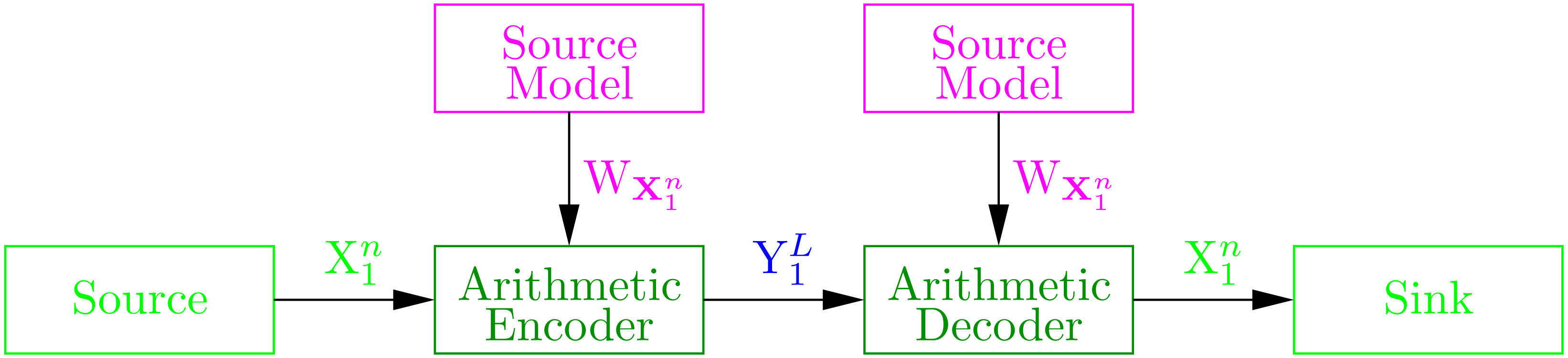
where

$$P(x_k | s(\mathbf{x}_{1-m}^{k-1})) \triangleq \begin{cases} 1 - \theta_{s(\mathbf{x}_{1-m}^{k-1})} & (\text{for } x_k = 0) \\ \theta_{s(\mathbf{x}_{1-m}^{k-1})} & (\text{for } x_k = 1) \end{cases}.$$

$s(\mathbf{x}_{1-m}^{k-1})$ gives the relevant context (suffix) in the context tree $t$.

For notational simplicity, we will omit the conditioning on $\mathbf{x}_{1-m}^{-1}$ in the following.

## Arithmetic Coding (Part 1)




We assume that we can use an arithmetic encoder/decoder (see e.g. Cover & Thomas).

In order to use the arithmetic encoder/decoder, we need to have a source model, i.e. a "device" that assigns some weight $W(\mathbf{x}_1^n)$ to every possible sequence $\mathbf{x}_1^n$ such that

- $0 \leq W(\mathbf{x}_1^n) \leq 1$ and
- $\sum_{\mathbf{x}_1^n} W(\mathbf{x}_1^n) = 1$,

## Arithmetic Coding (Part 2)

Writing $W(\overline{\mathbf{x}}_1^n) = \prod_{k=1}^{n} W(x_k|\mathbf{x}_1^{k-1})$, what the arithmetic encoder / decoder effectively needs, is the following.

At time step $k$, an arithmetic encoder/decoder needs to know

$$W_{\mathbf{X}_k|X_1^{k-1}}(0|\overline{\mathbf{x}}_1^{k-1}) \quad \text{and} \quad W_{\mathbf{X}_k|X_1^{k-1}}(1|\overline{\mathbf{x}}_1^{k-1}),$$

where $\overline{\mathbf{x}}_1^{k-1}$ is the observed output sequence up to time step $k-1$.

Using arithmetic coding, one can guarantee the following upper bound on the length of the output sequence:

$$L(\overline{\mathbf{x}}_1^n) < \log \frac{1}{W(\overline{\mathbf{x}}_1^n)} + \underbrace{2}_{\substack{\text{coding}\\\text{redundancy}}} .$$

## Source Coding (Case 1: Known Source)

The simplest case is when the source encoder and decoder know the source.

We choose the weights to be

$$W(\mathbf{x}_1^n) \triangleq P(\mathbf{x}_1^n) \quad \text{(for each } \mathbf{x}_1^n\text{)}.$$

Assuming a stationary source, this choice leads to the following asymptotic behavior:

$$\lim_{n \to \infty} \frac{L(\overline{\mathbf{x}}_1^n)}{n} < H(X) + \varepsilon \quad \text{w.p. } 1$$

for any $\varepsilon > 0$, where $H(X)$ is the entropy rate of the source.

## Source Coding (Case 2: Unknown Source) (P. 1)

In universal source coding, we assume to have a set $\Sigma$ of sources.

- $\sigma \in \Sigma$ is chosen with some (unknown) probability $P(\sigma)$.
- $\sigma \in \Sigma$ produces a string $\mathbf{x}_1^n$ with (known) probability $P(\mathbf{x}_1^n | \sigma)$.

