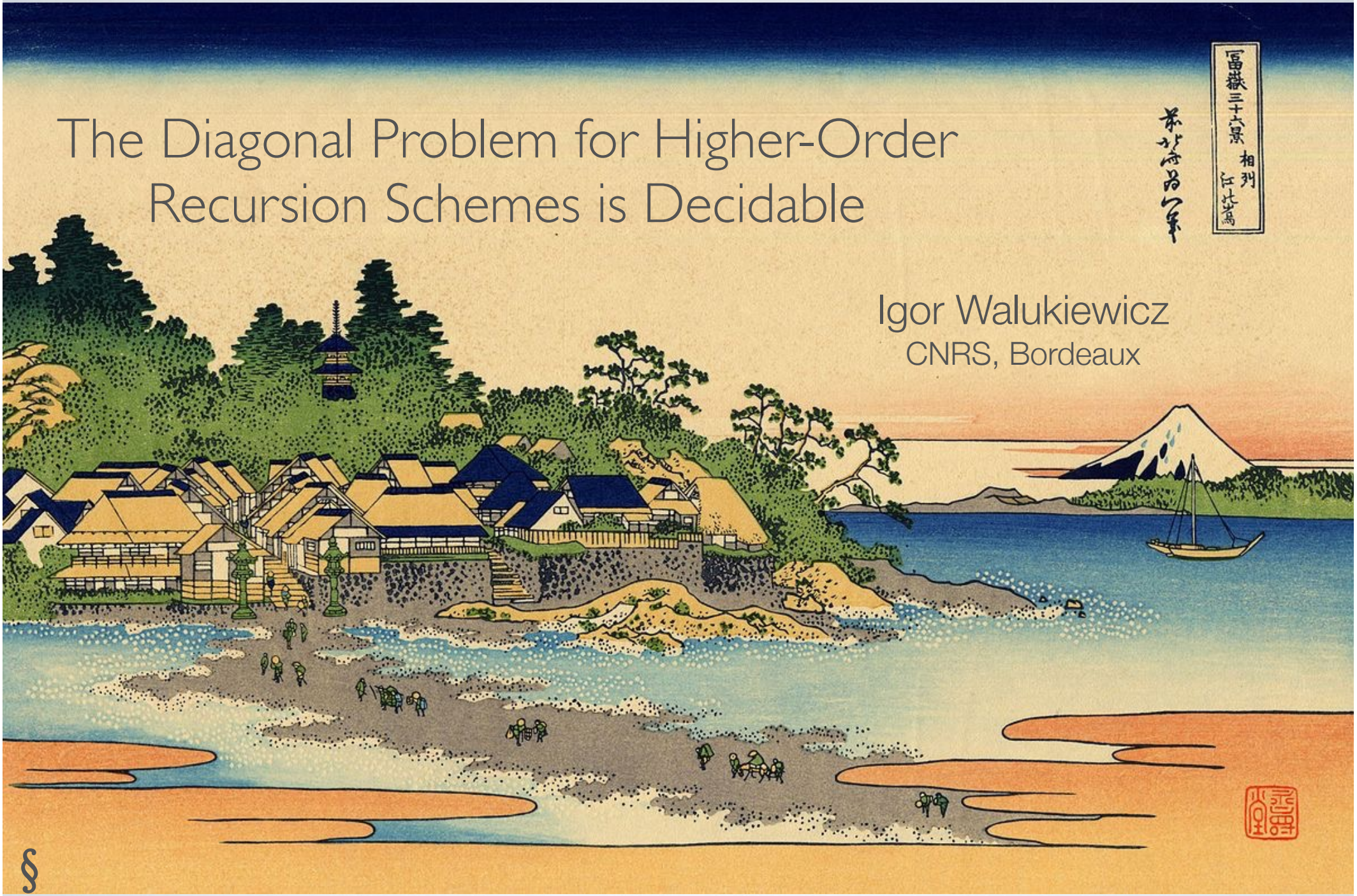
The Diagonal Problem for Higher-Order Recursion Schemes is Decidable



Igor Walukiewicz
CNRS, Bordeaux

§

The diagonal problem:

Given a language $L \subseteq \Sigma^*$, decide if for every $n \in \mathbb{N}$ there is a word in L where every letter from Σ appears at least n times.

a*b*c*

aab*c* + a*b*cc

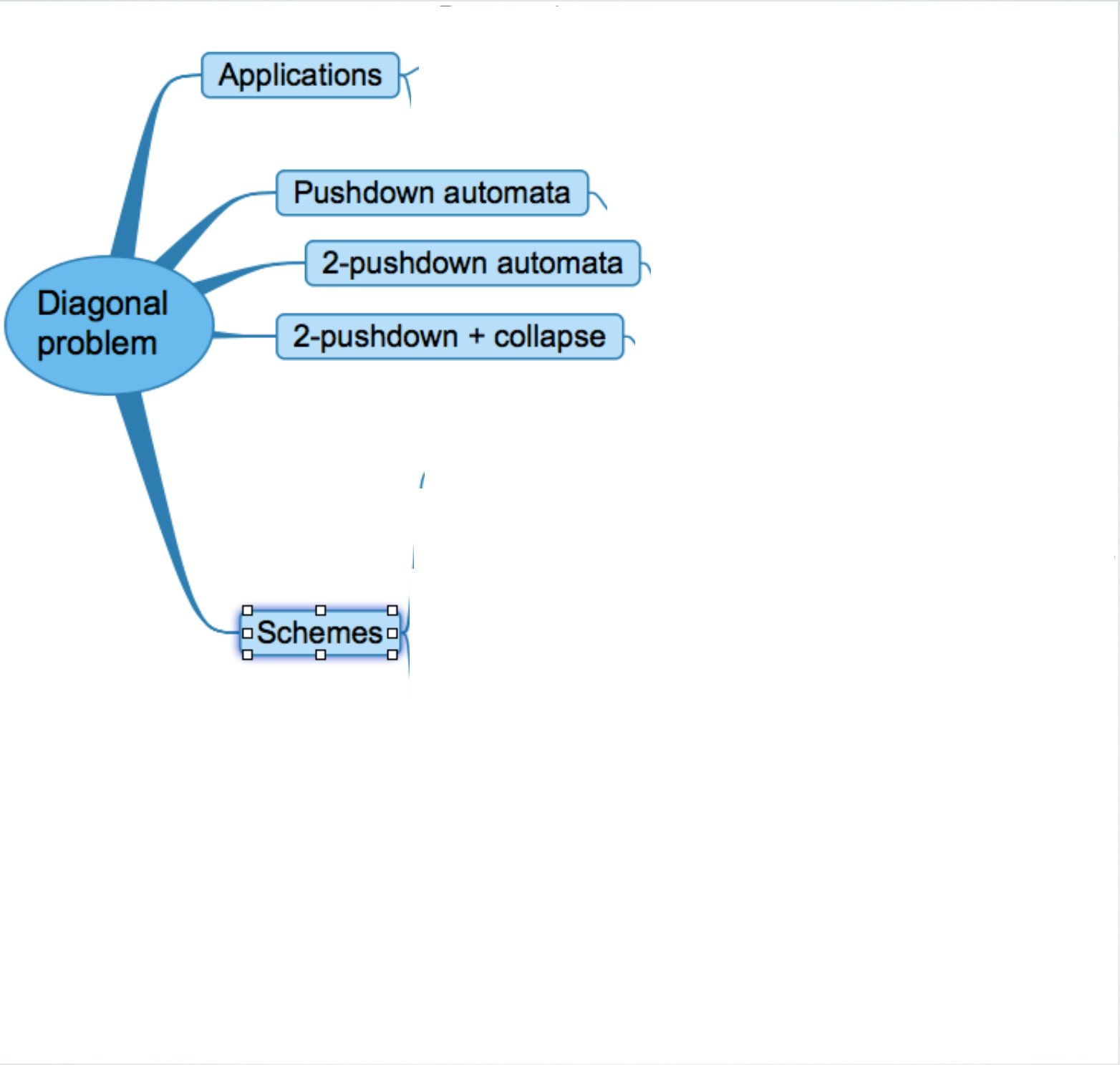
The diagonal problem:

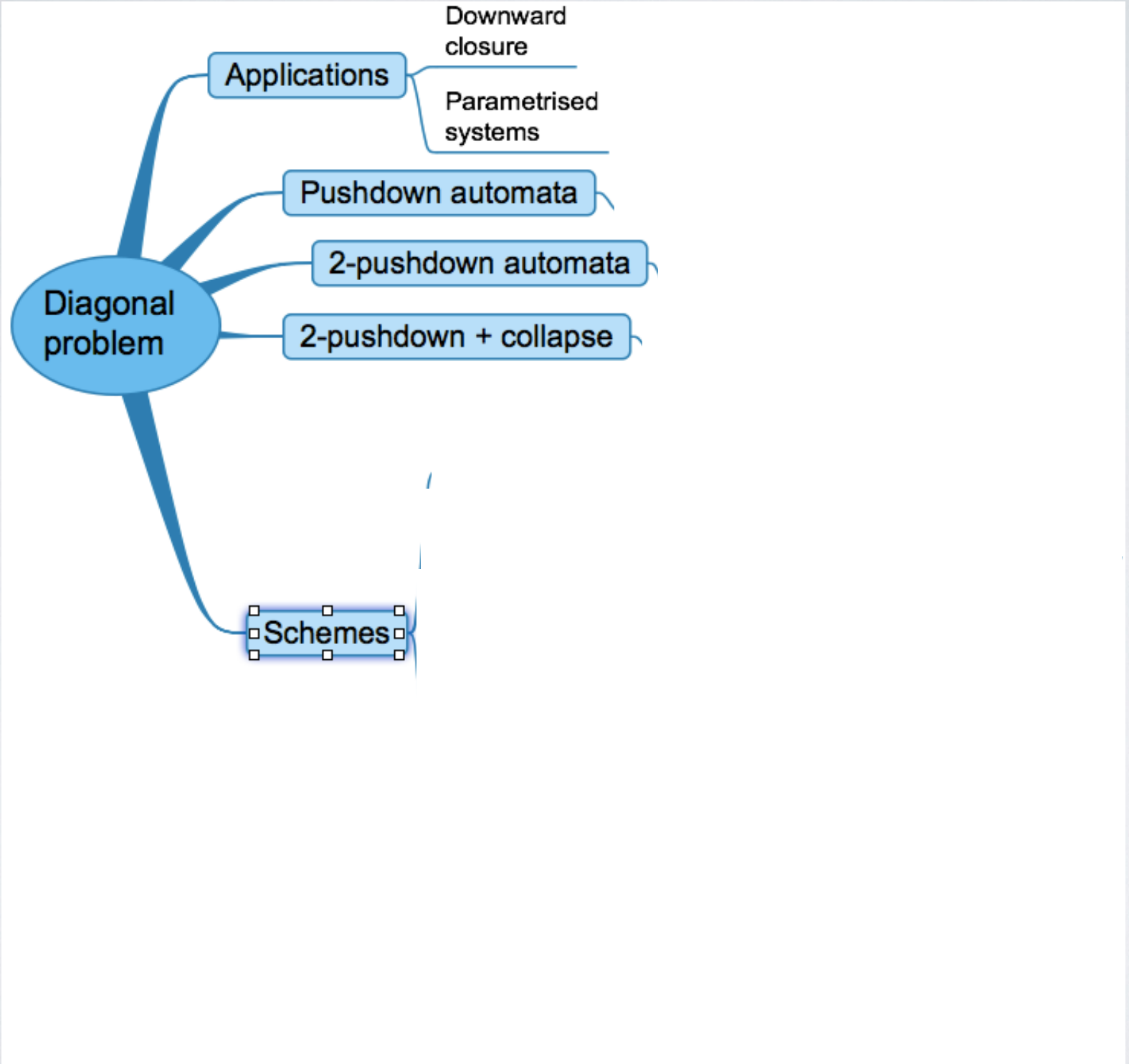
Given a language $L \subseteq \Sigma^*$, decide if for every $n \in \mathbb{N}$ there is a word in L where every letter from Σ appears at least n times.

A langue generated by a scheme

$$S \to F\,b\,e \qquad F\,g\,x \to g\,x \qquad F\,g\,x \to a\,(F\,(B\,g)\,(c\,x)) \qquad B\,g\,x \to b\,(g\,x)$$

generates $a^n b^{n+1} c^n e$.


Applications
Pushdown automata
2-pushdown automata
Diagonal problem
2-pushdown + collapse
Schemes


Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
2-pushdown automata
2-pushdown + collapse
Schemes

Why diagonal problem for schemes?

- It has some applications.
- It is the first non-regular property that is shown decidable.

The diagonal problem:

Given a language $L \subseteq \Sigma^*$, decide if for every $n \in \mathbb{N}$ there is a word in L where every letter from Σ appears at least n times.

Computing downward closure of a language

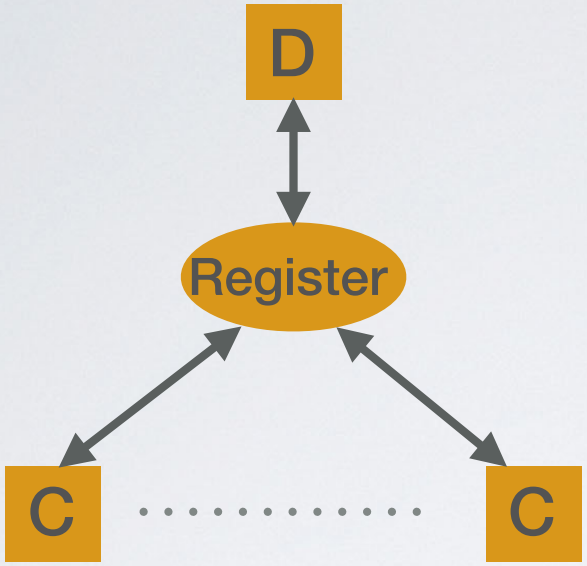
A reduction to the diagonal problem [Zetzsche]:

If a language class is a full trio and has decidable diagonal problem then it has computable downward closures.

