KU LEUVEN

kProlog

an algebraic Prolog for kernel programming

Francesco Orsini
Paolo Frasconi
Luc De Raedt

DTAI
Declarative Languages & Artificial Intelligence

Outline

- Motivation
- kProlog^{S}
  - Algebraic T_P-operator
  - Tensor operations
- kProlog
  - Algebraic T_P-operator with meta-functions
  - Cyclic programs
- kProlog $^{S[\mathbf{x}]}$
  - Graph kernels
- Conclusions

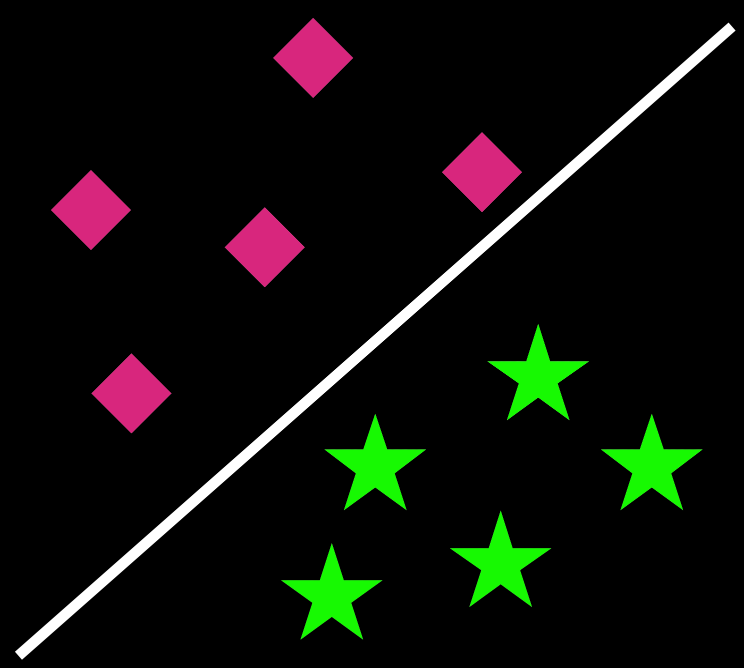
Outline

- Motivation
- kProlog^{S}
  - Algebraic T_P-operator
  - Tensor operations
- kProlog
  - Algebraic T_P-operator with meta-functions
  - Cyclic programs
- kProlog $^{S[\mathbf{x}]}$
  - Graph kernels
- Conclusions

Motivation

We want to design an algebraic Prolog for learning with linear separators.

Linear separators



Prolog

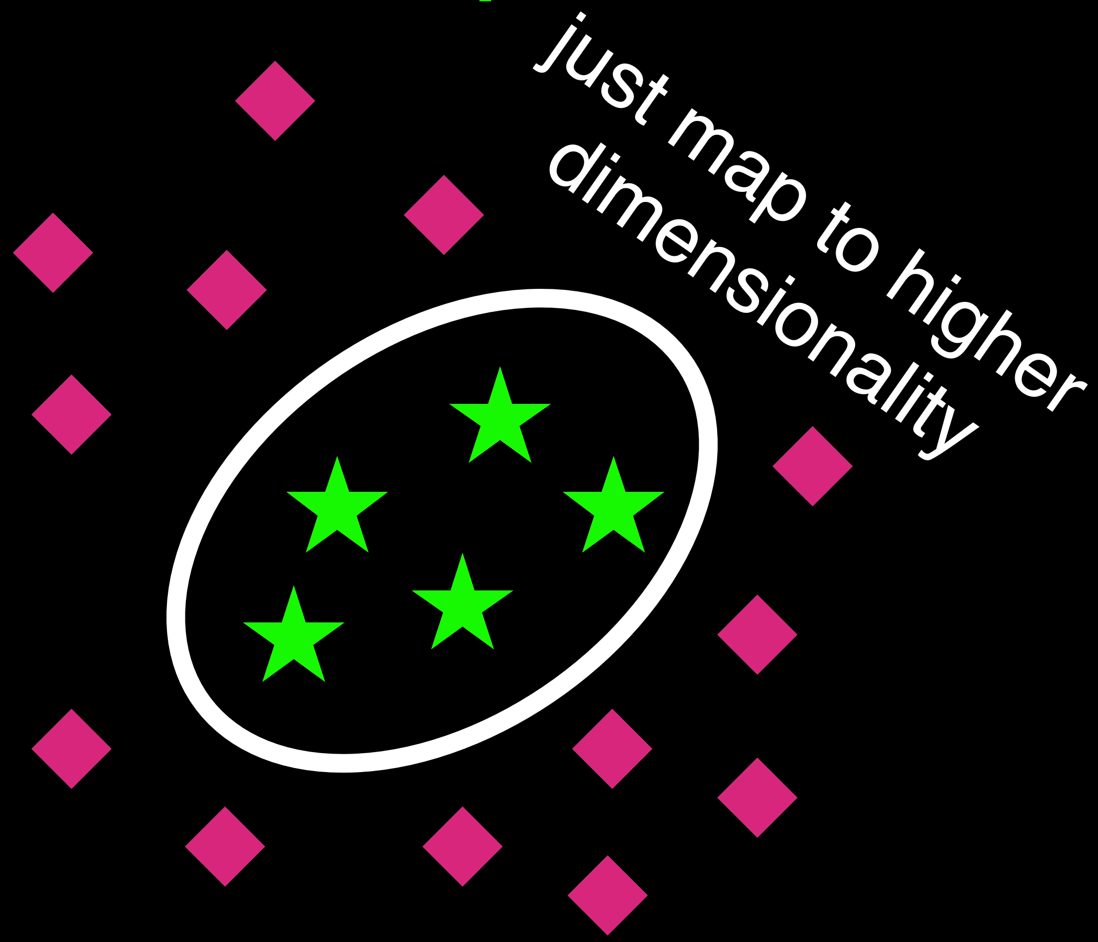
?-

Which Prolog program should I write?

Motivation

We want to design an algebraic Prolog for learning with linear separators.

Linear separators



Prolog

?-

Which Prolog program should I write?

Motivation

	Probabilistic programming	Learning with kernels	Kernel programming	Short description
ProbLog & PRISM	✓	✗	✗	facts are labeled and **labels** are **combined** using **logic**
kLog	✗	✓	✗	grounds **logic** to a **graph**, calls an external **graph kernel**
kProlog	✓	✓	✓	fact **labels** capture the **kernel**, **logic** allows to program the **kernel**

Outline

- Motivation
- kProlog^{S}
  - Algebraic T_P-operator
  - Tensor operations
- kProlog
  - Algebraic T_P-operator with meta-functions
  - Cyclic programs
- kProlog$^{S[\mathbf{x}]}$
  - Graph kernels
- Conclusions

kProlog^{S}

In kProlog^{S} facts are labeled with semiring elements.

kProlog^{S}

In kProlog^{S} facts are labeled with semiring elements.

Sounds like algebraic ProbLog without disjoint sums.

kProlog^S

A kProlog^S program P is a 4-tuple (F, R, S, ℓ) where:

- F is a finite set of facts,
- R is a finite set of definite clauses (also called rules),
- S is a semiring with sum $\oplus$ and product $\otimes$ operations, whose neutral elements are 0_S and 1_S respectively.
- $\ell : F \to S$ is a function that maps facts to semiring values.

Algebraic interpretation

An algebraic interpretation $I_w = (I, w)$
of a ground kProlog^S program $P = (F, R, S, \ell)$
is a set of tuples $(a, w(a))$
where:

- a is an atom in the Herbrand base A
- $w(a)$ is an algebraic formula over the fact labels $\{\ell(f) | f \in F\}$.