Note that we observe $\mathbf{x}_1^n$ but not $\sigma$.

## Source Coding (Case 2: Unknown Source) (P. 2)

How can we choose the weights $W(\mathbf{x}_1^n)$ for our source model?

- If we knew $\sigma$, we would of course take $W(\mathbf{x}_1^n) \triangleq P(\mathbf{x}_1^n|\sigma)$.
- But we do not know $\sigma$, therefore let us take the following approach:
  - We choose $W(\sigma) > 0$ s.t. $\sum_\sigma W(\sigma) = 1$ or $\int W(\sigma)\, \mathrm{d}\sigma = 1$.
  - We choose $W(\mathbf{x}_1^n|\sigma) \triangleq P(\mathbf{x}_1^n|\sigma)$.
  - We choose $W(\mathbf{x}_1^n, \sigma) \triangleq W(\mathbf{x}_1^n|\sigma)W(\sigma)$.
  - We choose $W(\mathbf{x}_1^n) \triangleq \sum_\sigma W(\mathbf{x}_1^n, \sigma) = \sum_\sigma W(\mathbf{x}_1^n|\sigma)W(\sigma)$.

  I.e. we average over all source models in the set $\Sigma$ in order to obtain some $W(\mathbf{x}_1^n)$.

## Source Coding (Case 2: Unknown Source) (P. 3)

How well does this choice for $W(\mathbf{x}_1^n)$ work? For simplicity, let us assume that $\Sigma$ is a finite set. We consider the following lower bound on $W_{\mathbf{X}_1^n}(\overline{\mathbf{x}}_1^n)$:

$$W_{\mathbf{X}_1^n}(\overline{\mathbf{x}}_1^n) = \sum_{\sigma} W_{\mathbf{X}_1^n|\Sigma}(\overline{\mathbf{x}}_1^n|\sigma) W_{\Sigma}(\sigma) \geq \max_{\sigma} W_{\mathbf{X}_1^n|\Sigma}(\overline{\mathbf{x}}_1^n|\sigma) W(\sigma),$$

This leads to the following upper bound:

$$L(\overline{\mathbf{x}}_1^n) < \min_{\sigma} \left[ \log \frac{1}{P_{\mathbf{X}_1^n|\Sigma}(\overline{\mathbf{x}}_1^n|\sigma)} + \underbrace{\log \frac{1}{W(\sigma)}}_{\text{model/parameter redundancy}} \right] + \underbrace{2}_{\text{coding redundancy}} .$$

Under the assumption of stationary sources we obtain

$$\lim_{n\to\infty} \frac{L(\overline{\mathbf{x}}_1^n)}{n} < H(X|\sigma) + \varepsilon \quad \text{w.p. } 1,$$

for any $\varepsilon > 0$ and where $H(X|\sigma)$ is the entropy rate of the actual source model $\sigma$.

## Source Coding (Case 2: Unknown Source) (P. 4)

Question: how do we compute

$$W_{\mathbf{X}_k|X_1^{k-1}}(0|\bar{\mathbf{x}}_1^{k-1}) \quad \text{and} \quad W_{\mathbf{X}_k|X_1^{k-1}}(1|\bar{\mathbf{x}}_1^{k-1}),$$

efficiently?

Remember, we defined $W(\mathbf{x}_1^n)$ to be

$$W(\mathbf{x}_1^n) \triangleq \sum_{\sigma} W(\mathbf{x}_1^n, \sigma) = \sum_{\sigma} W(\sigma) W(\mathbf{x}_1^n|\sigma) = \sum_{\sigma} W(\sigma) \prod_{k=1}^{n} W(x_n|\xi_n, \sigma)$$

Here we used

$$\xi_n \triangleq \mathbf{x}_{k-D}^{k-1} = (X_{k-D}, \ldots, X_{k-1}),$$

where $D$ is the maximal depth needed for any source model $\sigma \in \Sigma$.