Downward closure is computable for:

- Context-free languages [Courcelle]
- Petri Nets [Habermehl, Meyer, Wimmel]
- Order 2 pushdown automata [Zetzsche]
- Higher-order pushdown automata [Hague, Kochems, Ong]

Example: parametrised systems [Hague]



- Every process can read and write to the register
- No locking
- The number of C-processes is not determined

Thm [Muscholl, La Torre, W.]:

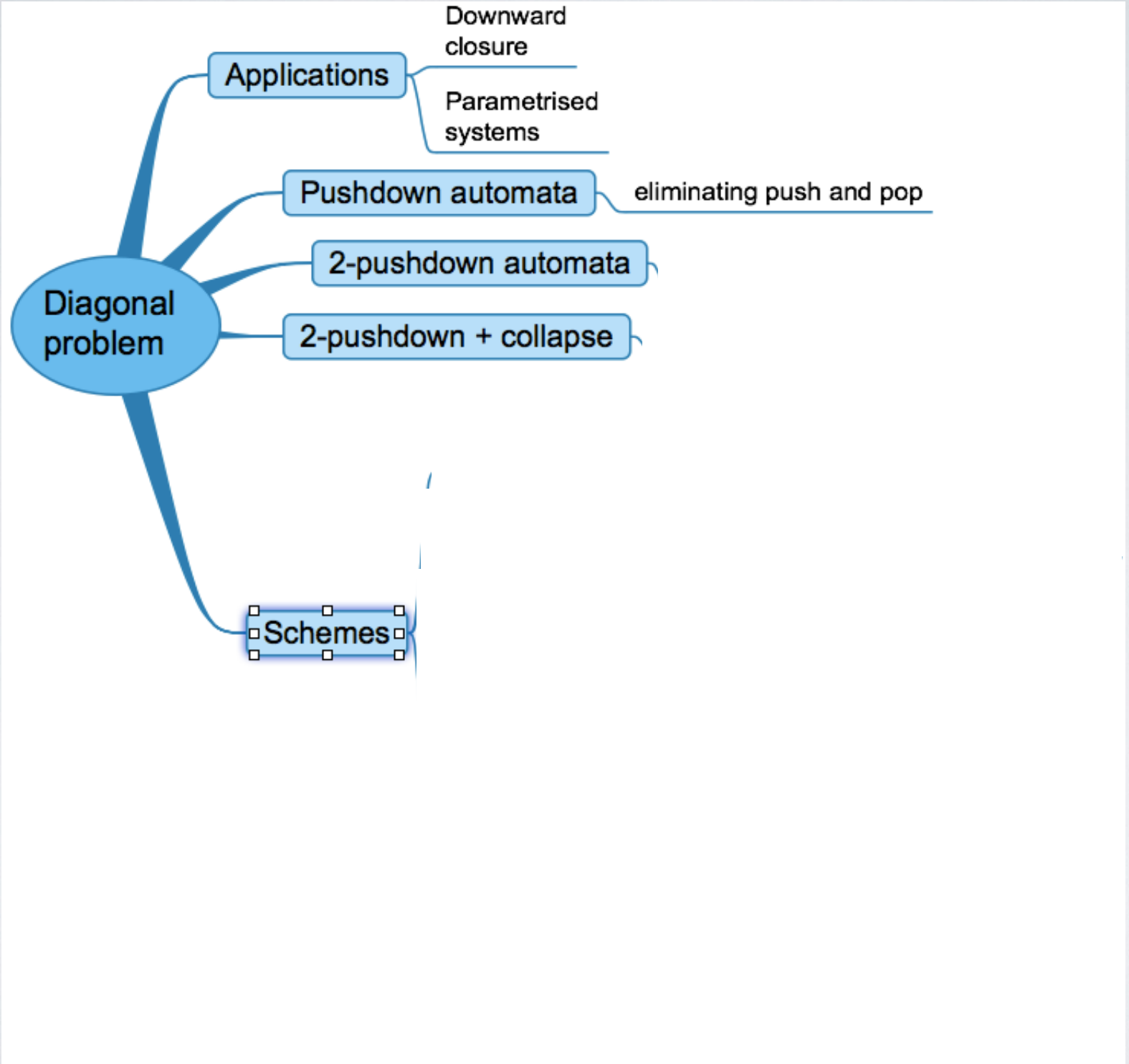
If Class(C) has decidable reachability problem, and
Class(D) has effective downwards closure then
the reachability problem for (Class(C),Class(D))-systems is decidable.

Theorem:

The diagonal problem for higher-order recursive schemes is decidable.

The diagonal problem:

Given a language $L \subseteq \Sigma^*$, decide if for every $n \in \mathbb{N}$ there is a word in L where every letter from Σ appears at least n times.


Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
2-pushdown + collapse
Schemes

A pushdown automaton $\mathcal{A}$:

$(p,\gamma) \to (p', push(\gamma'))$ $\quad$ $(p,\gamma) \to (p', pop)$ $\quad$ $(p,\gamma) \xrightarrow{c} (p', nop)$.

A finite automaton $\mathcal{A}^\sharp$:

$(p,\gamma,r) \xrightarrow{c} (p',\gamma',r')$ $\quad$ if $(p,\gamma) \to (p', push(\gamma'))$ and $Comp_c(r',\gamma,r)$

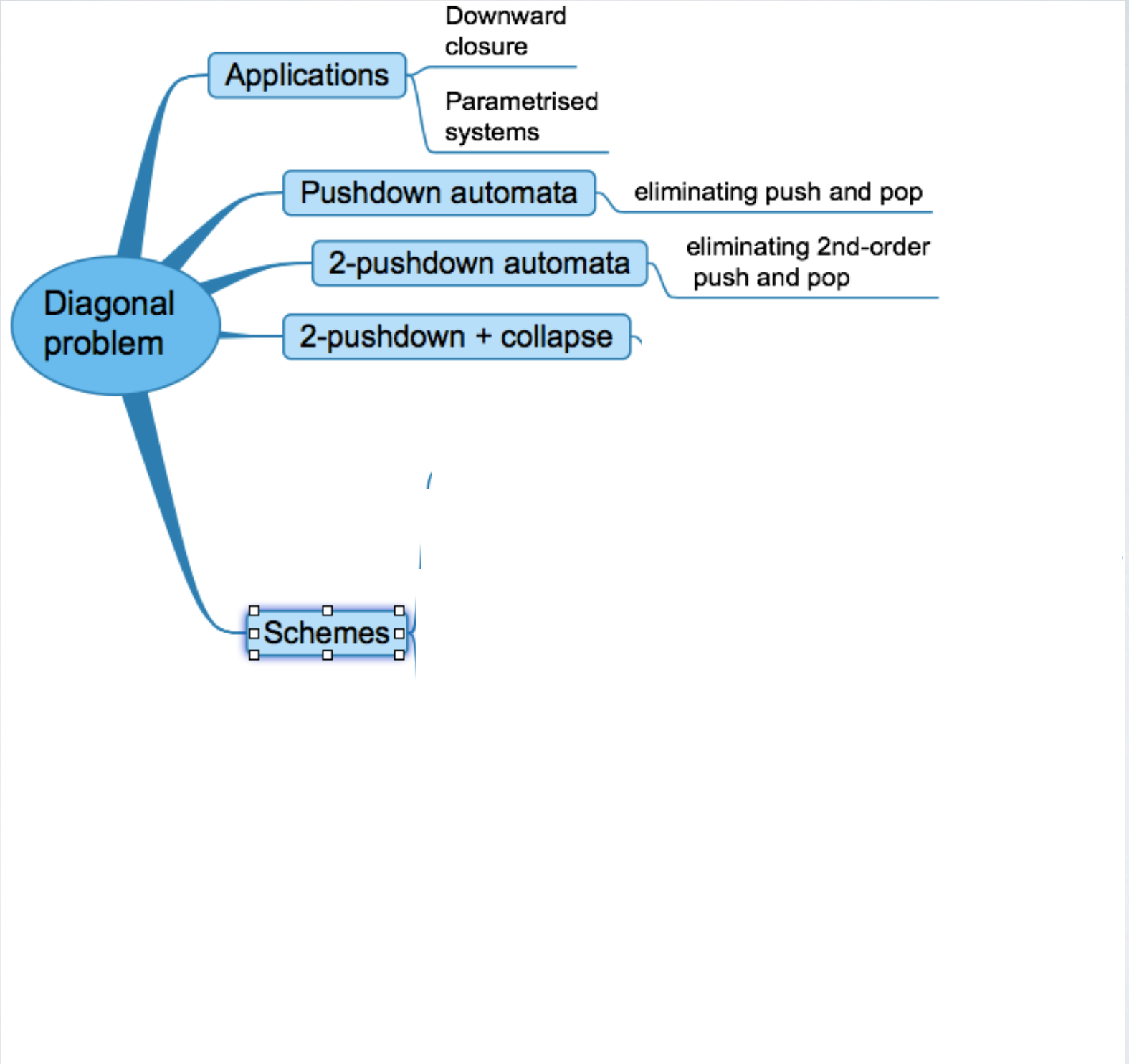
$(p,\gamma,r) \xrightarrow{c} (r',\gamma,r)$ $\quad$ if $(p,\gamma) \to (p', push(\gamma'))$ and $Comp_c(p',\gamma',r')$

$(p,\gamma,r) \to \top$ $\quad$ if $(p,\gamma) \to (r, pop)$

$(p,\gamma,r) \xrightarrow{c} (p',\gamma,r)$ $\quad$ if $(p,\gamma) \xrightarrow{c} (p', nop)$

Fact:

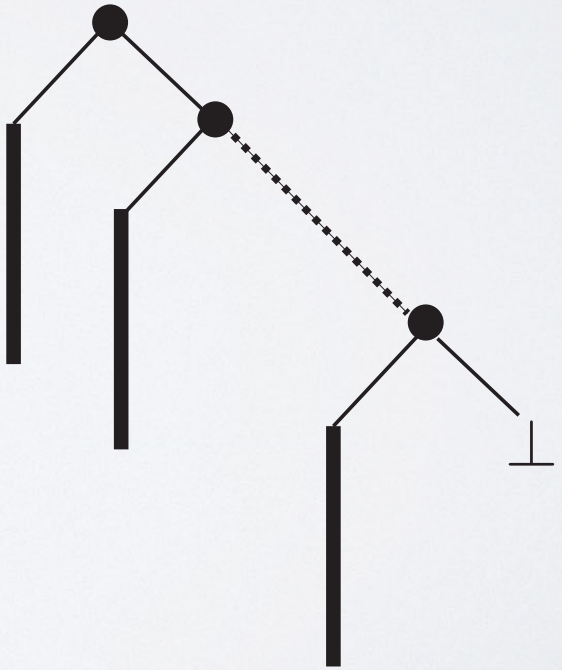
If $\mathcal{A}$ has a run from $(p,\gamma s)$ to (r,s) with m occurrences of b, then $\mathcal{A}^\sharp$ has a run from (p,γ,r) to $\top$ with $\geq \log(m)$ occurrences of b.


Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
Schemes

A 2-pushdown automaton :

$$
\begin{array}{rl}
p, \bullet(x_1, x_2) \to p', \bullet(b(x_1), x_2) & push_1(b) \\
p, \bullet(x_1, x_1) \to p', \bullet(x_1, \bullet(x_1, x_2)) & push_2 \\
p, \bullet(a(x_1), x_2) \to p', \bullet(x_1, x_2) & pop_1 \\
p, \bullet(x_1, \bullet(y_1, y_2)) \to p', \bullet(y_1, y_2) & pop_2 \\
p, \bullet(a(x_1), x_2) \xrightarrow{c} p', \bullet(a(x_1), x_2) & \text{input}
\end{array}
$$

A 2-stack:

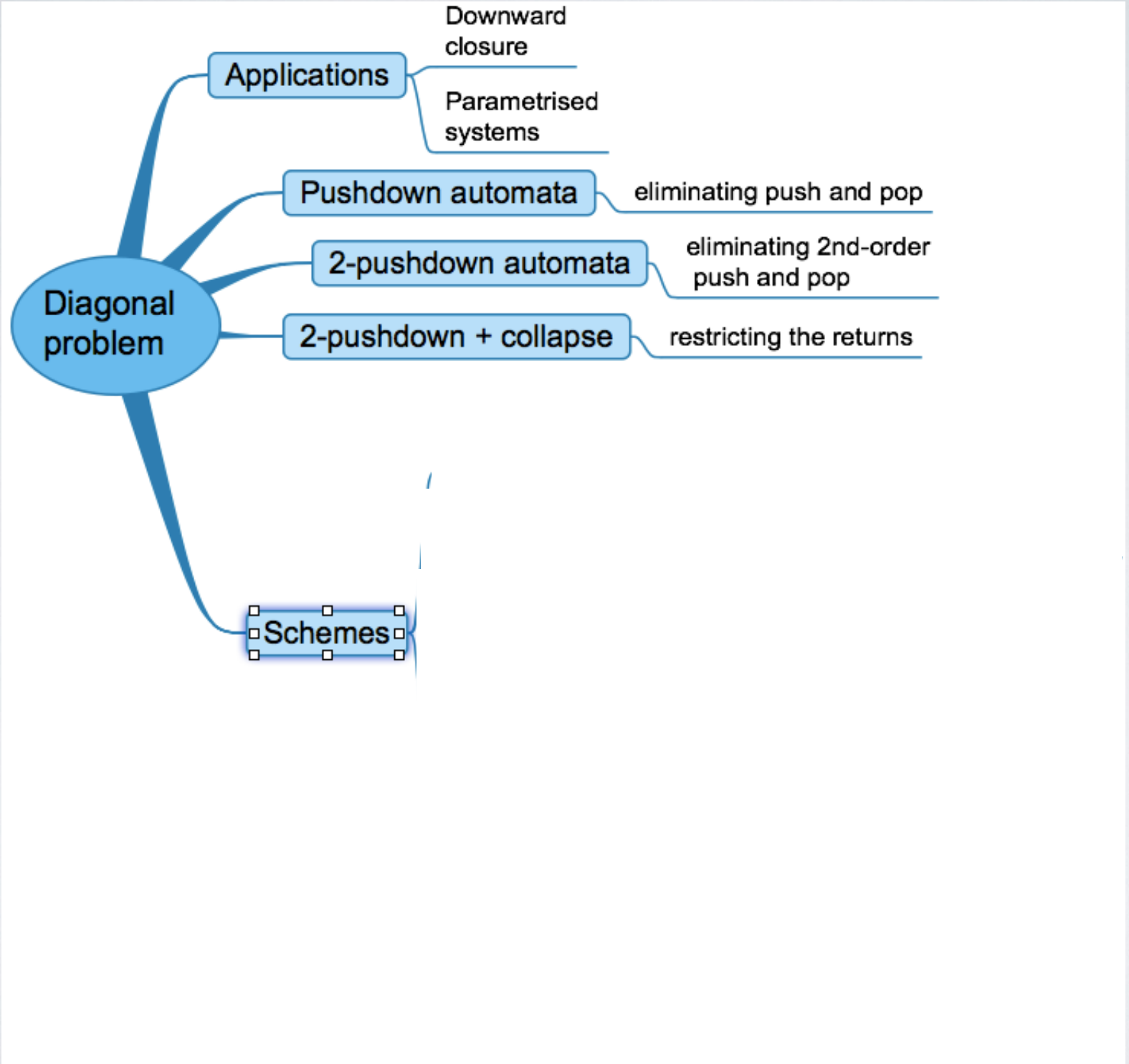


$$p, \bullet(x_1, x_2) \to p', \bullet(b(x_1), x_2) \qquad push_1(b)$$
$$p, \bullet(x_1, x_1) \to p', \bullet(x_1, \bullet(x_1, x_2)) \qquad push_2$$
$$p, \bullet(a(x_1), x_2) \to p', \bullet(x_1, x_2) \qquad pop_1$$
$$p, \bullet(x_1, \bullet(y_1, y_2)) \to p', \bullet(y_1, y_2) \qquad pop_2$$
$$p, \bullet(a(x_1), x_2) \xrightarrow{c} p', \bullet(a(x_1), x_2) \qquad \text{input}$$

Eliminating 2-rules

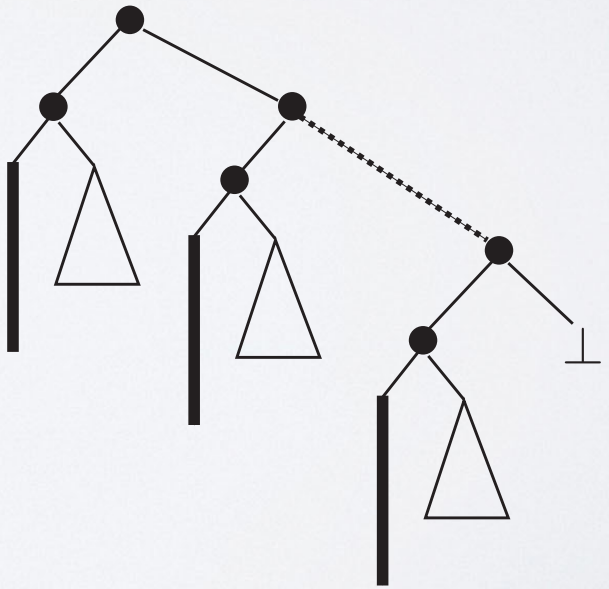
$$p, \bullet(x_1, x_2) \xrightarrow{c} p', \bullet^n(x_1, q) \qquad \text{choose } push_2$$
$$\text{if } Comp_c(q, \bullet(x_1, x_2))$$
$$p, \bullet(x_1, x_2) \xrightarrow{c} q, \bullet(x_1, x_2) \qquad \text{jump over}$$
$$\text{if } Comp_c(p', \bullet(x_1, q))$$
$$p, \bullet(x_1, q) \xrightarrow{c} \top \qquad pop_2$$
$$\text{if } q = p'$$

If $\mathcal{A}$ has a run $(p, t) \to (q, s)$ with n occurrences of b then $\mathcal{A}^\sharp$ has a run $(p, t[q/s]) \to \top$ with $\geq \log(m)$ occurrences of b.


Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes

A 2-pushdown automaton with collapse :

$$
\begin{aligned}
p, \bullet(x_1, x_2) &\to p', \bullet(b(x_1, \bullet(x_1, x_2)), x_2,) && push_1(b)\\
p, \bullet(x_1, x_2) &\to p', \bullet(x_1, \bullet(x_1, x_2))) && push_2\\
p, \bullet(a(x_1, y), x_2) &\to p', \bullet(x_1, x_2) && pop_1\\
p, \bullet(x_1, x_2) &\to p', x_2 && pop_2\\
p, \bullet(a(x_1, y), x_2,) &\to p', y && \text{collapse}\\
p, \bullet(a(x_1, y), x_2) &\xrightarrow{c} p', \bullet(a(x_1, y), x_2) && \text{input}
\end{aligned}
$$



[Clemente, Parys, Salvati, W. ’15]

A 2-pushdown automaton with collapse :

$$
\begin{array}{rll}
p, \bullet(x_1, x_2) & \to p', \bullet(b(x_1, \bullet(x_1, x_2)), x_2,) & push_1(b) \\
p, \bullet(x_1, x_2) & \to p', \bullet(x_1, \bullet(x_1, x_2))) & push_2 \\
p, \bullet(a(x_1, y), x_2) & \to p', \bullet(x_1, x_2) & pop_1 \\
p, \bullet(x_1, y) & \to p', y & pop_2 \\
p, \bullet(a(x_1, y), x_2,) & \to p', y & \text{collapse} \\
p, \bullet(a(x_1, y), x_2) & \xrightarrow{c} p', \bullet(a(x_1, y), x_2) & \text{input}
\end{array}
$$

Problem: When doing $push_2$ we cannot be sure that the computation will go back to x_2.

Actually some computations will and some will not.

The main idea:
guess what exit will be used and record this guess in the tree.

Convert $\mathcal{A}$ to $\mathcal{A}^{\downarrow}$ working on tree stacks with a highlighted path.

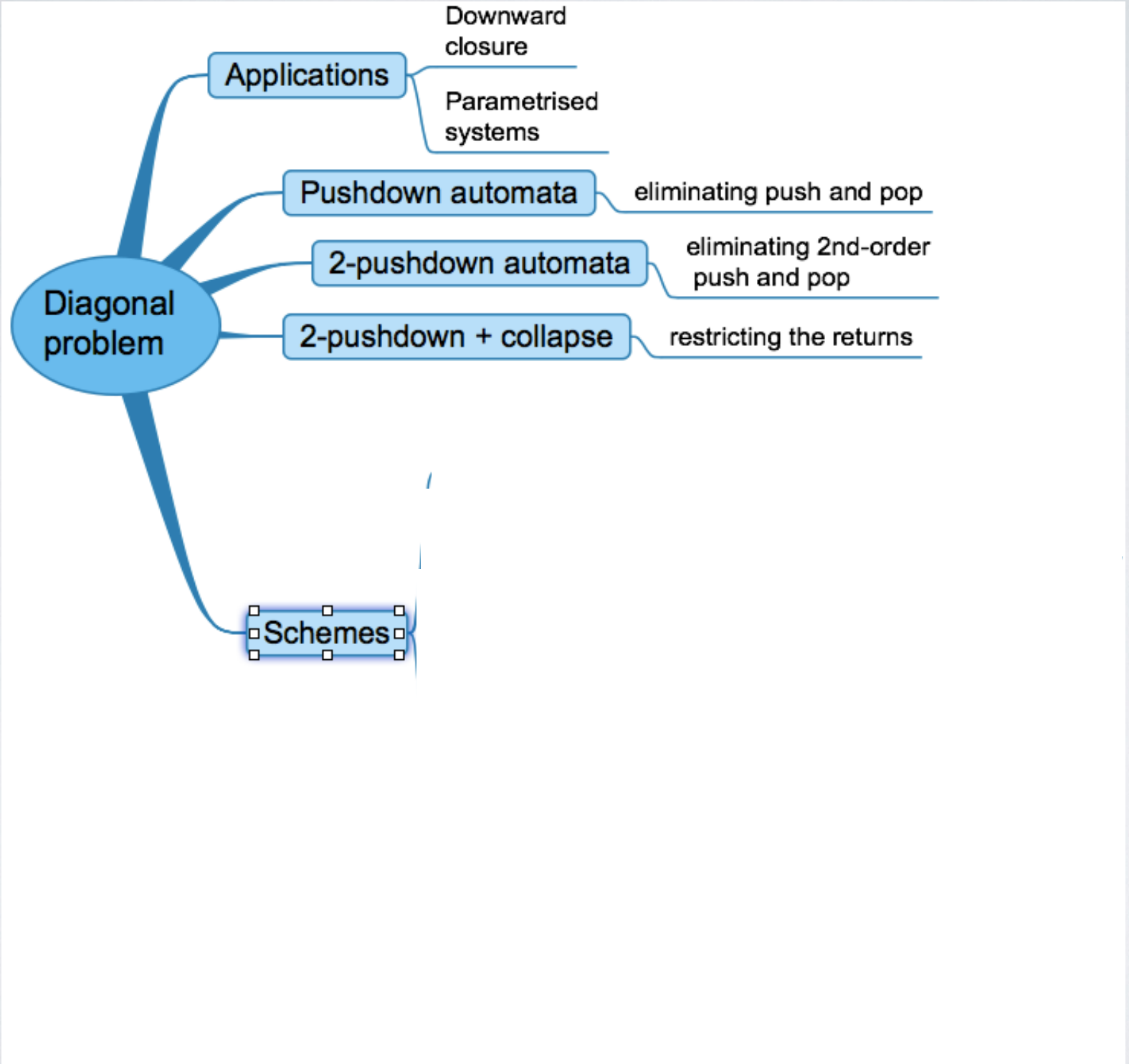
Rule

$$p, \bullet(x_1, x_2) \to p', \bullet(b(x_1), x_2) \qquad push_1(b)$$

is translated to

$$p, \bullet^i(x_1, x_2) \to p', \bullet^j(b^k(x_1, \bullet^l(x_1, x_2)), x_2,) \qquad push_1(b)$$

if $i = 2$ then $j = 2$ or $(j = 1 \wedge k = l = 2)$.
if $i = 1$ then $(j = 1 \wedge k = 2 \wedge l = 1)$ or $(j = 1 \wedge k = 2)$.

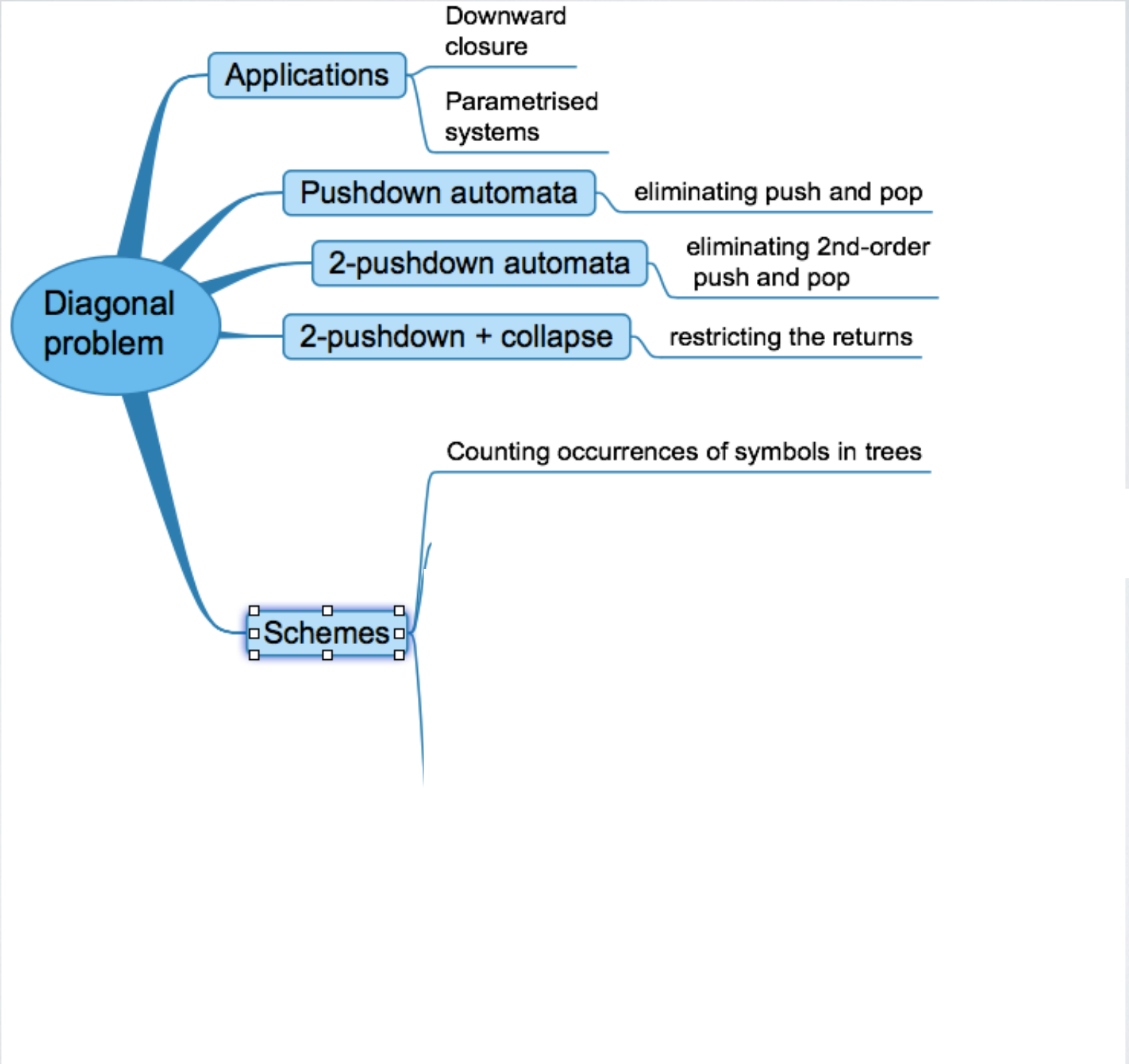

Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes

Back to schemes

A langue generated by a scheme

$$S \to F\,b\,e \qquad F\,g\,x \to g\,x \qquad F\,g\,x \to a\,(F\,(B\,g)\,(c\,x)) \qquad B\,g\,x \to b\,(g\,x)$$

generates $a^n b^{n+1} c^n e$.