Algebraic T_P-operator

Let $P = (F, R, S, \ell)$ be a ground algebraic logic program with Herbrand base A.
Let $I_w = (I, w)$ be an algebraic interpretation with pairs $(a, w(a))$.
Then the $T_{(P,S)}$-operator is $T_{(P,S)}(I_w) = \{(a, w'(a)) | a \in A\}$ where:

$$w'(a) = \begin{cases} \ell(a) & \text{if } a \in F \\ \bigoplus_{\substack{\{b_1,\dots,b_n\} \subseteq I \\ a :- b_1,\dots,b_n}} \bigotimes_{i=1}^{n} w(b_i) & \text{if } a \in A \setminus F \end{cases}.$$

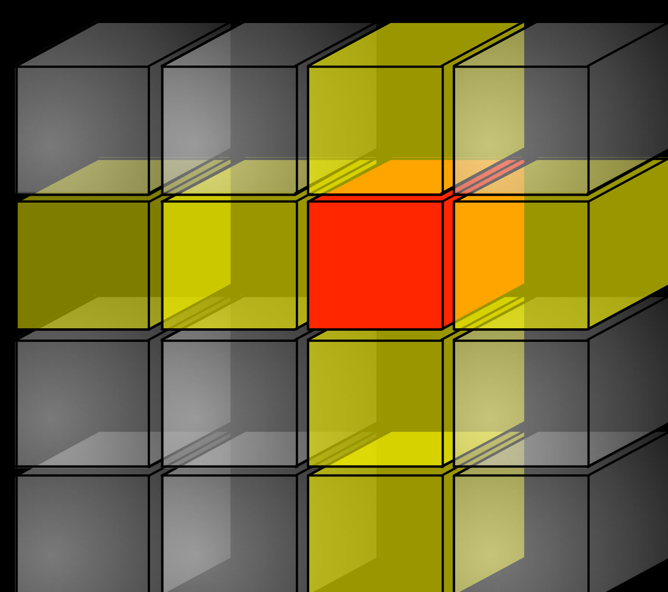
Algebraic T_P-operator

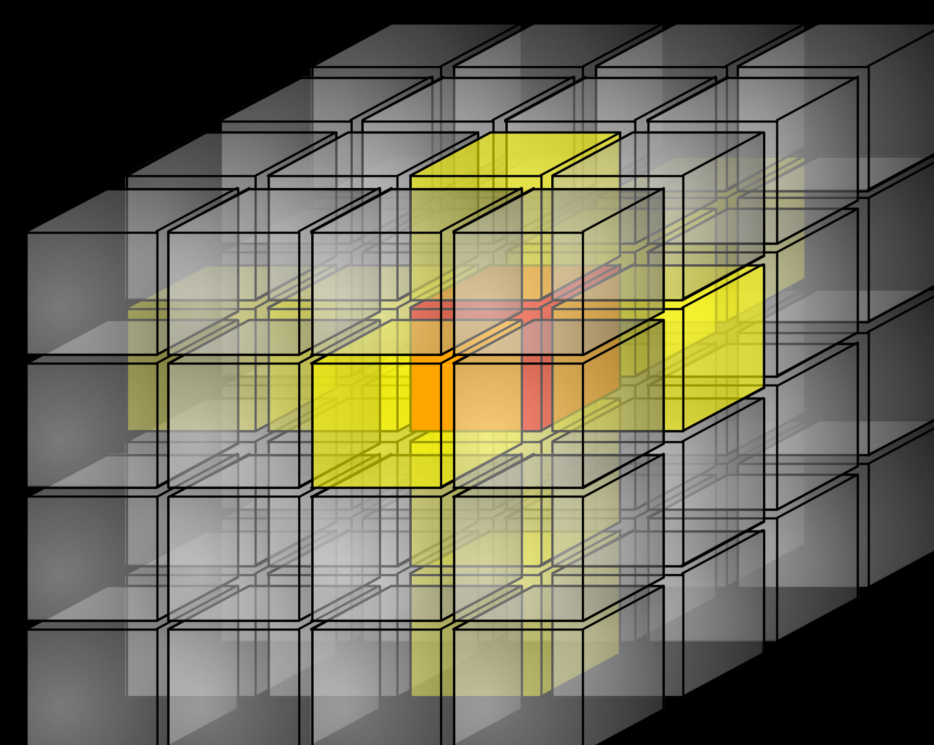
logic	T_P-operator	example w(a) = 0.5 w(b) = 0.3 w(c) = 0.9
a :- a, b.	w(a) $\otimes$ w(b)	0.5 x 0.3
	$\oplus$	+
a :- c.	w(c)	0.9
	=	=
	$T_P(\{ a \})$	1.05

Tensor operations

- n-ary predicate a/n represents n-mode tensor.
- a ground atom $a(d_1, ...d_n)$ represents n-mode tensor.
- $d_1, \ldots, d_n$ are elements of the Herbrand universe and are the indices that identify a cell.

Vectors and matrices are particular cases of 1-mode and 2-mode tensors respectively.





Tensor operations

algebra	**kProlog**

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$$	`:- declare(a/2, int).`
	
	`1::a(0, 0).`
	`2::a(0, 1).`
	`3::a(1, 1).`

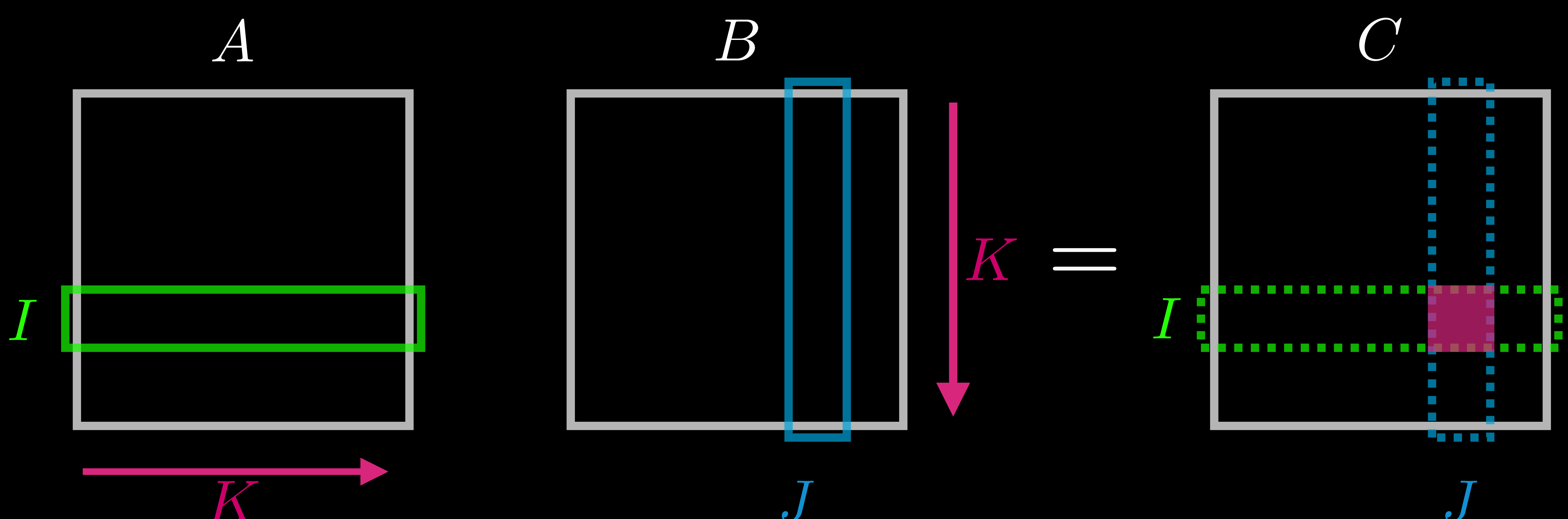
$$B = \begin{bmatrix} 2 & 1 \\ 5 & 1 \end{bmatrix}$$	`:- declare(b/2, int).`
	