## Source Coding (Case 2: Unknown Source) (P. 5)

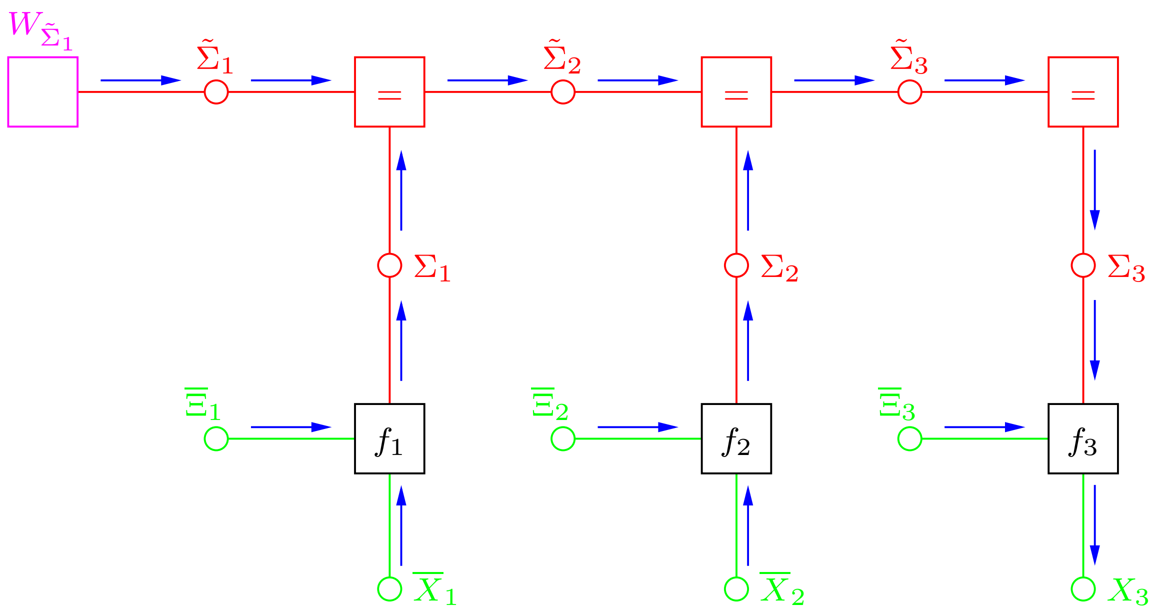
$\Longrightarrow$ Representing the pmf $W(\mathbf{x}_1^n, \sigma)$ by a factor graph, we get the desired quantities by using the sum-product algorithm.

$$W_{\mathbf{X}_k | X_1^{k-1}}(x_k | \overline{\mathbf{x}}_1^{k-1}) \propto W_{\mathbf{X}_k, X_1^{k-1}}(x_k, \overline{\mathbf{x}}_1^{k-1})$$

$$= \underbrace{\underbrace{\sum_{\sigma}}_{\text{Sum}} \underbrace{W(\sigma) W(x_k | \overline{\xi}_k, \sigma) \prod_{\ell=1}^{k-1} W(\overline{x}_\ell | \overline{\xi}_\ell, \sigma)}_{\substack{\text{Factor graph} \\ \text{(written as product)}}}}_{\text{Sum-product algorithm}}$$

# Source Coding (Case 2: Unknown Source) (P. 6)

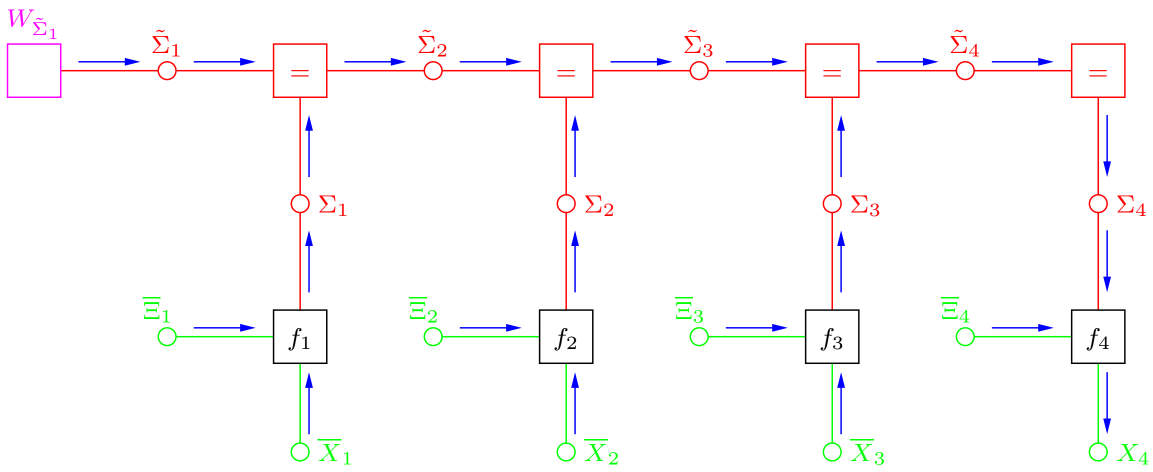
$$W_{\mathbf{X}_k|X_1^{k-1}}(x_k|\overline{\mathbf{x}}_1^{k-1}) \propto W_{\mathbf{X}_k,X_1^{k-1}}(x_k,\overline{\mathbf{x}}_1^{k-1}) = \underbrace{\underbrace{\sum_{\sigma}}_{\text{Sum}} \underbrace{W(\sigma)W(x_k|\overline{\xi}_k,\sigma)\prod_{\ell=1}^{k-1} W(\overline{x}_\ell|\overline{\xi}_\ell,\sigma)}_{\substack{\text{Factor graph}\\ \text{(written as product)}}}}_{\text{Sum-product algorithm}}$$




Shown here for the case $k = 3$.