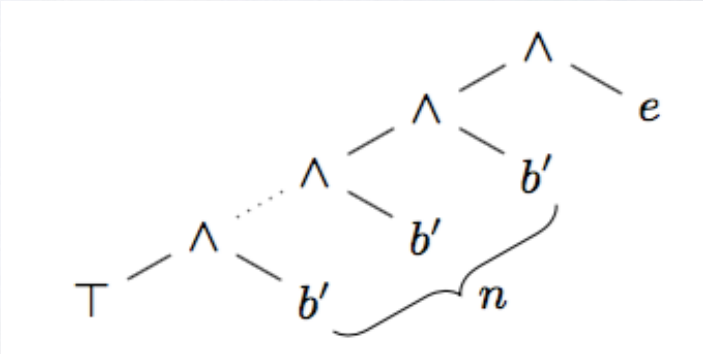

Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes
Counting occurrences of symbols in trees

We will count letters in trees not in words

We extend $\Sigma_{\mathcal{S}}$ to $\Sigma_{\mathcal{S}'}$ by adding a', $\top : 0$, $\wedge : 2$.

Two trees K and K' are equivalent if they have the same number of occurrences of each symbol counting a and a' as the same symbol, and ignoring $\top$, $\wedge$.

$b^n e$



Lowering the order of a scheme

Operation on types:

$$o\downarrow = o, \quad \text{and} \quad (\beta \to \gamma)\downarrow = \begin{cases} \gamma\downarrow & \text{if } \beta = o, \\ (\beta\downarrow) \to (\gamma\downarrow) & \text{otherwise.} \end{cases}$$

Translation of a scheme:

$$S \to F\, e$$

$$F\, x \to x$$

$$F\, x \to F\, (b\, x)$$

Generated trees:

$$b^n e$$

Lowering the order of a scheme

Operation on types:

$$o\downarrow = o, \quad \text{and} \quad (\beta \to \gamma)\downarrow = \begin{cases} \gamma\downarrow & \text{if } \beta = o, \\ (\beta\downarrow) \to (\gamma\downarrow) & \text{otherwise.} \end{cases}$$

Translation of a scheme:

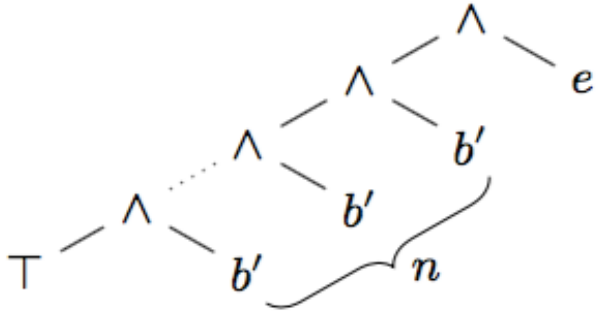
$$S \to F\, e \qquad S' \to \begin{array}{ccc} & \wedge & \\ F' & & e \end{array}$$

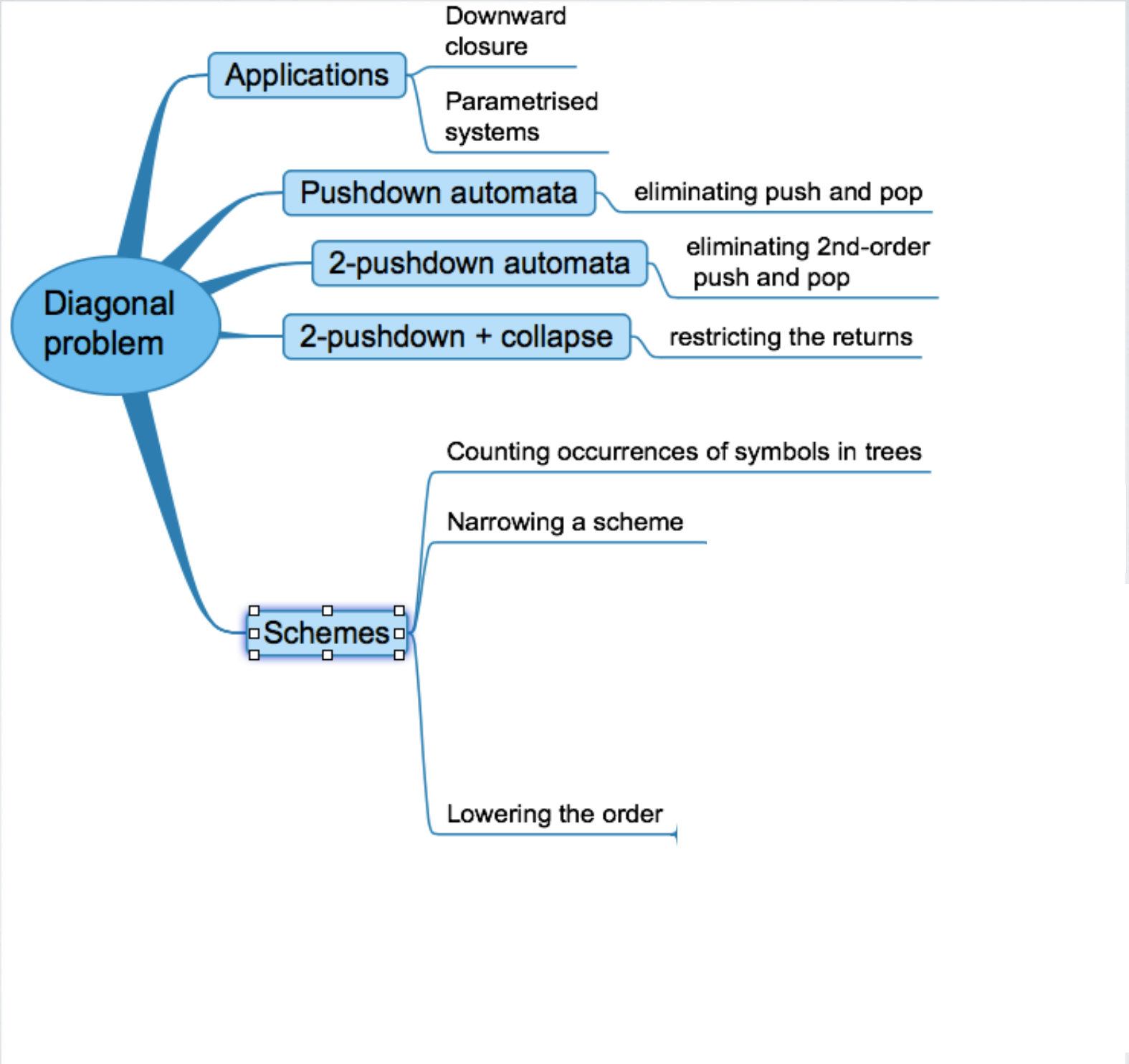
$$F\, x \to x \qquad F' \to \top$$

$$F\, x \to F\, (b\, x) \qquad F' \to \begin{array}{ccc} & \wedge & \\ F' & & b' \end{array}$$

Generated trees:

$$b^n e$$




Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes
Counting occurrences of symbols in trees
Narrowing a scheme
Lowering the order

Theorem:

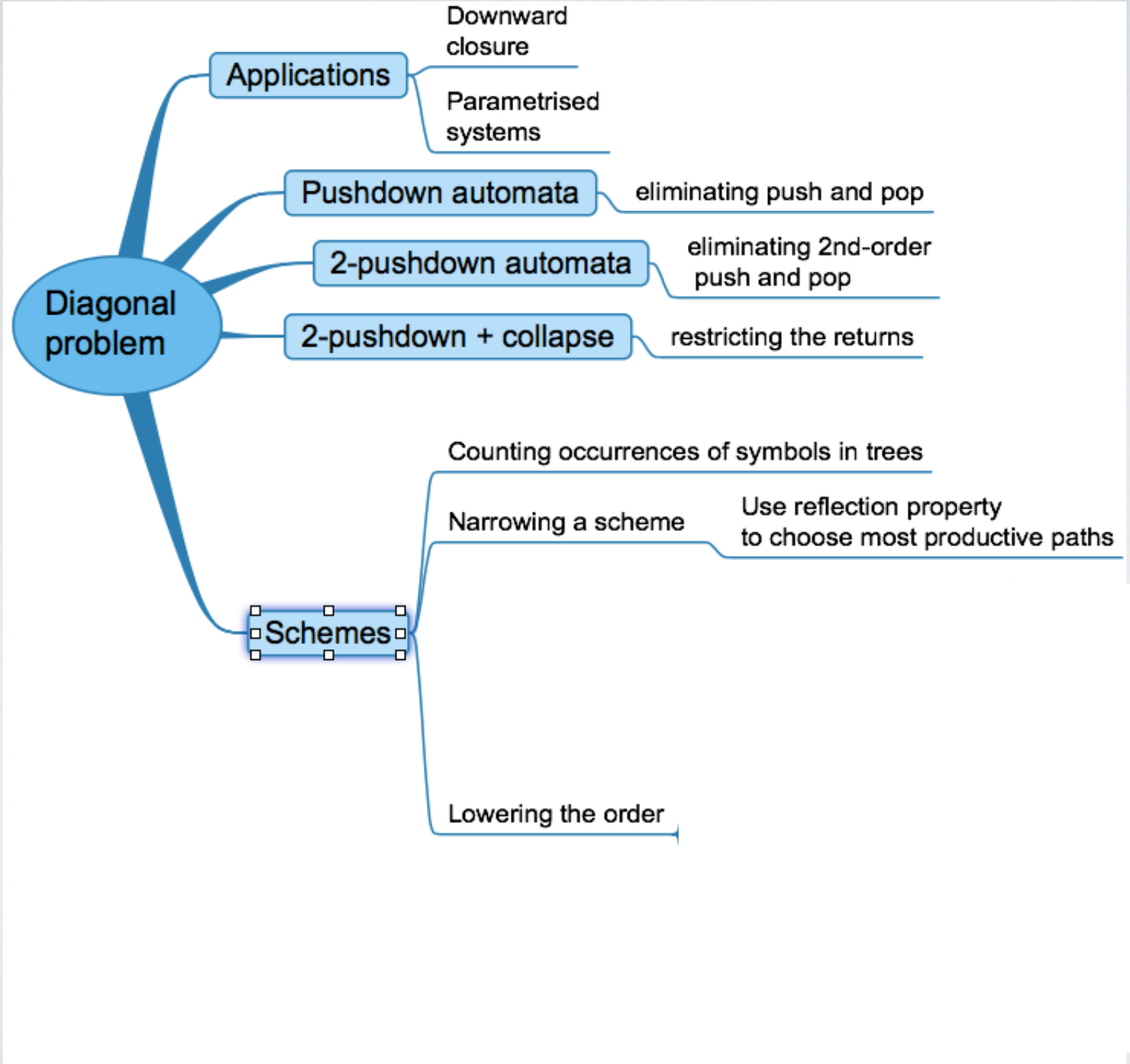
The diagonal problem for higher-order recursive schemes is decidable.

1. Narrowing a scheme:

Constructing a narrow scheme which has the diagonal property iff the original scheme does.

2. Lowering the order of a narrow scheme

Constructing a scheme of a lower order that generates tress equivalent to those generated by the narrow scheme.

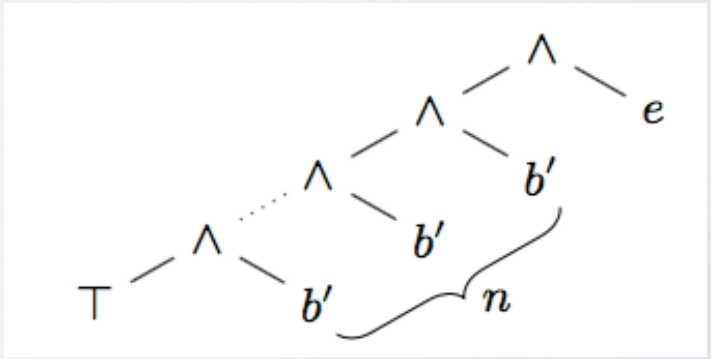

Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes
Counting occurrences of symbols in trees
Narrowing a scheme
Use reflection property to choose most productive paths
Lowering the order

Narrowing a scheme

A tree K is Σ_0-narrow if all leaves of K are labelled with Σ_0 and every constant from Σ_0 appears exactly once.