	`2::b(0, 0).`
	`1::b(0, 1).`
	`5::b(1, 0).`
	`1::b(1, 1).`

Tensor operations

algebra	kProlog	example
transpose A^t	`c(I, J) :- a(J, I).`	$\begin{bmatrix}1 & 2\\ 0 & 3\end{bmatrix}^t = \begin{bmatrix}1 & 0\\ 2 & 3\end{bmatrix}$
sum $A + B$	`c(I, J) :- a(I, J).` `c(I, J) :- b(I, J).`	$\begin{bmatrix}1 & 2\\ 0 & 3\end{bmatrix} + \begin{bmatrix}2 & 1\\ 5 & 1\end{bmatrix} = \begin{bmatrix}3 & 3\\ 5 & 4\end{bmatrix}$
Hadamard product (element-wise product) $A \odot B$	`c(I, J) :- a(I, J), b(I, J).`	$\begin{bmatrix}1 & 2\\ 0 & 3\end{bmatrix} \odot \begin{bmatrix}2 & 1\\ 5 & 1\end{bmatrix} = \begin{bmatrix}2 & 2\\ 0 & 3\end{bmatrix}$

Tensor operations

algebra	kProlog	example
matrix product AB	`c(I, J) :- a(I, K), b(K, J).`	$\begin{bmatrix}1&2\\0&3\end{bmatrix}\begin{bmatrix}2&1\\5&1\end{bmatrix}=\begin{bmatrix}12&3\\15&3\end{bmatrix}$



Tensor operations

algebra	kProlog
Kronecker product $A \otimes B$	`c(i(Ia, Ib), j(Ja, Jb)):- a(Ia, Ja), b(Ib, Jb).`

The indices of the result are compound terms.
Tensor relational algebra lacks of functions, so the Kronecker product can not be naturally.

example

$$\overset{A}{\begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}} \otimes \overset{B}{\begin{bmatrix} 2 & 1 \\ 5 & 1 \end{bmatrix}} = \begin{bmatrix} 1\begin{bmatrix} 2 & 1 \\ 5 & 1 \end{bmatrix} & 2\begin{bmatrix} 2 & 1 \\ 5 & 1 \end{bmatrix} \\ 0\begin{bmatrix} 2 & 1 \\ 5 & 1 \end{bmatrix} & 3\begin{bmatrix} 2 & 1 \\ 5 & 1 \end{bmatrix} \end{bmatrix} = \overset{C}{\begin{bmatrix} 2 & 1 & 4 & 2 \\ 5 & 1 & 10 & 2 \\ 0 & 0 & 6 & 3 \\ 0 & 0 & 15 & 3 \end{bmatrix}}$$

Outline

- Motivation
- kProlog^{S}
  - Algebraic T_P-operator
  - Tensor operations
- kProlog
  - Algebraic T_P-operator with meta-functions
  - Cyclic programs
- kProlog $^{S[\mathbf{x}]}$
  - Graph kernels
- Conclusions

kProlog

kProlog overcomes the limitations of kProlog^{S}.

We introduce:

- multiple semirings in the same program,
- meta-functions and meta-clauses to overcome the limits imposed by the semiring sum $\oplus$ and product $\otimes$ operations.

kProlog: meta-functions

A meta-function `m`: $S_1 \times \ldots \times S_m \rightarrow S'$ is a function that maps m semiring values $x_i \in S_i,\ i = 1, \ldots, k$ to a value of type S', where $S_1, \ldots, S_k$ and S' can be distinct semirings.

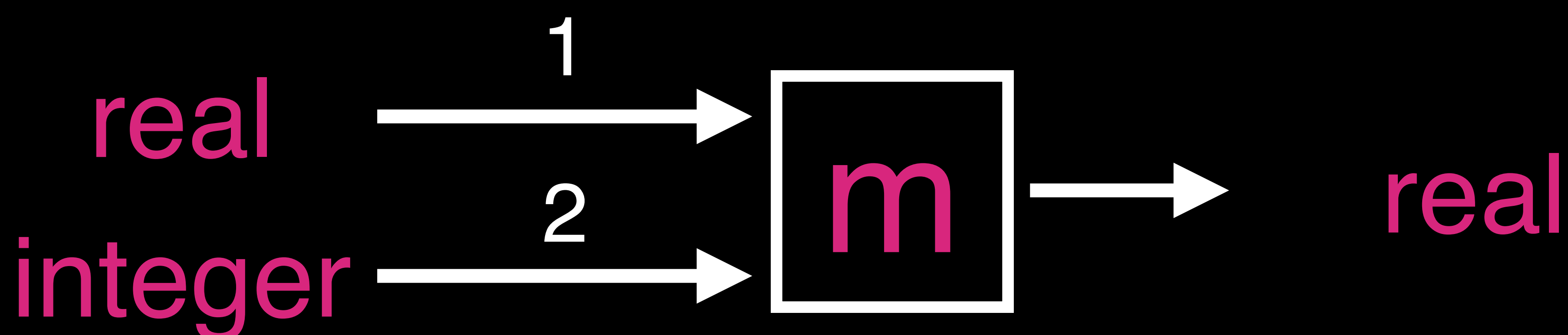
Let `a_1,...,a_k` be algebraic atoms, the syntax

```
@m[a_1,...,a_k]
```

expresses that the meta-function `@m` is applied to the semiring values of the atoms `a_1,...,a_k`.

kProlog: meta-functions

$$m : \mathbb{R} \times \mathbb{Z} \to \mathbb{R}$$



$$m : (x, y) \mapsto ysin(x)$$

kProlog: meta-clauses

In the kProlog language a meta-clause

```
h :- b_1,...,b_n.
```

is a universally quantified expression where:

- `h` is an atom
- `b_1,...,b_n` can be either:
  - body atoms or
  - meta-functions applied to other algebraic atoms.

For a given meta-clause, if the head is labeled with the semiring S, also the labels of the body atoms and the return types of the meta-functions must be on the semiring S.

kProlog: meta-clauses

```
a :- a,b.
a :- @sin[c].
```

meta-clause

kProlog: program

A kProlog program P is a union of kProlog^{S_i} programs and meta-clauses.

Algebraic T_P-operator with meta-clauses

Let P be meta-transformed kProlog program with facts F and atoms A.
Let $I_w = (I, w)$ be an algebraic interpretation with pairs $(a, w(a))$.
Then the T_P-operator is $T_P(I_w) = \{(a, w'(a)) | a \in A\}$ where:

$$w'(a) = \begin{cases} \ell(a) & \text{if } a \in F \\ w'_{\text{CLAUSE}}(a) \bigoplus w'_{\text{META}}(a) & \text{if } a \in A \setminus F \end{cases}$$

$$w'_{\text{CLAUSE}}(a) = \bigoplus_{\substack{\{b_1,\ldots,b_n\} \subseteq I \\ a :- b_1,\ldots,b_n}} \bigotimes_{i=1}^{n} w(b_i)$$

The same as in kProlog^S

Contribute from the meta-functions.

$$w'_{\text{META}}(a) = \bigoplus_{\substack{\{b_1,\ldots,b_k\} \subseteq I \\ a :- @m[b_1,\ldots,b_k]}} m(w(b_1), \ldots, w(b_k))$$