# Source Coding (Case 2: Unknown Source) (P. 7)

$$W_{\mathbf{X}_k | X_1^{k-1}}(x_k | \bar{\mathbf{x}}_1^{k-1}) \propto W_{\mathbf{X}_k, X_1^{k-1}}(x_k, \bar{\mathbf{x}}_1^{k-1}) = \underbrace{\underbrace{\sum_{\sigma}}_{\text{Sum}} \underbrace{W(\sigma) W(x_k | \bar{\xi}_k, \sigma) \prod_{\ell=1}^{k-1} W(\bar{x}_\ell | \bar{\xi}_\ell, \sigma)}_{\substack{\text{Factor graph} \\ \text{(written as product)}}}}_{\text{Sum-product algorithm}}$$




Shown here for the case $k = 4$. Note that the messages that we used for getting the relevant probabilities for $k = 3$ can be reused.

## Choosing the Set of Sources (Part 1)

Up to now, we just said that the sources comes from some set $\Sigma$, but we have not specified $\Sigma$.

It would be desirable that $\Sigma$ contains all (binary) tree sources up to depth $D$ for some fixed $D \geq 1$. Remember, that a tree source is characterized

- by the context tree $t$,
- by a set of parameters $\boldsymbol{\theta}^{(t)}$ where $0 \leq \theta_s^{(t)} \leq 1$ for all $s \in t$.

Some complexity considerations:

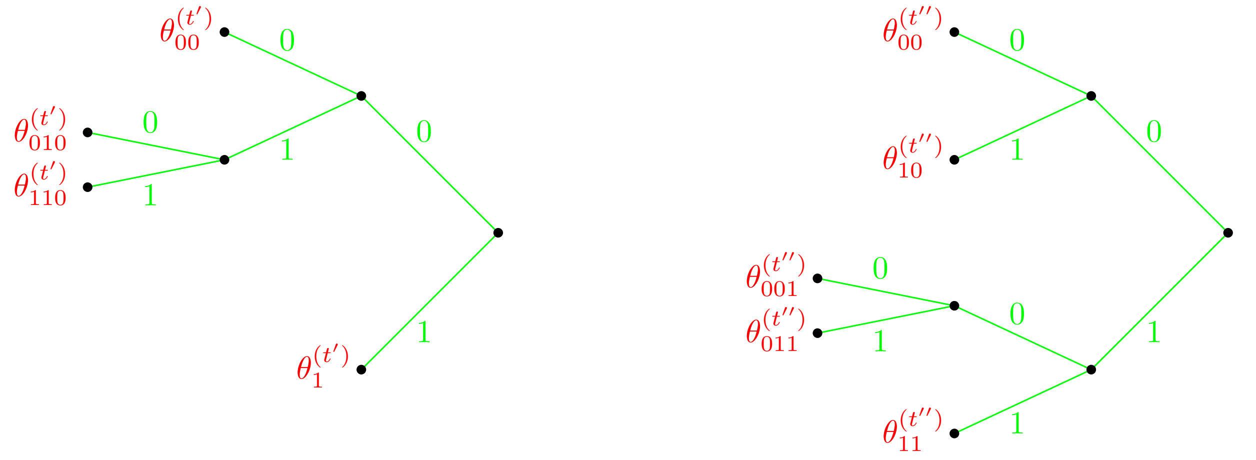
- number of context trees of maximal depth $D$: approx. $2^{2^D}$
- number of parameters per tree source: number of leaves of context tree (which is less than or equal to $2^D$).
- All parameters $\theta_s^{(t)}$ are continuous.

## Choosing the Set of Sources (Part 2)

While performing the sum-product algorithm, we have to sum/integrate over the whole set $\Sigma$. In order to get an efficient algorithm, one has to make careful choices for the weights of the models in $\Sigma$.

Let $\sigma = (t, \boldsymbol{\theta}^{(t)})$. Willems, Shtarkov, and Tjalkens made the following crucial choices:

- Instead of having a parameter vector $\boldsymbol{\theta}^{(t)}$ for each context tree $t$, we will have a parameter vector $\boldsymbol{\theta}$ that contains a $\theta_s$ for each possible suffix $s$ of length at most $D$.