A HORS is Σ_0-narrow if it generates only Σ_0-narrow trees.

$b^n e$



Narrowing a scheme

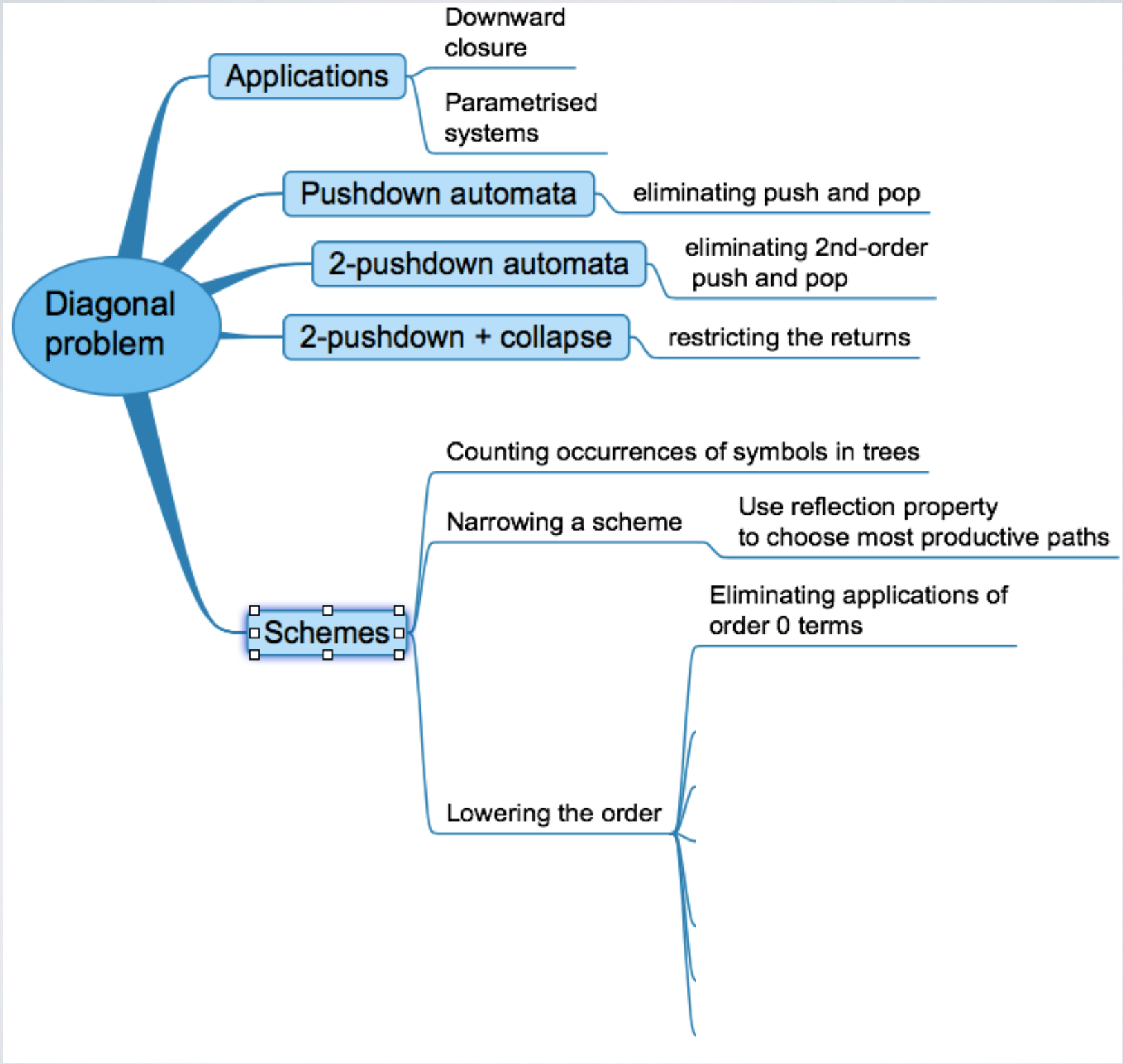
Producing a run tree:

We can create a HORS $\mathcal{S}'$ that is of the same order as $\mathcal{S}$, and generates run trees of some finite automaton on $\mathcal{B}$ on trees generated by $\mathcal{S}$.

We can restrict to parts of the run determined by some $Q' \subseteq Q$.

Narrowing a HORS:

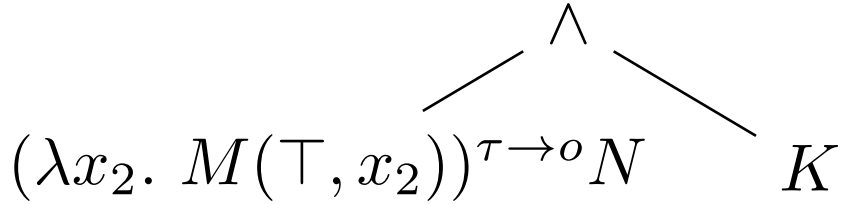
For a HORS $\mathcal{S}$ and a set of symbols Σ, one can construct a narrow HORS $\mathcal{S}'$ of the same order as $\mathcal{S}$, and sets of symbols $\Sigma_1, \ldots, \Sigma_k$ such that $Diag_{\Sigma}(\mathcal{S})$ holds iff there is $i \in \{1, \ldots, k\}$ for which $Diag_{\Sigma_i}(\mathcal{S}')$ holds.


Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes
Counting occurrences of symbols in trees
Narrowing a scheme
Use reflection property to choose most productive paths
Eliminating applications of order 0 terms
Lowering the order

Lowering the order of a scheme

We omit arguments of type o: $o{\downarrow} = o, \quad \text{and} \quad (\beta \to \gamma){\downarrow} = \begin{cases} \gamma{\downarrow} & \text{if } \beta = o, \\ (\beta{\downarrow}) \to (\gamma{\downarrow}) & \text{otherwise.} \end{cases}$

$$(\lambda x_1, x_2.\ M(x_1, x_2))^{o \to \tau \to o} K\ N$$



Example 1

Translation:

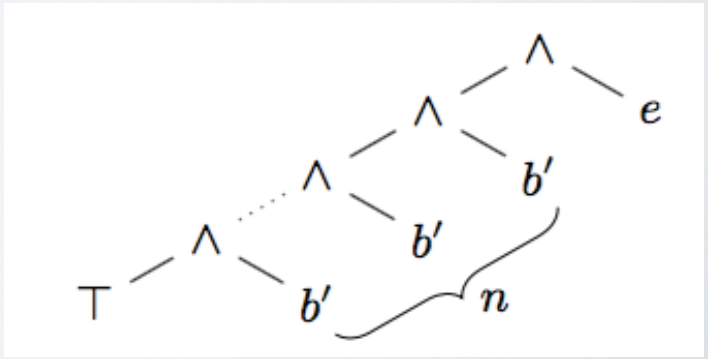
$$S \to F\, e \qquad\qquad S' \to \wedge(F', e)$$

$$F\, x \to x \qquad\qquad F' \to \top$$

$$F\, x \to F\,(b\, x) \qquad\qquad F' \to \wedge(F', b')$$

Generated trees:

$$b^n e$$



Example 2

Translation:

$$S \to F\, b\, e$$

$$F\, g\, x \to g\, x$$

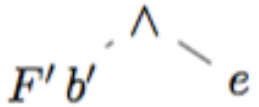
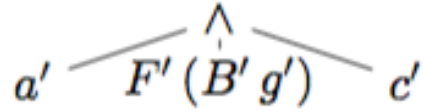
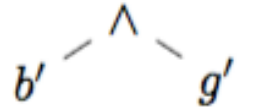
$$F\, g\, x \to a\, (F\, (B\, g)\, (c\, x))$$

$$B\, g\, x \to b\, (g\, x)$$

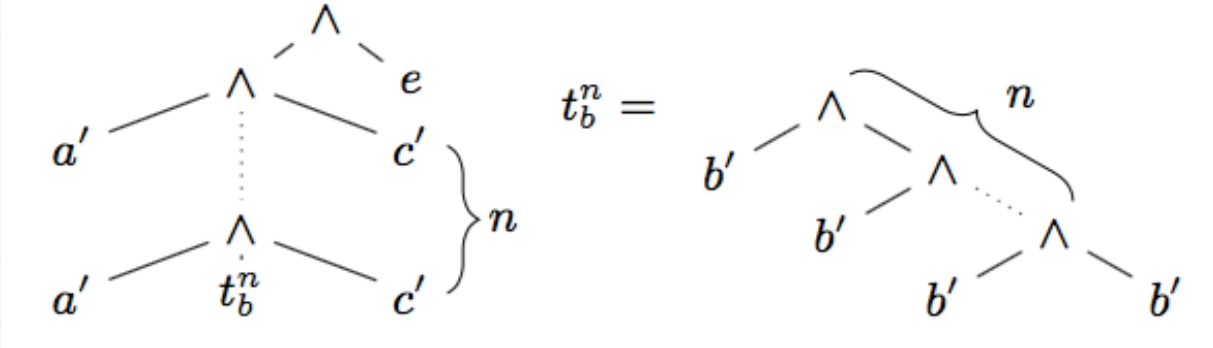
$$a^n b^{n+1} c^n e$$

Example 2

Translation:

$S \to F\, b\, e$	$S' \to$ 
$F\, g\, x \to g\, x$	$F'\, g' \to g'$
$F\, g\, x \to a\, (F\, (B\, g)\, (c\, x))$	$F'\, g' \to$ 
$B\, g\, x \to b\, (g\, x)$	$B'\, g' \to$ 

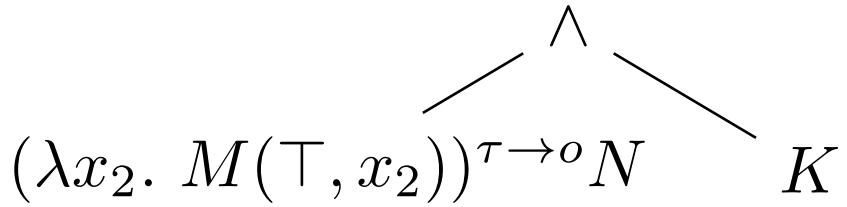
$$a^n b^{n+1} c^n e$$



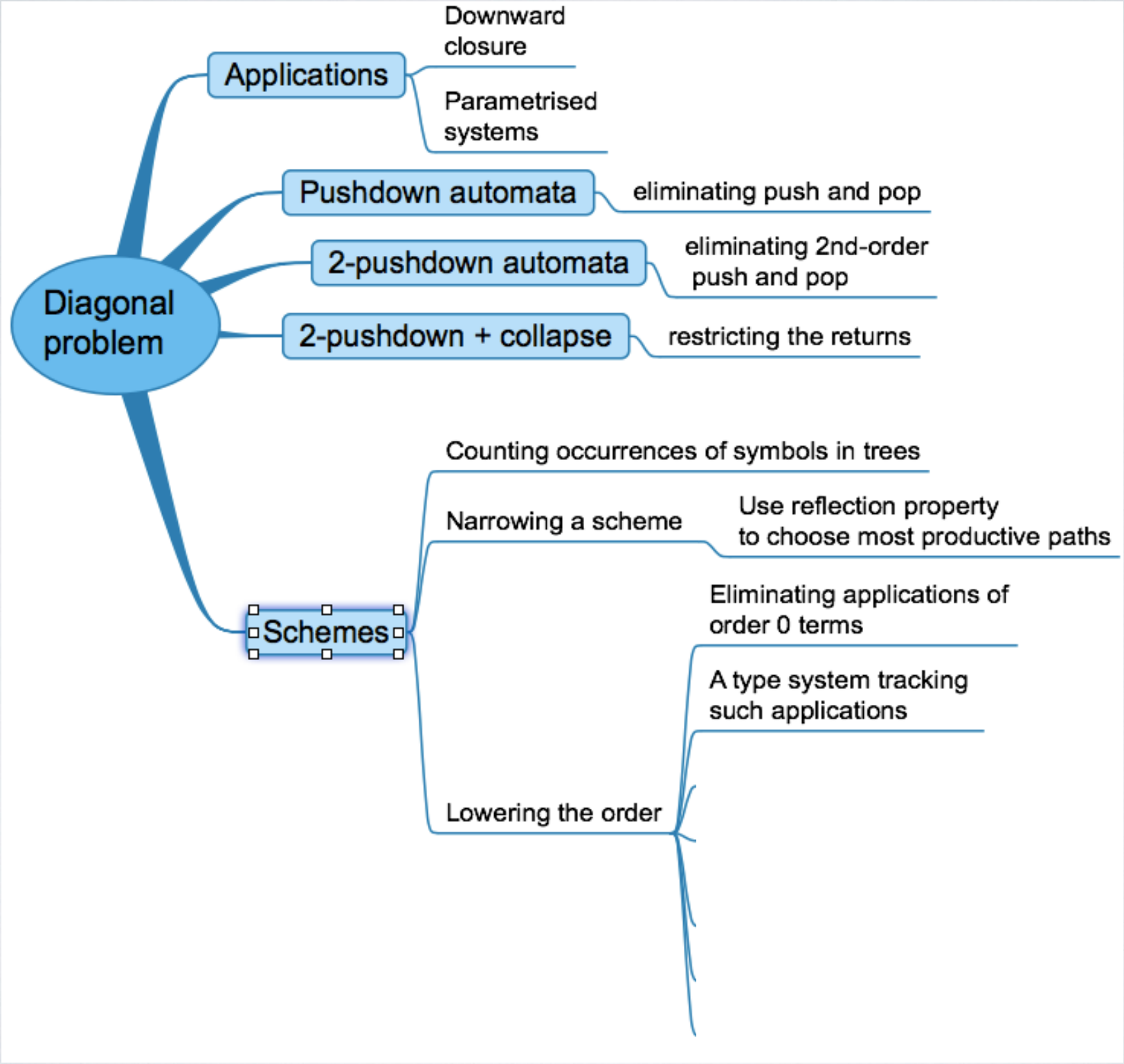
Lowering the order of a scheme

We omit arguments of type o: $o{\downarrow} = o, \quad \text{and} \quad (\beta \to \gamma){\downarrow} = \begin{cases} \gamma{\downarrow} & \text{if } \beta = o, \\ (\beta{\downarrow}) \to (\gamma{\downarrow}) & \text{otherwise.} \end{cases}$

$$(\lambda x_1, x_2.\ M(x_1, x_2))^{o \to \tau \to o} K\ N$$



We need to know that x_1 is used in M, and that it is used exactly once.


Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes
Counting occurrences of symbols in trees
Narrowing a scheme
Use reflection property to choose most productive paths
Lowering the order
Eliminating applications of order 0 terms
A type system tracking such applications

A type system

$\mathcal{T}^o = \{\mathbf{r}\}$

$\mathcal{T}^{\alpha\to\beta} = \{(S_1,\tau_1),\ldots,(S_k,\tau_k)\} \to \tau$

$(S_i,\tau_i) \in \mathcal{LT}^\alpha, \quad S_i \cap S_j = \emptyset$

$\tau \in \mathcal{T}^\beta$

$\mathcal{LT}^\alpha = \mathcal{P}(\Sigma_0) \times \mathcal{T}^\alpha$, if $\alpha = o$ then the first component not empty.

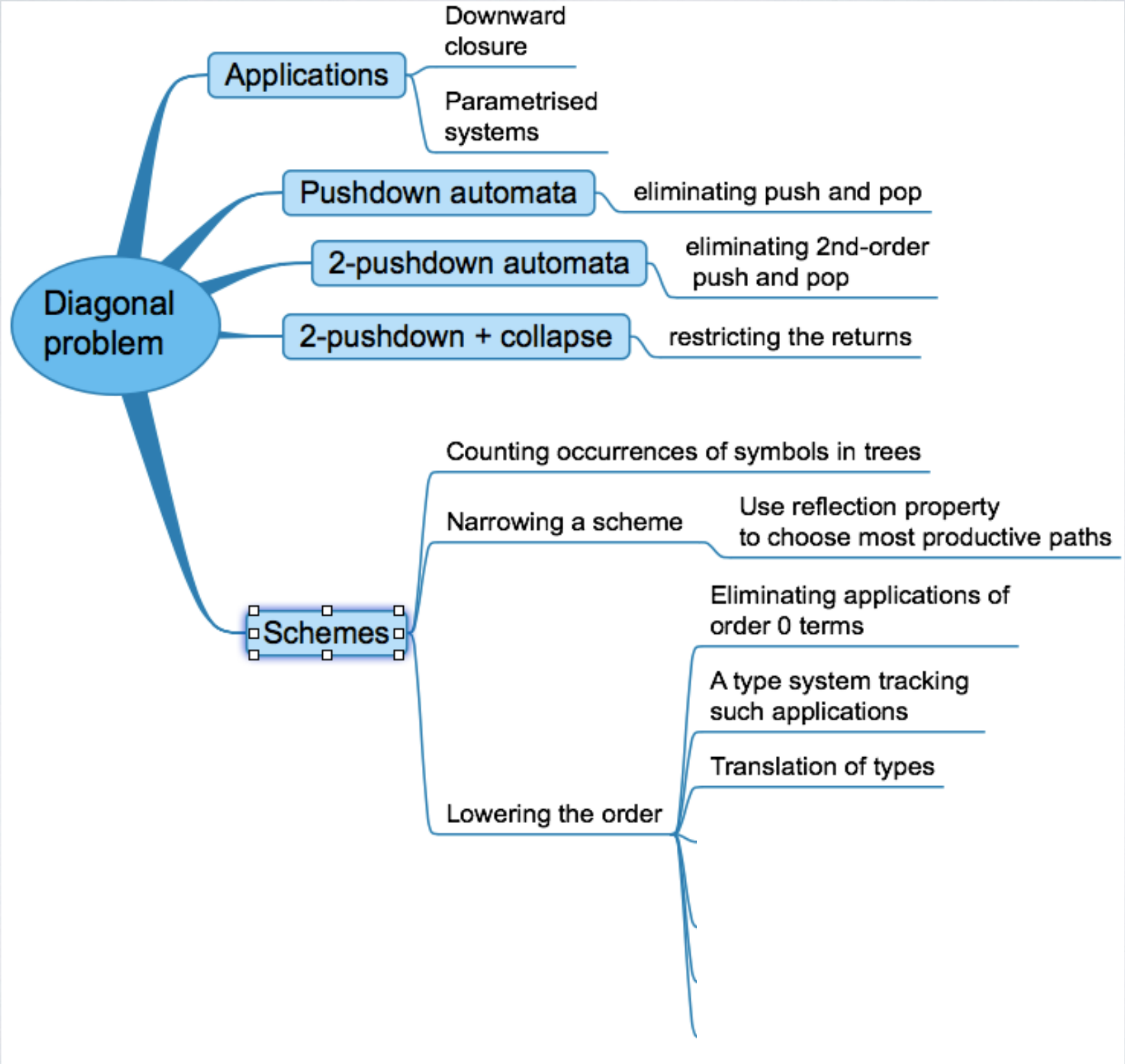
Typing restricts what trees can be derived

$$\frac{}{\Gamma, x:\lambda \vdash x:\lambda} \qquad \frac{}{\Gamma \vdash A : (\emptyset,\tau)} \qquad \frac{}{\Gamma \vdash a^o : (\{a\},\mathbf{r})}$$

$$\frac{r \geq 1}{\Gamma \vdash a^{o^r\to o} : (\emptyset, \{(S_1,\mathbf{r})\} \to \cdots \to \{(S_r,\mathbf{r})\} \to \mathbf{r})}$$

$$\frac{\Gamma \vdash L : (S_0, \{(S_1,\tau_1),\ldots,(S_k,\tau_k)\} \to \tau) \qquad \Gamma \vdash M : (S_i,\tau_i) \text{ for each } i \in \{1,\ldots,k\}}{\Gamma \vdash L\,M : (S_0 \cup \cdots \cup S_k, \tau)}$$

where $S_0 \cap (S_1 \cup \cdots \cup S_k) = \emptyset$


Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes
Counting occurrences of symbols in trees
Narrowing a scheme
Use reflection property to choose most productive paths
Lowering the order
Eliminating applications of order 0 terms
A type system tracking such applications
Translation of types

A type system

$\mathcal{T}^o = \{\mathbf{r}\}$

$$\mathcal{T}^{\alpha\to\beta} = \{(S_1,\tau_1),\ldots,(S_k,\tau_k)\} \to \tau$$
$$(S_i,\tau_i) \in \mathcal{LT}^\alpha, \quad S_i \cap S_j = \emptyset$$

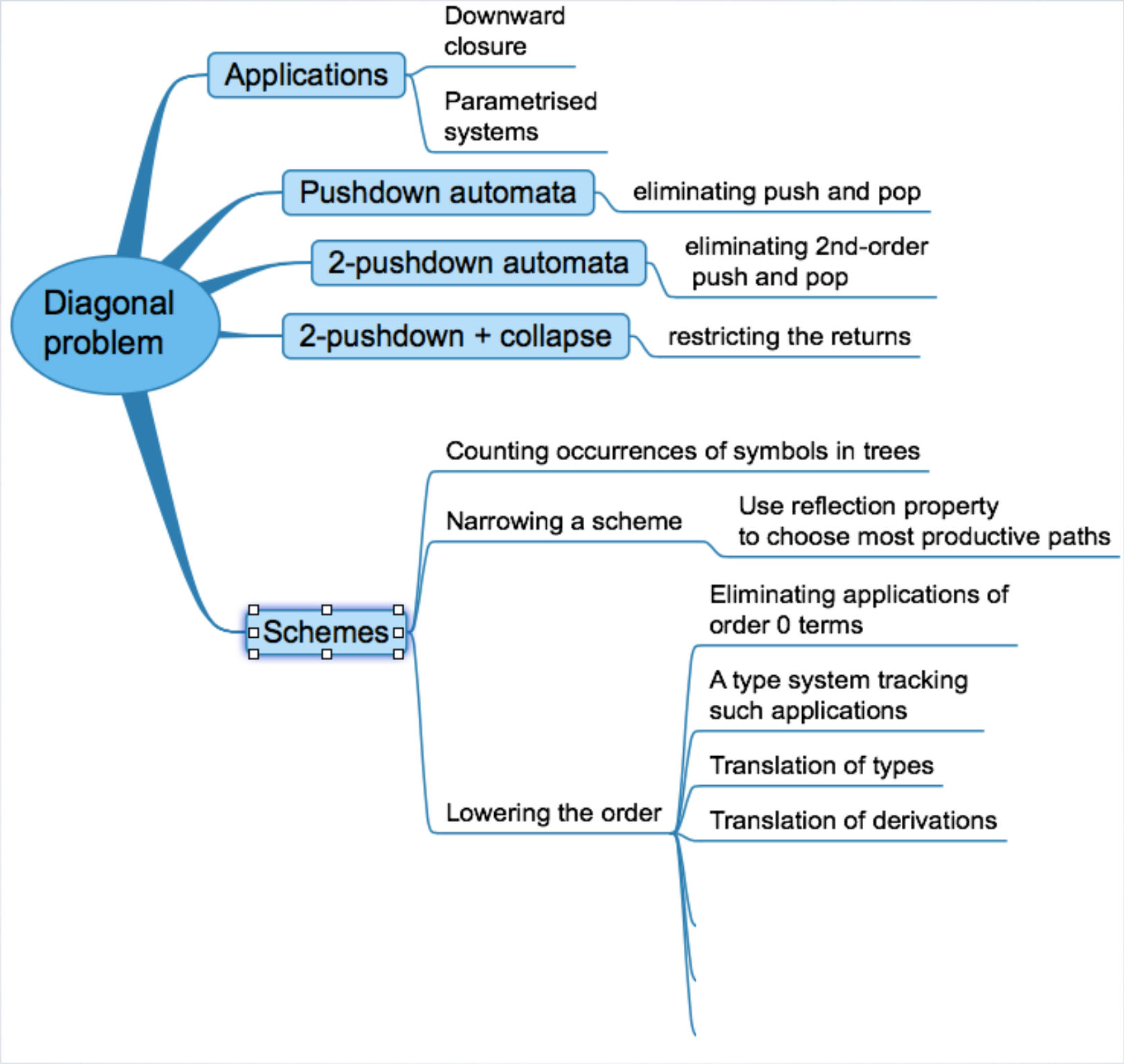
$\mathcal{LT}^\alpha = \mathcal{P}(\Sigma_0) \times \mathcal{T}^\alpha$, if $\alpha = o$ then the first component not empty.

Translation on types

- $tr(\mathbf{r}) = o$,
- if $\tau = (\{(S_1,\tau_1),\ldots,(S_k,\tau_k)\} \to \tau') \in \mathcal{T}^{\alpha\to\beta}$ then

$$tr(\tau) = \begin{cases} tr(\tau_1) \to \cdots \to tr(\tau_k) \to tr(\tau') & \text{if } \alpha \neq o, \\ tr(\tau') & \text{if } \alpha = o. \end{cases}$$

Remark: if $\tau \in \mathcal{T}^\alpha$, then $ord(tr(\tau)) = \max(0, ord(\alpha) - 1)$.


Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes
Counting occurrences of symbols in trees
Narrowing a scheme
Use reflection property to choose most productive paths
Lowering the order
Eliminating applications of order 0 terms
A type system tracking such applications
Translation of types
Translation of derivations

$$\frac{}{\Gamma, x : \lambda \vdash x : \lambda} \qquad \frac{}{\Gamma \vdash A : (\emptyset, \tau)} \qquad \frac{}{\Gamma \vdash a^o : (\{a\}, \mathbf{r})}$$

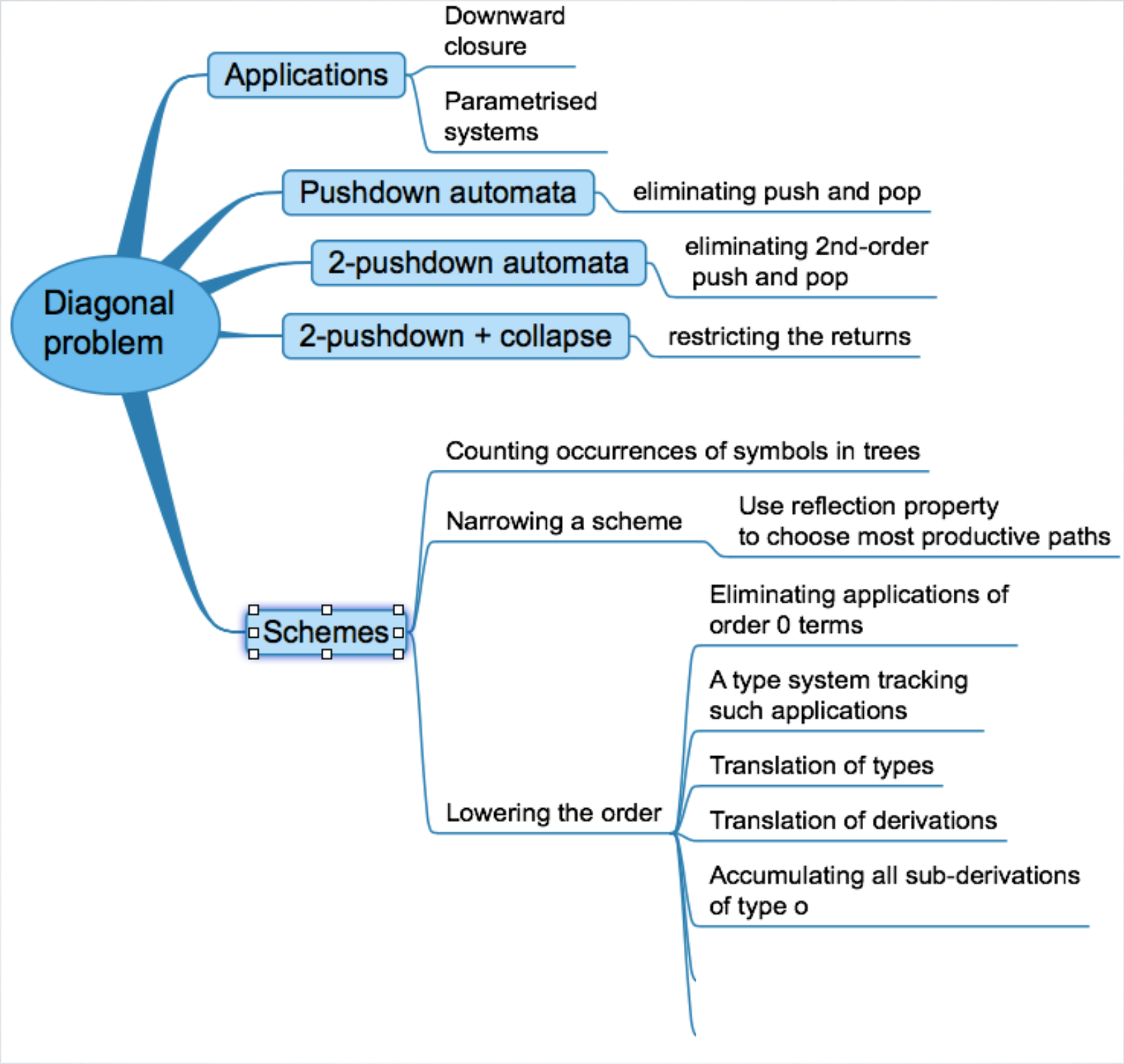
$$\frac{r \geq 1}{\Gamma \vdash a^{o^r \to o} : (\emptyset, \{(S_1, \mathbf{r})\} \to \cdots \to \{(S_r, \mathbf{r})\} \to \mathbf{r})}$$

$$\frac{\Gamma \vdash L : (S_0, \{(S_1, \tau_1), \ldots, (S_k, \tau_k)\} \to \tau) \qquad \Gamma \vdash M : (S_i, \tau_i) \text{ for each } i \in \{1, \ldots, k\}}{\Gamma \vdash L\,M : (S_0 \cup \cdots \cup S_k, \tau)}$$

where $S_0 \cap (S_1 \cup \cdots \cup S_k) = \emptyset$

We define a term $tr(D)$, where D is a derivation for $\Gamma \vdash K : \lambda$,

- If $K = a$ then $tr(D) = a'$.
- If $K = x^\alpha$ then $tr(D) = \top$ if $\alpha = o$, and $tr(D) = x\restriction_\lambda$ otherwise.
- If $K = A$ then $tr(D) = A\restriction_\tau$.
- If $K = L\,M$ then $tr(D) = tr(D_0)$ if $M : o$;
  otherwise $tr(D) = tr(D_0)\, tr(D_1) \ldots tr(D_k)$.


Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes
Counting occurrences of symbols in trees
Narrowing a scheme
Use reflection property to choose most productive paths
Lowering the order
Eliminating applications of order 0 terms
A type system tracking such applications
Translation of types
Translation of derivations
Accumulating all sub-derivations of type o

$$\frac{}{\Gamma, x : \lambda \vdash x : \lambda} \qquad \frac{}{\Gamma \vdash A : (\emptyset, \tau)} \qquad \frac{}{\Gamma \vdash a^{o} : (\{a\}, \mathbf{r})}$$

$$\frac{r \geq 1}{\Gamma \vdash a^{o^{r} \to o} : (\emptyset, \{(S_1, \mathbf{r})\} \to \cdots \to \{(S_r, \mathbf{r})\} \to \mathbf{r})}$$

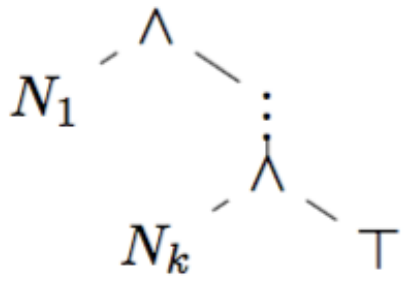
$$\frac{\Gamma \vdash L : (S_0, \{(S_1, \tau_1), \ldots, (S_k, \tau_k)\} \to \tau) \qquad \Gamma \vdash M : (S_i, \tau_i) \text{ for each } i \in \{1, \ldots, k\}}{\Gamma \vdash L\,M : (S_0 \cup \cdots \cup S_k, \tau)}$$

where $S_0 \cap (S_1 \cup \cdots \cup S_k) = \emptyset$

- If $K = L\,M$ then $tr(D) = tr(D_0)$ if $M : o$; otherwise $tr(D) = tr(D_0)\,tr(D_1)\,\ldots\,tr(D_k)$.

$$tr_{cum}(D) = \begin{cases} tr(D); tr_{cum}(D_1); \ldots; tr_{cum}(D_m) & \text{if } \alpha = o, \\ tr_{cum}(D_1); \ldots; tr_{cum}(D_m) & \text{otherwise.} \end{cases}$$

$merge(N_1; \ldots; N_k)$ is



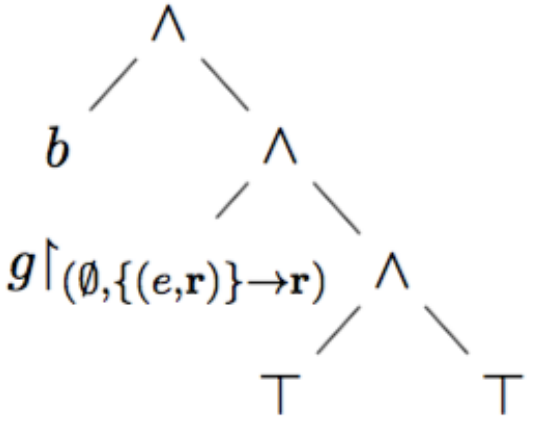
Example b(gx): $\Gamma \equiv x : (e, \rho), g : (\emptyset, \{(e, \rho)\} \to \rho)$

$$\dfrac{\Gamma \vdash b : (\emptyset, \{(e, \mathbf{r})\} \to \mathbf{r}) \qquad \dfrac{\Gamma \vdash g : (\emptyset, \{(e, \mathbf{r})\} \to \mathbf{r}) \qquad \Gamma \vdash x : (e, \mathbf{r})}{\Gamma \vdash g\,x : (e, \mathbf{r})}}{\Gamma \vdash b\,(g\,x) : (e, \mathbf{r})}$$

We have:

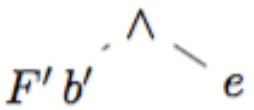
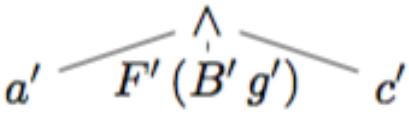
$tr(D) = b'$ $\qquad$ $tr_{cum}(D) = b'; g\restriction_{(\emptyset, \{(e,\mathbf{r})\} \to \mathbf{r})}; \top$

$merge(tr_{cum}(D)) =$

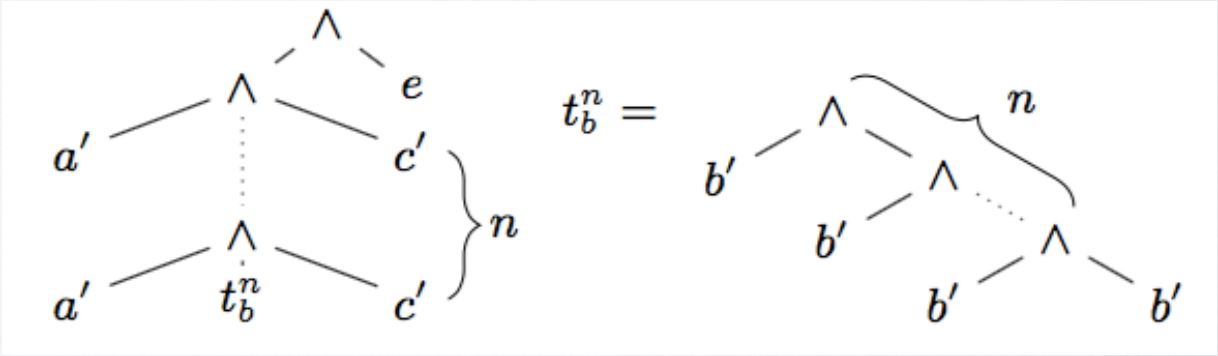


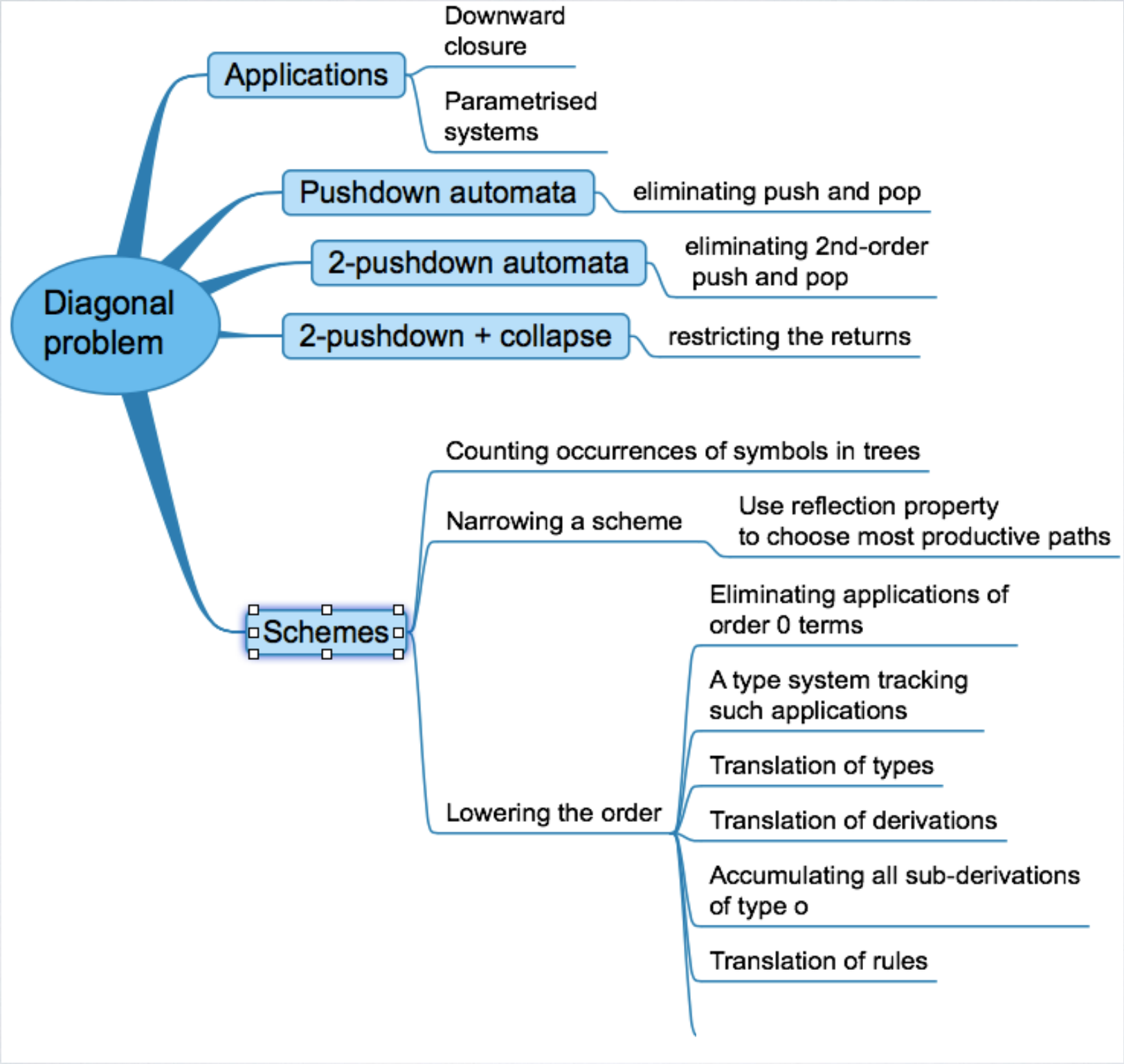
Example 2

Translation:

$S \to F\ e$	$S' \to$ 
$F\ g\ x \to g\ x$	$F'\ g' \to g'$
$F\ g\ x \to a\ (F\ (B\ g)\ (c\ x))$	$F'\ g' \to$ 
$B\ g\ x \to b\ (g\ x)$	$B'\ g' \to$ 