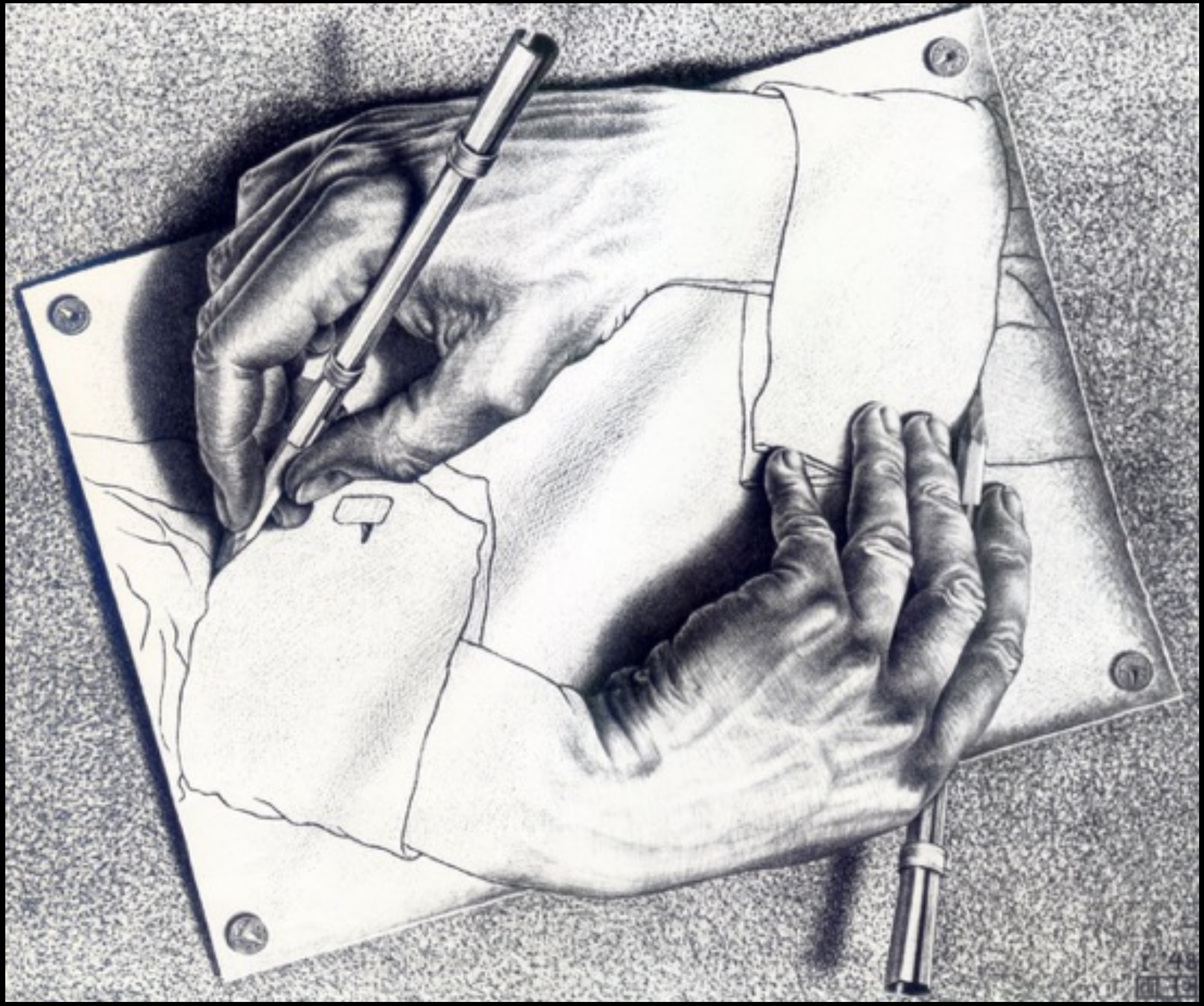
Algebraic T_P-operator with meta-clauses

logic	T_P-operator	example $w(a) = 0.5$ $w(b) = 0.3$ $w(c) = 0.9$
a:− a , b .	$w(a) \otimes w(b)$	0.5 x 0.3
	$\oplus$	+
a:− @sin [c] .	$\sin(w(c))$	sin(0.9) = 0.78…
	$=$	=
	$T_P(\{ a \})$	0.93…

Outline

- Motivation
- kProlog^S
  - Algebraic T_P-operator
  - Tensor operations
- kProlog
  - Algebraic T_P-operator with meta-functions
  - Cyclic programs
- kProlog $^{S[\mathbf{x}]}$
  - Graph kernels
- Conclusions

Cyclic programs



Evaluation of kProlog programs

see also [Whaley & al. 2015]

- meta-transformation ($P_0 \rightsquigarrow P$)
- grounding ($P \rightsquigarrow ground(P)$)
- partitioning in strata

  ($ground(P) \rightsquigarrow \{P_1, \ldots, P_n\}$ where $ground(P) = \bigcup_{i=1}^{n} P_i$)
- visit the strata sequentially $P_1, \ldots, P_n$:
- for reach stratum P_i:
  - if is acyclic apply the algebraic T_P-operator once.
  - if is cyclic apply the algebraic T_P-operator:
    - for the acyclic rules only once.
    - for the cyclic rules until convergence of the weights.

```
:- declare(<pred.>/<n>, <sem.>).
vs
:- declare(<pred.>/<n>, <sem.>, <update-type>).
```

additive
vs
destructive
updates

Evaluation of kProlog programs

FOR LANGUAGE LAWYERS

```
P_1, ..., P_n = scc(ground(P)) // find the strongly connected components
                               // in the ground program
π = topsort(STRATA) // find a permutation that sorts
                    // the strata in topological order
for f in F
  // initialise w(f) to the
  // weight of the fact f
end

for i in π
  w(a) := 0_s ∀ a ∈ P_i \ F
  w_old := w
  for rule in NREC(P_i)
    h = head(rule)
    w(h) := w(h) + T_(P_i, w_old)(rule)
  end
  w_old := w
  while w_old != w
    Δw(head(rule)) = 0_s ∀ rule ∈ REC(S)
    for rule in REC(P_i)
      h = head(rule)
      Δw(h) += T_(P_i, w_old)(rule)
    end

    for rule in REC(S)
      if rule is additive
        w(head(rule)) := w_old(head(rule)) + Δw(head(rule))
      else // rule is destructive
        w(head(rule)) := Δw(head(rule))
      end
    end
  end
end
```

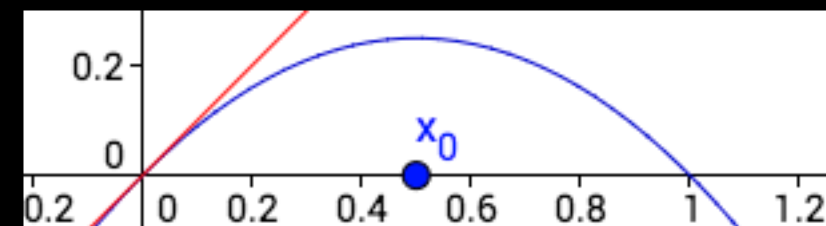
:- declare(<pred.>/<n>, <sem.>).
vs
:- declare(<pred.>/<n>, <sem.>, <update-type>).

additive
vs
destructive
updates

Cyclic programs

meta-function definition

$$g : \mathbb{R} \to \mathbb{R} \quad g(x) = x(1-x)$$

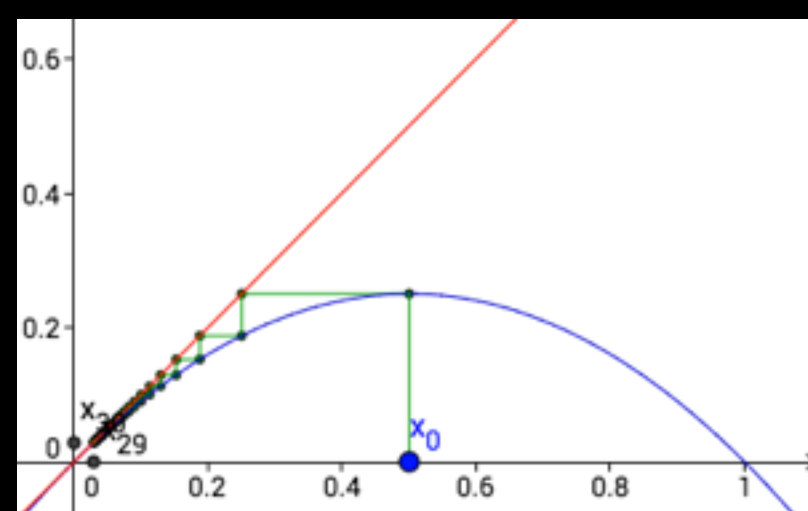


we want to compute:

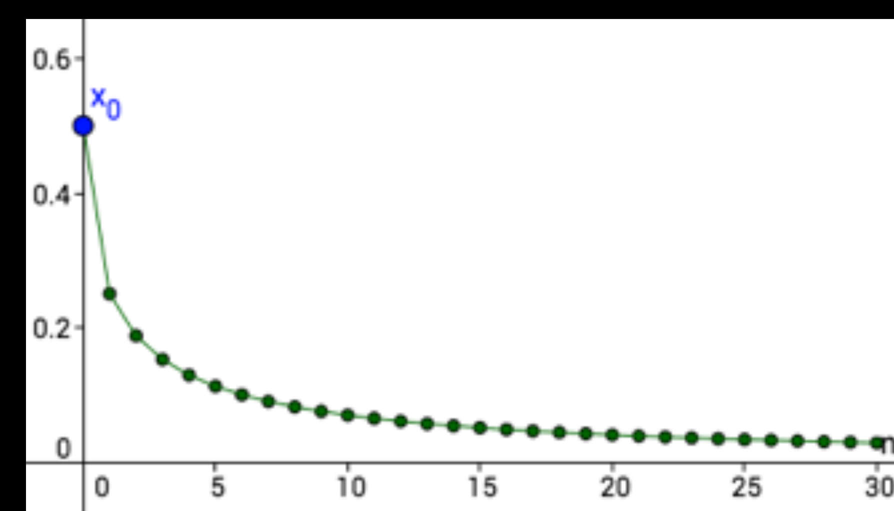
$$\lim_{n \to \infty} g^n(x_0), \text{ where } x_0 = 0.5$$

$$g^0(x_0) = x_0$$
$$g^1(x_0) = g(x_0)$$
$$g^2(x_0) = g(g(x_0))$$
$$\vdots$$
$$g^n(x_0) = \underbrace{(g \odot \ldots \odot g)}_{\substack{\text{function}\\ \text{composition}\\ n \text{ times}}}(x_0)$$

Cobweb Plot



Solution



images generated with:
http://mathinsight.org/applet/function_iteration_cobweb_combined

Cyclic programs

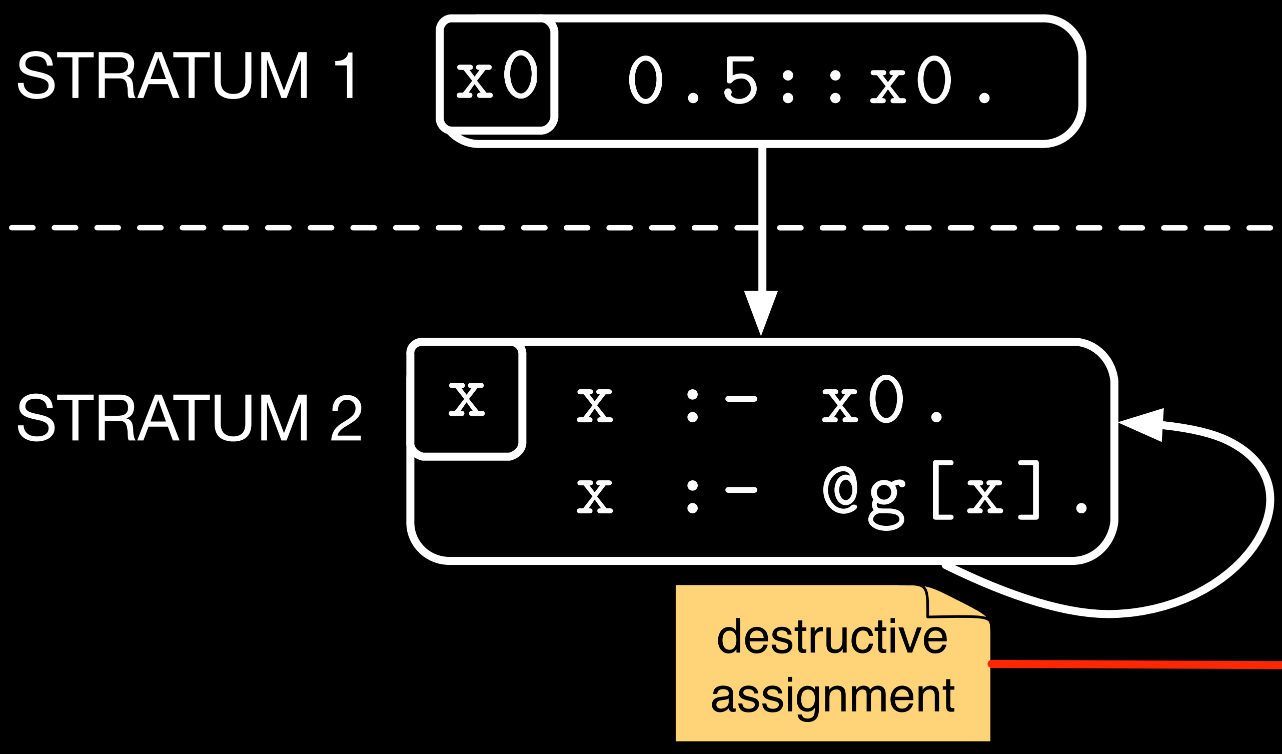
```
:- declare(x, real, destructive).
:- declare(x0, real).

0.5::x0.
x :- x0.
x :- @g[x].
```

destructive assignment

meta-function definition

$g : \mathbb{R} \to \mathbb{R}$

$g(x) = x(1 - x)$



w(x0) := 0.5

w(x) := w(x0)
$w^{old} := w$
while w(x) != w^{old}(x)
  Δw(x) := g(w^{old}(x))
  w(x) := Δ w(x)
end

Outline

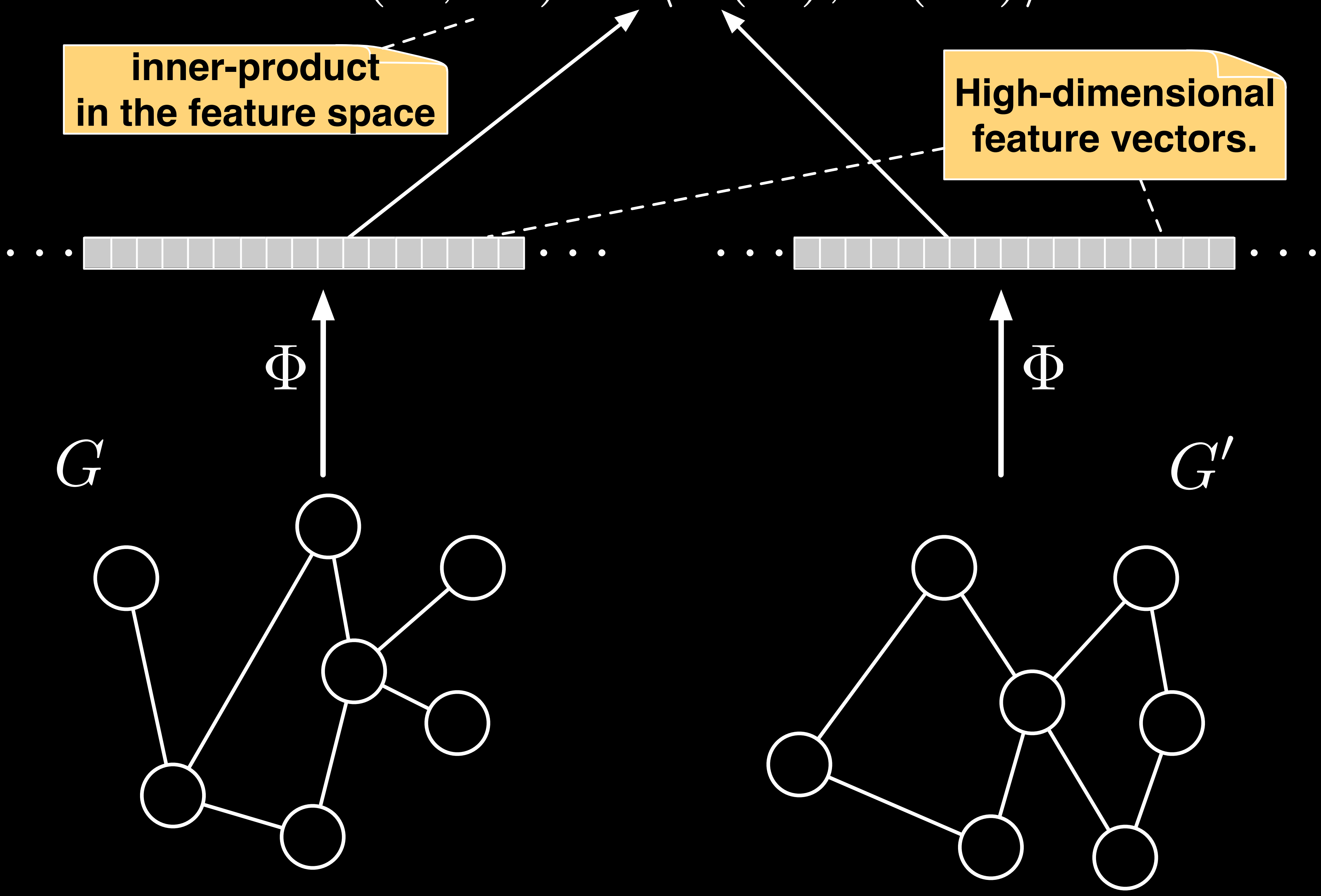
- Motivation
- kProlog^S
  - Algebraic T_P-operator
  - Tensor operations
- kProlog
  - Algebraic T_P-operator with meta-functions
  - Cyclic programs
- kProlog $^{S[\mathbf{x}]}$
  - Graph kernels
- Conclusions