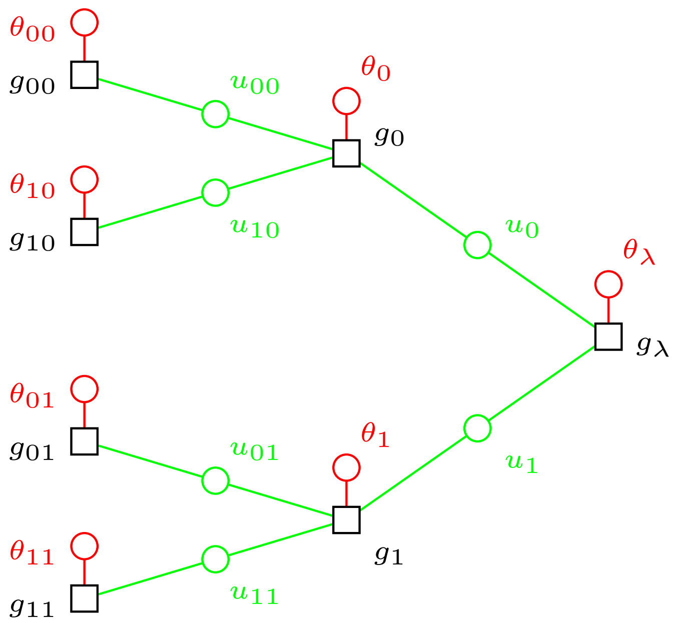


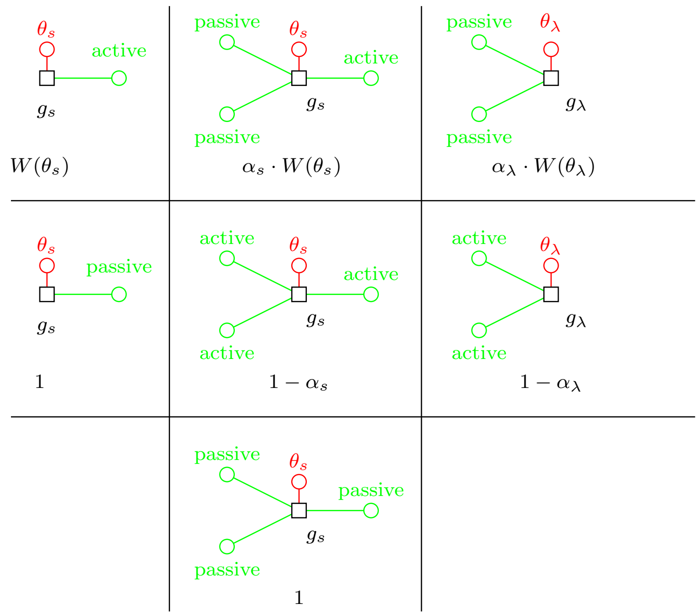

E.g., here we set $\theta_{00} = \theta_{00}^{(t')} = \theta_{00}^{(t'')}$.

# **Choosing the Set of Sources** (Part 3)

Furthermore . . .

- We implicitly define the pmf/pdf $W(\sigma) = W(t, \boldsymbol{\theta})$ by specifying a so-called context-tree factor graph whose global function corresponds to the desired pmf/pdf. (Shown here for $D = 2$.)

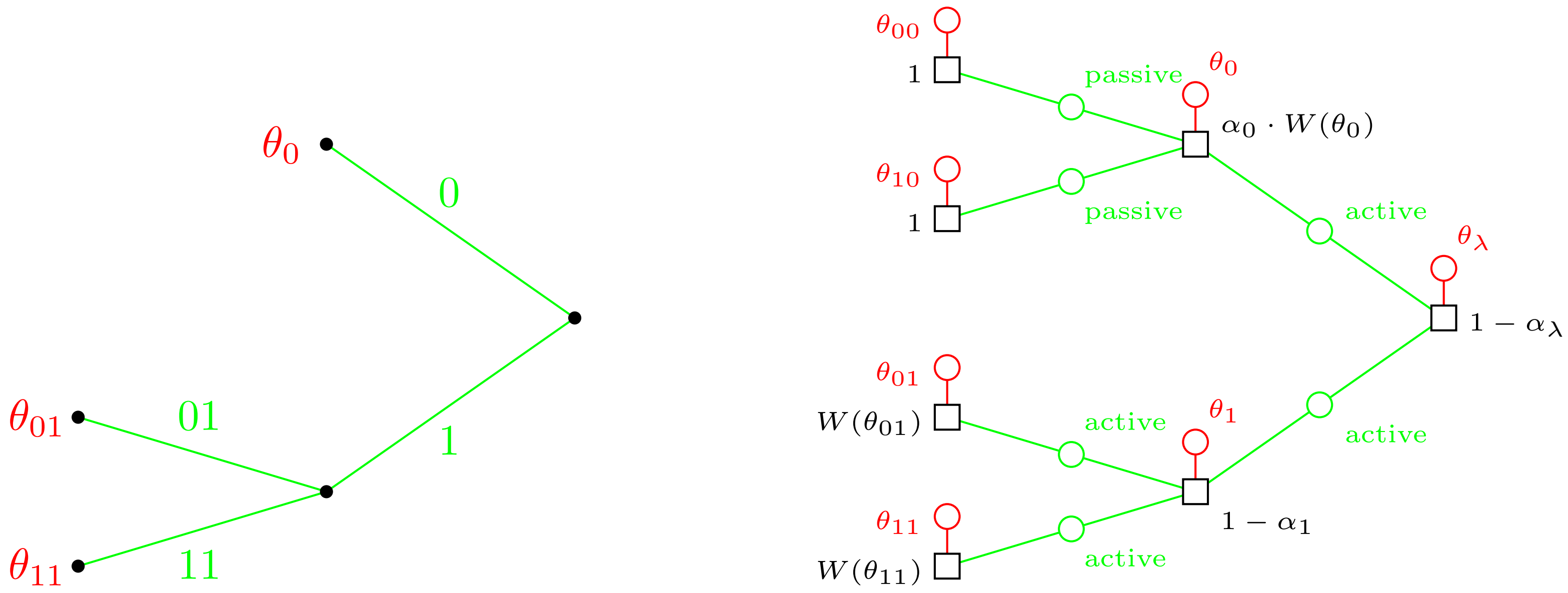



($\lambda$ denotes the empty string, $0 < \alpha_s < 1$, $s \in \mathcal{S}$.)