$a^n b^{n+1} c^n e$




Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes
Counting occurrences of symbols in trees
Narrowing a scheme
Use reflection property to choose most productive paths
Lowering the order
Eliminating applications of order 0 terms
A type system tracking such applications
Translation of types
Translation of derivations
Accumulating all sub-derivations of type o
Translation of rules

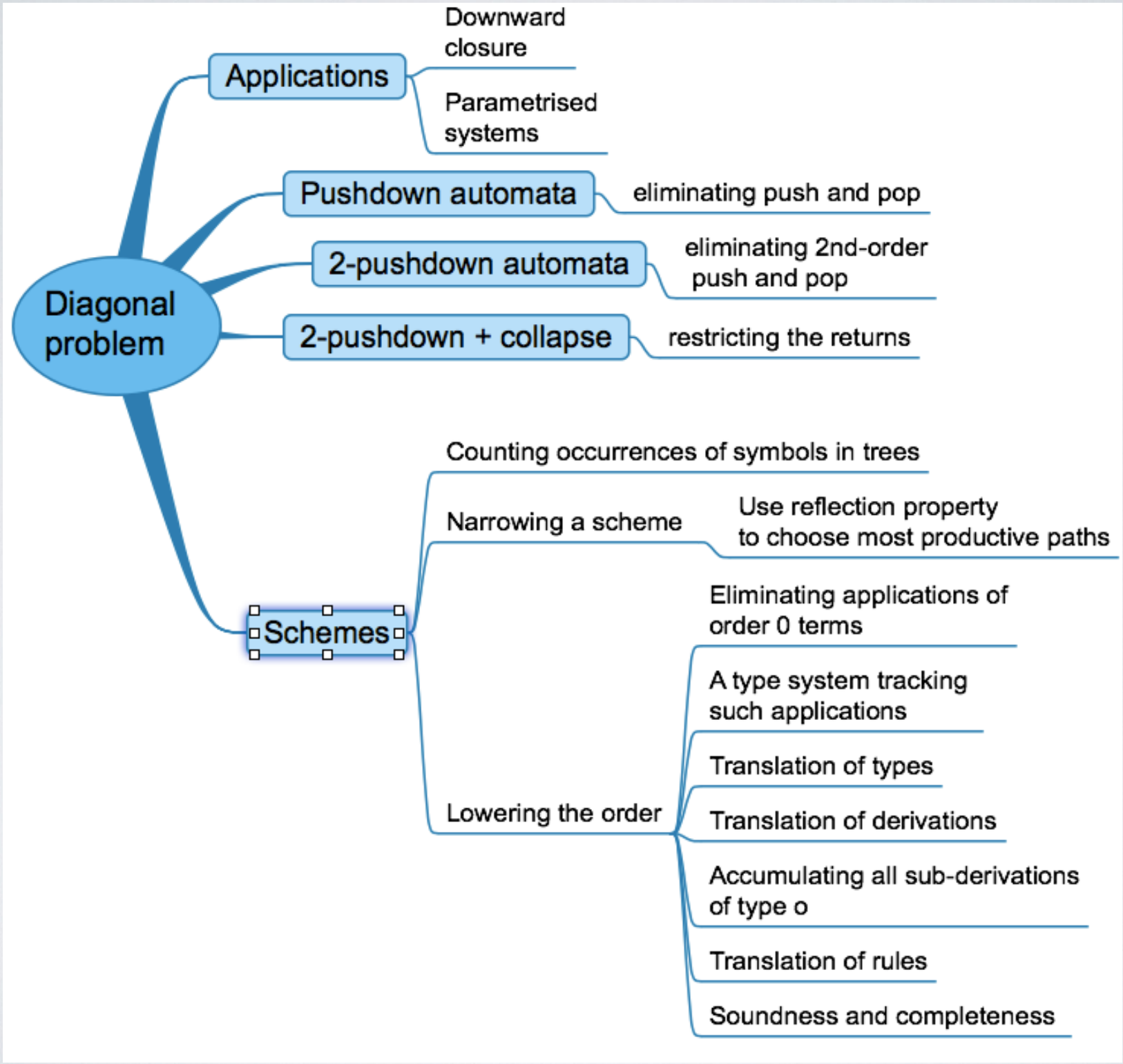
Translation of rules of a scheme

Take $Axy \to K$ where $x : o,\ y : \alpha$

For every pair of types $\lambda_x = (S_x, r),\ \lambda_y = (S_y, \tau)$

and a derivation D $x : \lambda_x, y : \lambda_y \vdash K : (S_x \cup S_y, r)$

we add a rule $A\ y\restriction_{\lambda_y} \to merge(tr_{cum}(D))$


Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes
Counting occurrences of symbols in trees
Narrowing a scheme
Use reflection property to choose most productive paths
Lowering the order
Eliminating applications of order 0 terms
A type system tracking such applications
Translation of types
Translation of derivations
Accumulating all sub-derivations of type o
Translation of rules
Soundness and completeness

Soundness

For every tree generated by Sch' there is an equivalent tree generated by Sch.

Reduction step:

Given a derivation D for $\vdash L : (S, \mathbf{r})$

and a reduction step $merge(tr_{cum}(D)) \rightarrow^{lf}_{\mathcal{S}'} P$

we find $L \rightarrow_{\mathcal{S}} L'$

and a derivation D' for $\vdash L' : (S, \mathbf{r})$

with $merge(tr_{cum}(D')) \simeq P$

Base case:

If $merge(tr_{cum}(D))$ is a tree then it is equivalent to L.

Completeness

For every tree generated by Sch there is an equivalent tree generated by Sch'.

Backwards reduction step:

Given a derivation D' for $\vdash L' : (S, \mathbf{r})$

and a reduction step $L \to_{\mathcal{S}} L'$

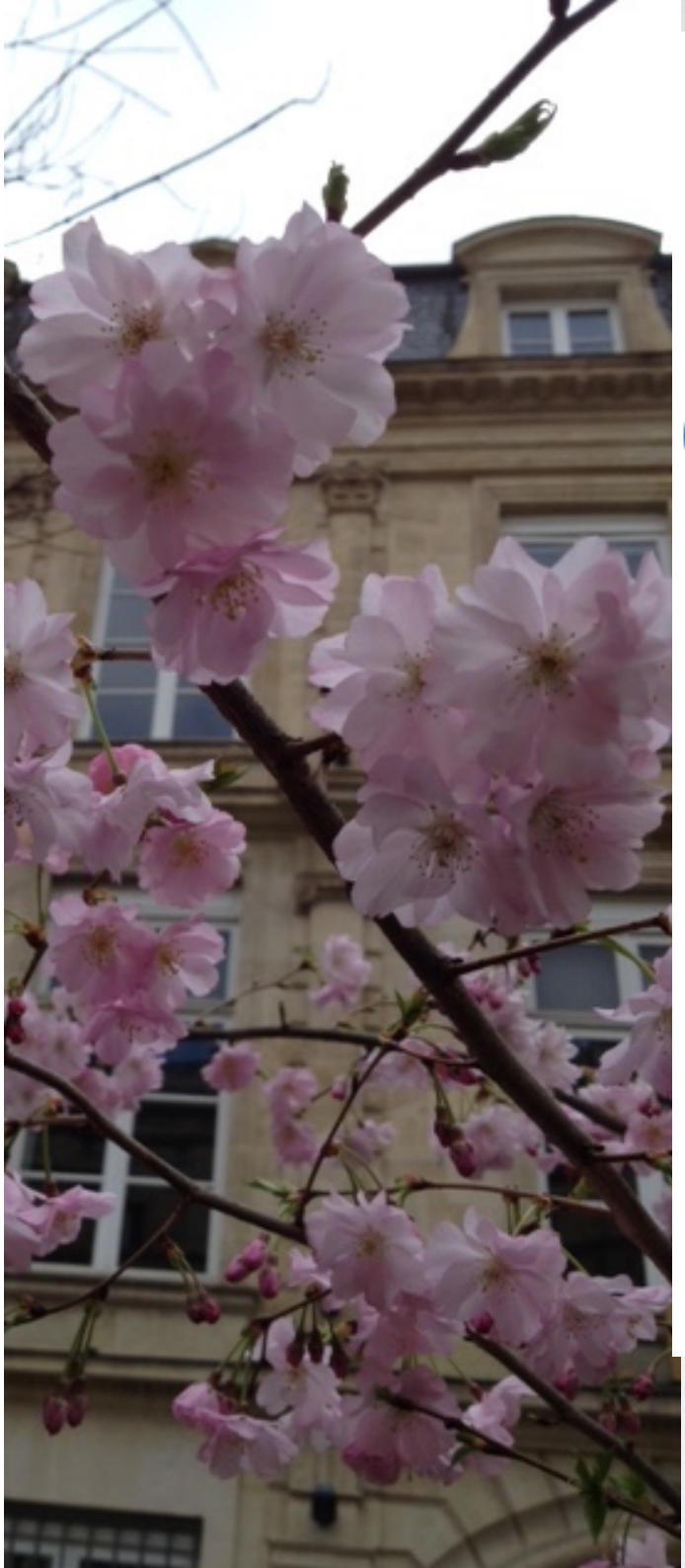
we find a reduction

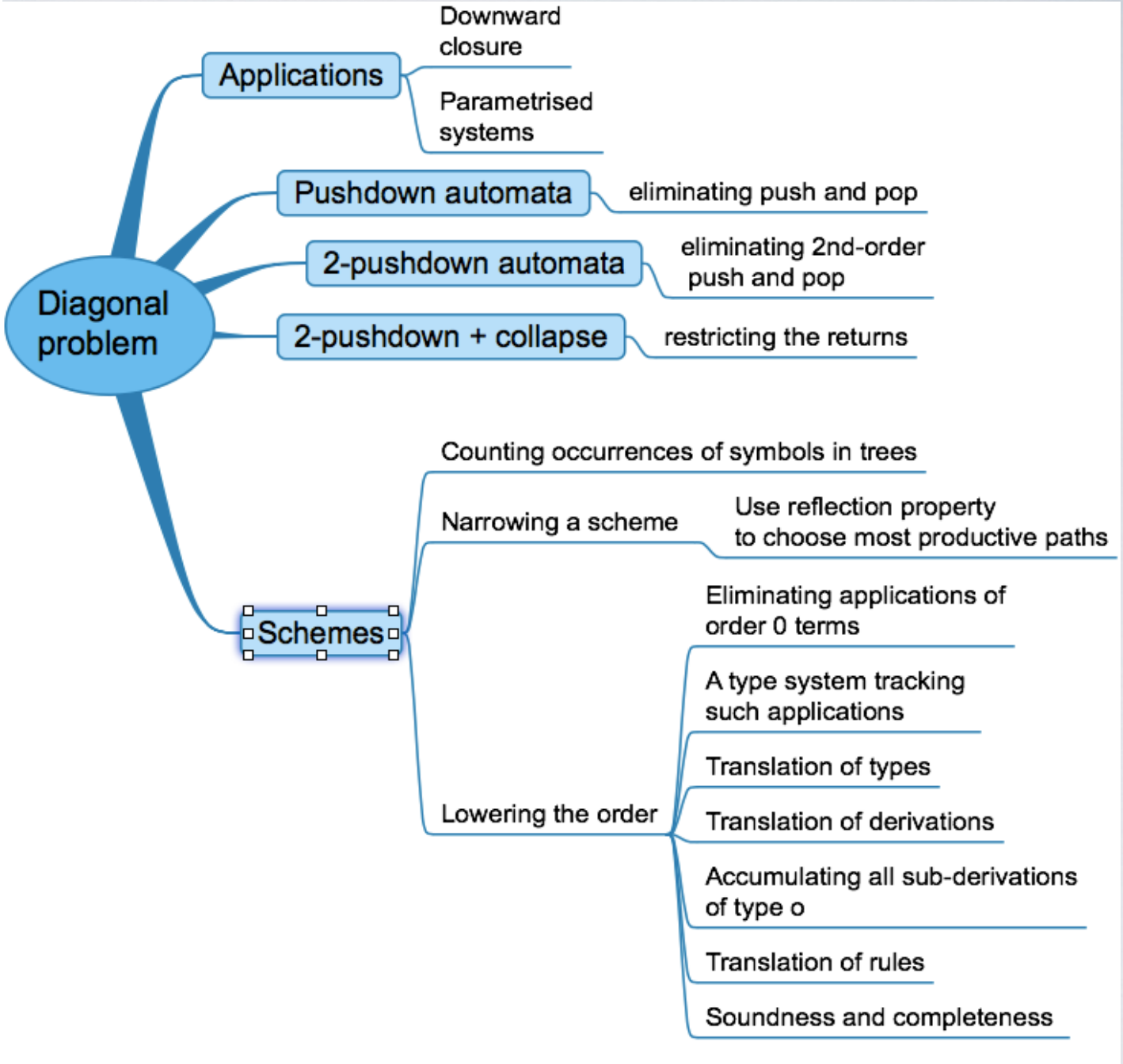
$$merge(tr_{cum}(D)) \to_{\mathcal{S}'} P \simeq merge(tr_{cum}(D'))$$

for some derivation D for $\vdash L : (S, \mathbf{r})$

Base case:

If K is a S-narrow tree then there is D for $\vdash K : (S, \mathbf{r})$ such that $merge(tr_{cum}(D)) \simeq K$.




Diagonal problem
Applications
Downward closure
Parametrised systems
Pushdown automata
eliminating push and pop
2-pushdown automata
eliminating 2nd-order push and pop
2-pushdown + collapse
restricting the returns
Schemes
Counting occurrences of symbols in trees
Narrowing a scheme
Use reflection property to choose most productive paths
Lowering the order
Eliminating applications of order 0 terms
A type system tracking such applications
Translation of types
Translation of derivations
Accumulating all sub-derivations of type o
Translation of rules
Soundness and completeness