kProlog$^{S}[\mathbf{x}]$

multivariate polynomials
for
feature extraction

Graph kernels

$$k(G, G') = \langle \Phi(G), \Phi(G') \rangle$$



Representing the structure of a machine learning problem

framework	domain structure	machine learning
conv. kernels on discrete data structures	graph	kernel
kProlog	logic program meta-clauses	algebraic labels meta-functions

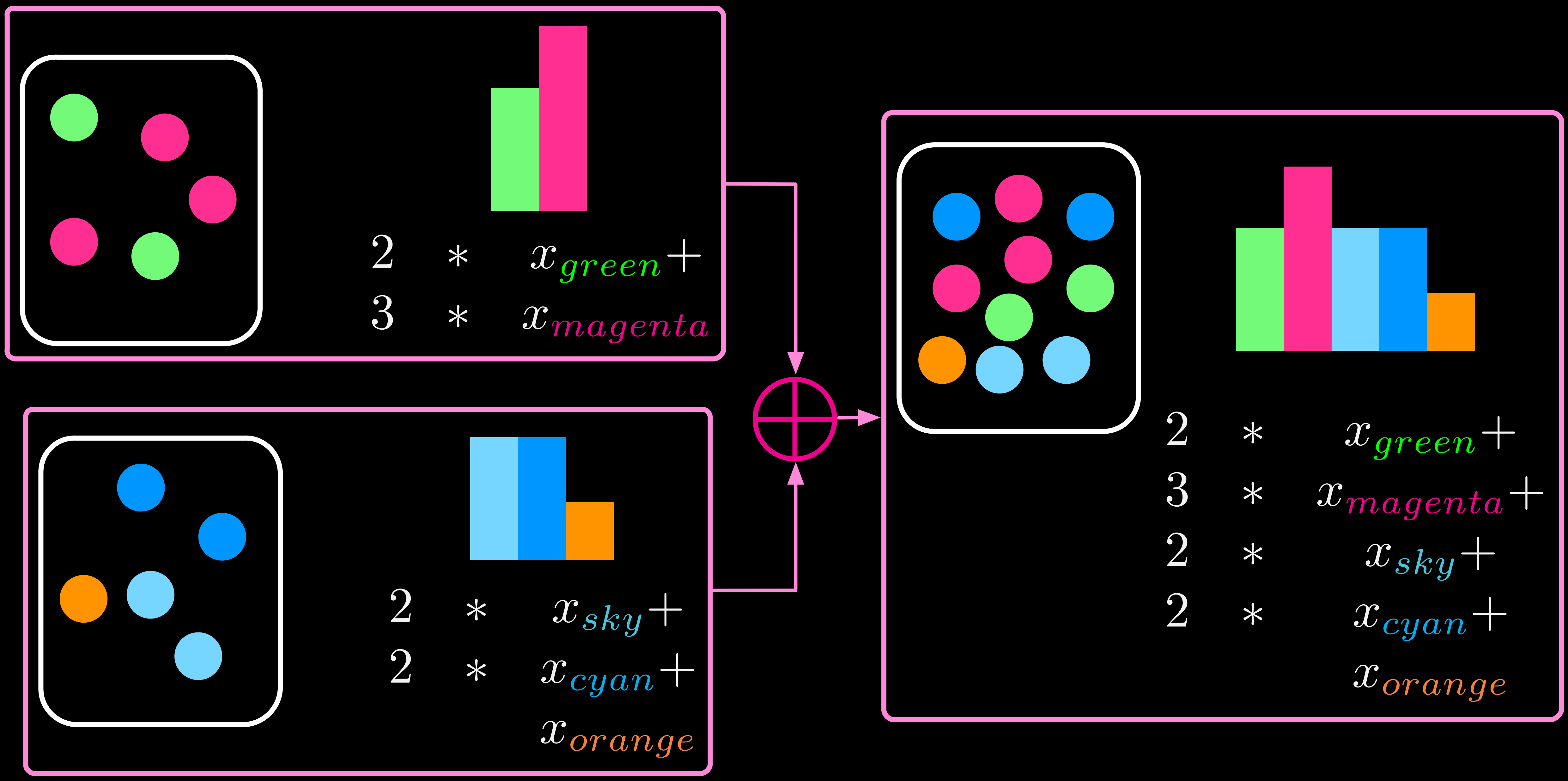
kProlog$^{S[\mathbf{x}]}$

some relevant operations

sum	$\bigoplus$
compress	$@id$
dot product	$@dot$

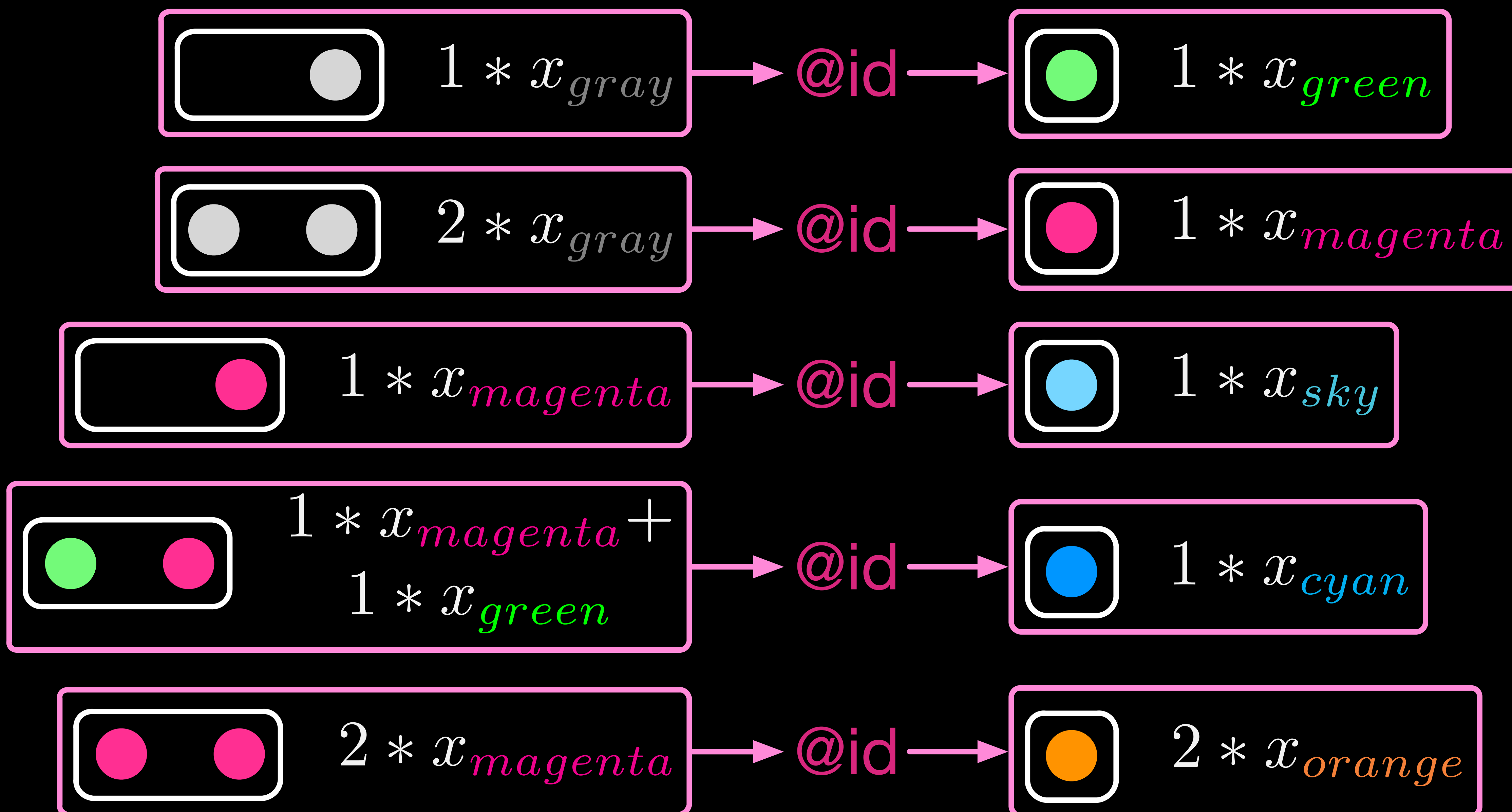
kProlog$^{S[\mathbf{x}]}$

semiring sum = feature addition



kProlog$^{S[\mathbf{x}]}$

@id function = feature compression



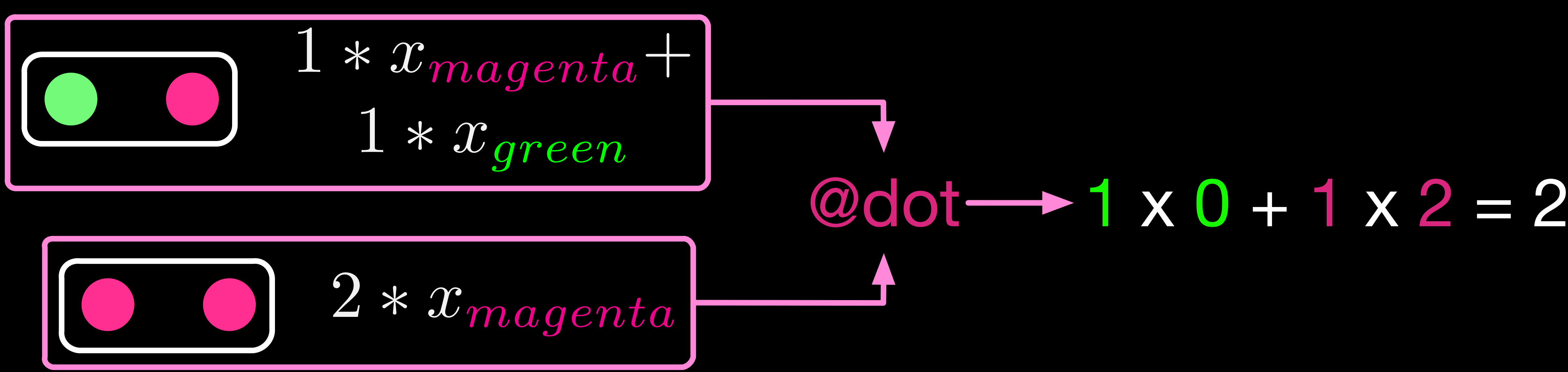
analogous of the f function in [Shervashidze et al. (2011)]

kProlog$^{S}[\mathbf{x}]$

@dot product

$$\langle \mathcal{P}(\mathbf{x}), \mathcal{Q}(\mathbf{x}) \rangle = \sum_{(p,\mathbf{e}) \in \mathcal{P}} \sum_{(q,\mathbf{e}) \in \mathcal{Q}} pq$$