# Choosing the Set of Sources (Part 4)

To be specific, let us see what weight is assigned to the tree source shown on the left-hand side.




As we see from the configuration in the factor graph on the right-hand side, the assigned weight (value of global function) is

$$W(t,\boldsymbol{\theta}) = \underbrace{(1-\alpha_\lambda)\cdot\alpha_0\cdot(1-\alpha_1)}_{W(t)}\cdot\underbrace{W(\theta_0)\cdot W(\theta_{01})\cdot W(\theta_{11})}_{W(\boldsymbol{\theta}^{(t)})}.$$

## Source Coding (Case 2: Unknown Source) (P. 8)

Some remarks:

- We still have to choose the values for $\alpha_s$ for each non-leaf $s \in \mathcal{S}$ and $W(\theta_s)$ for each $s \in \mathcal{S}$.
- With the choice

$$W(\theta_s) \triangleq \frac{1}{\pi\sqrt{(1-\theta_s)\theta_s}}, \quad \text{for } 0 \le \theta_s \le 1$$

and any $0 < \alpha_s < 1$, $s \in \mathcal{S}$, one can show that

$$L(\bar{\mathbf{x}}_1^n) < \min_t \left[ \min_{\boldsymbol{\theta}} \log \frac{1}{P(\bar{\mathbf{x}}_1^n | t, \boldsymbol{\theta})} + \underbrace{\log \frac{1}{W(t)}}_{\substack{\text{model} \\ \text{redundancy}}} + \underbrace{|\mathcal{S}| \gamma\left(\frac{n}{|\mathcal{S}|}\right)}_{\substack{\text{parameter} \\ \text{redundancy}}} \right] + \underbrace{2}_{\substack{\text{coding} \\ \text{redundancy}}}.$$

for some concave function $\gamma(\cdot)$, where

$$\gamma(z) \triangleq \begin{cases} z & (\text{for } 0 \le z < 1) \\ \frac{1}{2}\log(z) + 1 & (\text{for } 1 \le z) \end{cases}.$$

## Source Coding (Case 2: Unknown Source) (P. 9)

More remarks ...

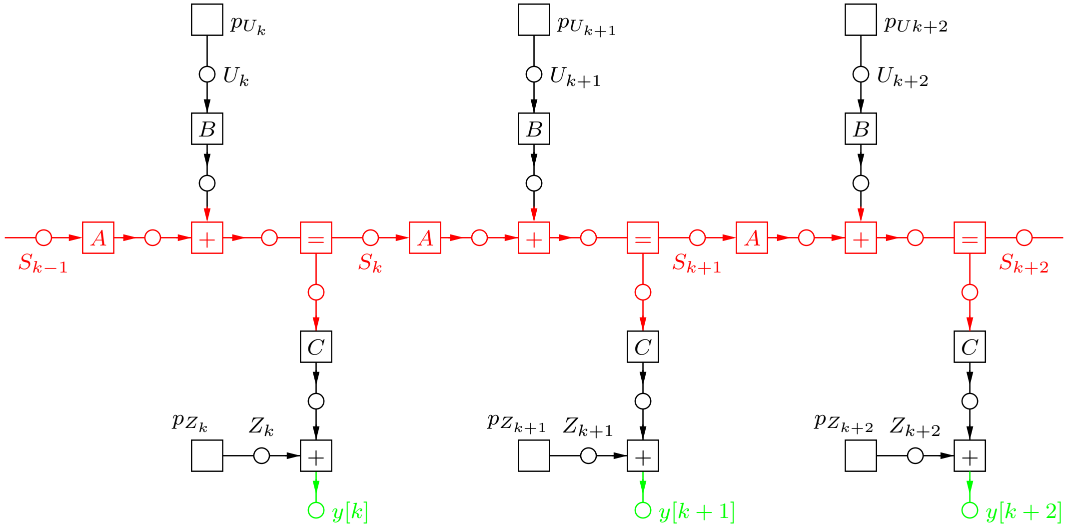
- Can the update rules of the sum-product algorithm be performed efficiently with the above choices?
  - All messages in the upper part of the factor graph are functions of $t$ and $\boldsymbol{\theta}$. Because of the special choice for $W(t, \boldsymbol{\theta})$, it turns out that all messages have this tree structure.
  - Therefore, a summation/integration over $\Sigma$ is a summation over all context trees and an integration over $0 \leq \theta_s \leq 1$ for all $s \in \mathcal{S}$. But because of the above tree structure, this summation can be performed with low complexity, i.e. with message passing in that tree.
  - Note that for the above statement it is crucial that the continous part of the messages are members of a countable set of functions; the above integrations can be performed analytically.
  - Moreover, it turns out that only on the order of $D$ messages have to be updated in the context-tree factor graph at each time step.

## Kalman Filtering (Part 1)

Consider the following state-space model:

$$s_k = As_{k-1} + Bu_k,$$
$$y_k = Cs_k + z_k$$

Assuming that $u_k$ and $z_k$ are statistically independent for all time steps $k$, we obtain the following factor graph for parts of the joint density.




Kalman filtering/smoothing/prediction corresponds then to performing the sum-product/max-product algorithm on this factor graph.

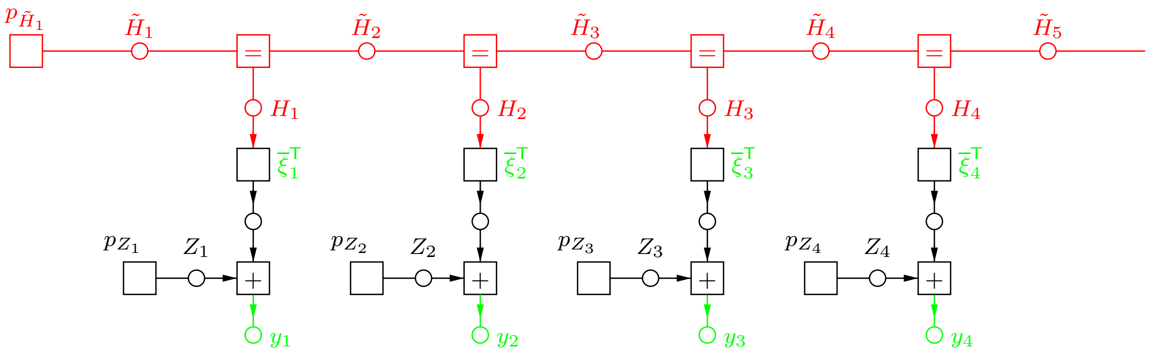
# **Kalman Filtering** (Part 2)

Consider the following problem. We observe the input $x_k$ and the noisy output $y_k$ of a FIR-filter of memory $D$:

$$y_k = \boldsymbol{\xi}_k^{\mathsf{T}} h + z_k,$$

where $\boldsymbol{\xi}_k \triangleq (x_k, x_{k-1}, \ldots, x_{k-D})^{\mathsf{T}}$, $h$ is the vector of unknown FIR filter coefficients, and $z_k$ are assumed to be statistically independent for all time steps $k$.

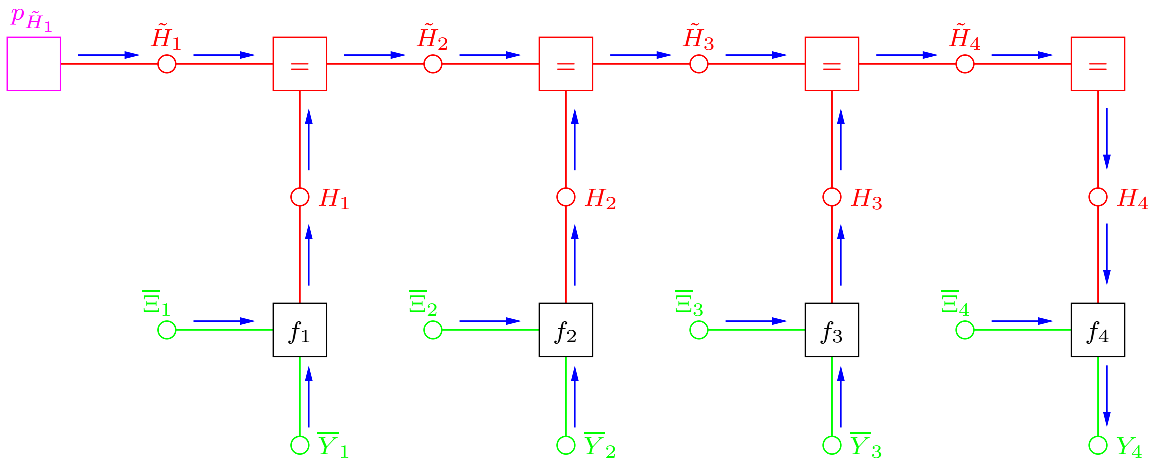
Setting $s_k \triangleq h$, $A \triangleq \mathbf{I}$, $B \triangleq 0$, $C \triangleq \xi_k^{\mathsf{T}}$, we can cast the estimation of $h$ as a Kalman filtering problem.




In the case that the noise $Z_k$ is Gaussian distributed, the update rules of the sum-product/max-product update rule correspond to the RLS (recursive least squares) adaptive filtering update rules.