example

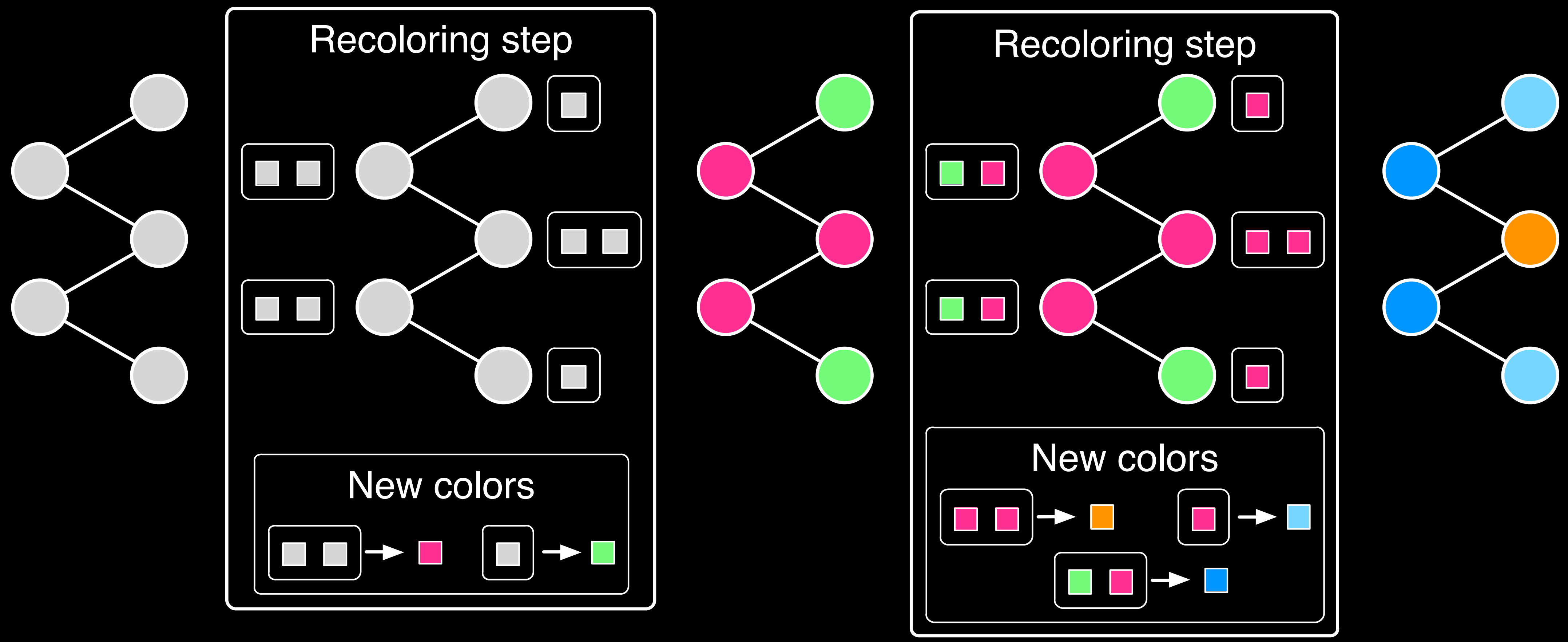


Weisfeiler-Lehman algorithm

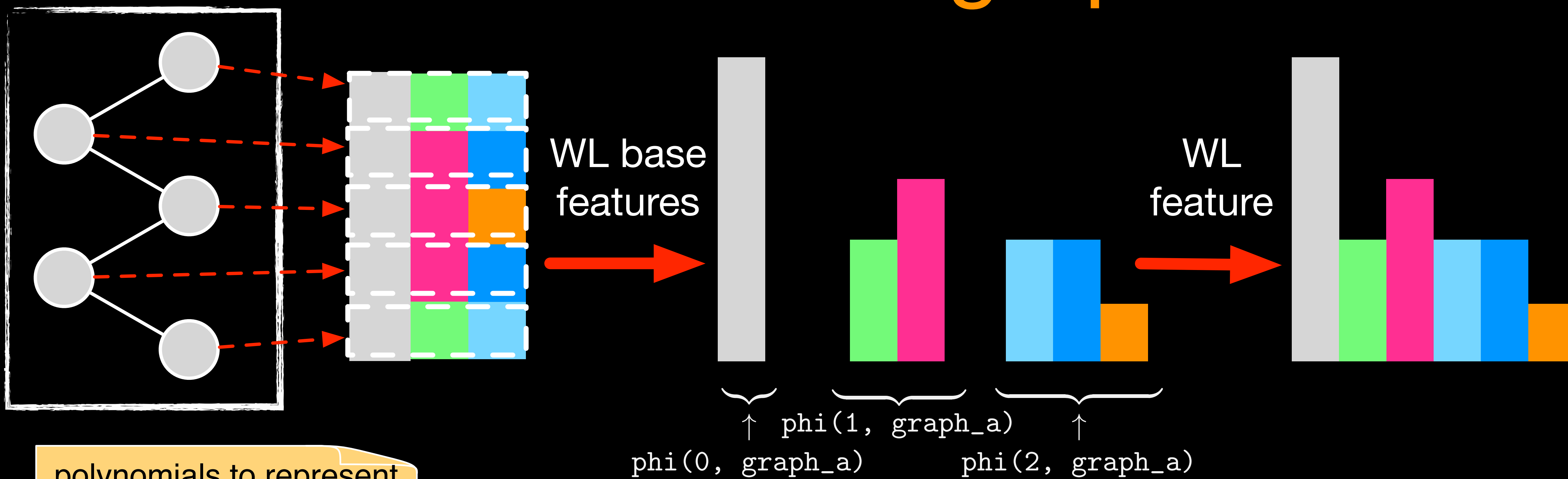
(a.k.a. color refinement)

Can also be used to initialise GI-testing algorithms.

$$\mathcal{L}^h(v) = \begin{cases} \ell(v) & \text{if } h = 0 \\ f(\{\mathcal{L}^{h-1}(w) | w \in \mathcal{N}(v)\}) & \text{if } h > 0 \end{cases}$$



Weisfeiler-Lehman graph kernel

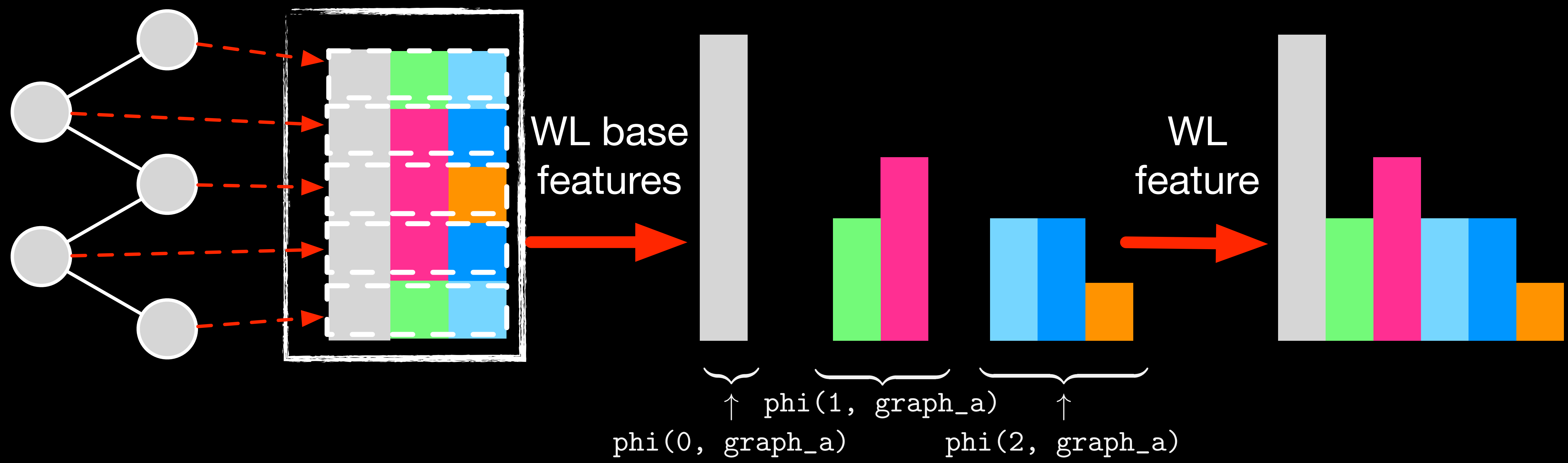


polynomials to represent graph labels

```
:- declare(vertex/2, polynomial(int)).
:- declare(edge_asymm/3, boolean).
:- declare(edge/3, polynomial(int)).

1 * x(gray)::vertex(graph_a, 1).
1 * x(gray)::vertex(graph_a, 2).
1 * x(gray)::vertex(graph_a, 3).
1 * x(gray)::vertex(graph_a, 4).
1 * x(gray)::vertex(graph_a, 5).