## **Kalman Filtering** (Part 3)

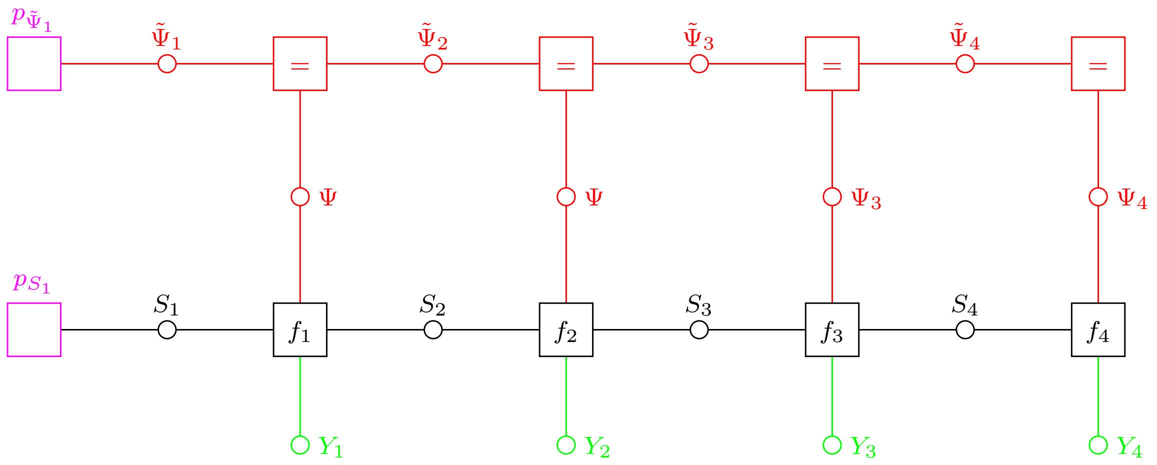
Reorganizing the factor graph, we obtain:




We clearly see the topological relationship to the CTW method.

# Hidden Markov Models

Consider a hidden Markov model where the transitions probabilities and the output probabilites are modeled by the parameter set $\Psi$.

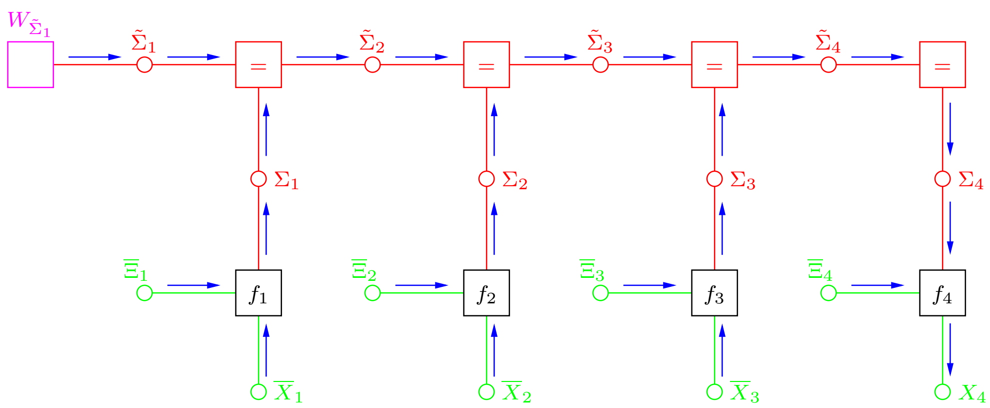


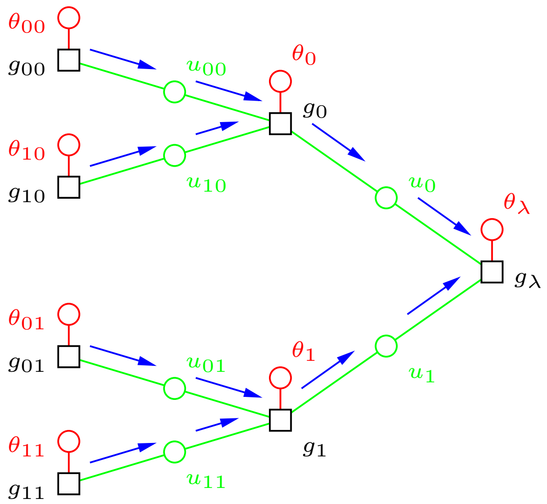

With the right update schedule, one obtains the Baum-Welch learning algorithm.

Connection to source coding. A possible source coding algorithm proceeds as follows: it learns the source statistics, transmits these statistics, and then sends the source that was compressed using the learned statistics.

## **Conclusions and Outlook** (Part 1)

- We have shown a graphical model interpretation of the CTW method.
  It can be seen as a form of structured summary propagation (Dauwels, Loeliger, Merkli, and Ostojic, Allerton 2003).



## Conclusions and Outlook (Part 2)

- We have shown what crucial choices make the algorithm efficient.
- We have shown connections to the RLS adaptive filtering algorithm and to the Baum-Welch learning algorithm.
- A comment on the complexity (Willems, Shtarkov, and Tjalkens, IT Newsletter, March 1997).
  “ The CTW method is generally considered to be rather complex. A state-of-the-art implementation requires 32 MByte of RAM. Today this may seem a lot, however for sure, in ten years from now this is “peanuts.” A challenging problem is to find methods that improve the compression rate of e.g. CTW by making use of the huge amounts of memory that will be available in the future. Of particular interest are of course methods that allow parallel implementation. ”