```

```
edge_asymm(graph_a, 1, 2).
edge_asymm(graph_a, 2, 3).
edge_asymm(graph_a, 3, 4).
edge_asymm(graph_a, 4, 5).

1.0::edge(Graph, A, B):-
     edge_asymm(Graph, A, B).

1.0::edge(Graph, A, B):-
     edge_asymm(Graph, B, A).
```

Weisfeiler-Lehman graph kernel



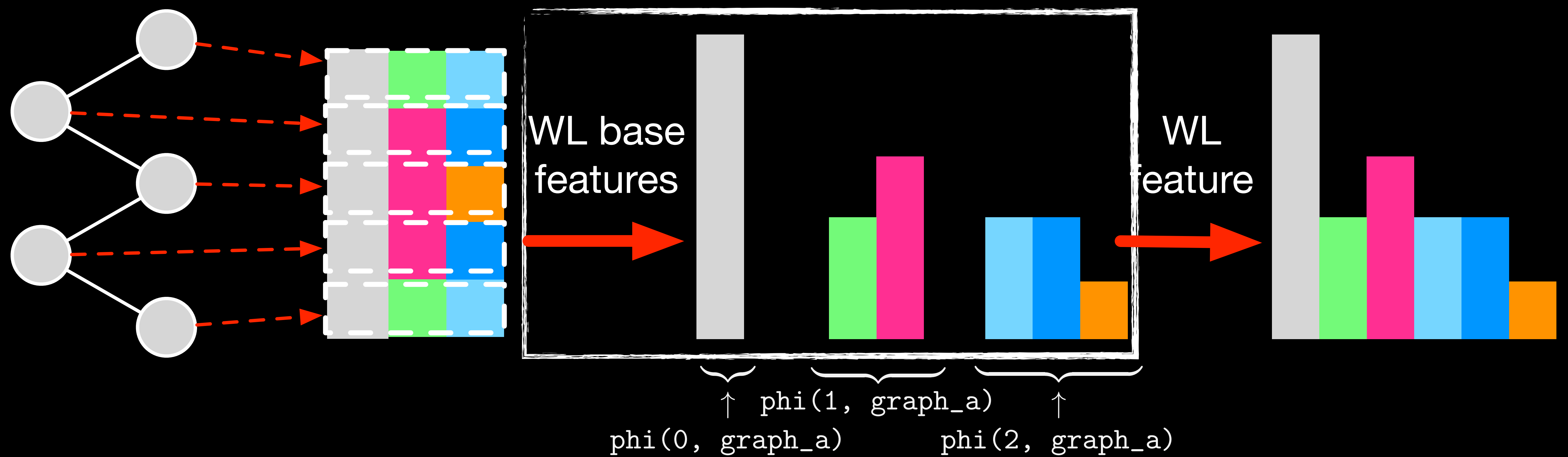

```
:- declare(wl_color/3,
    polynomial(int)).
:- declare(wl_color_multiset/3,
    polynomial(int)).

wl_color_multiset(H, Graph, V):-
  edge(Graph, V, W),
  wl_color(H, Graph, W).
```

```
wl_color(0, Graph, V):-
  vertex(Graph, V).

wl_color(H, Graph, V):-
  H > 0,
  H1 is H - 1,
  @id[wl_color_multiset(H1, Graph, V)].
```

polynomials represent multisets of labels

@id meta-function for recoloring

Weisfeiler-Lehman graph kernel



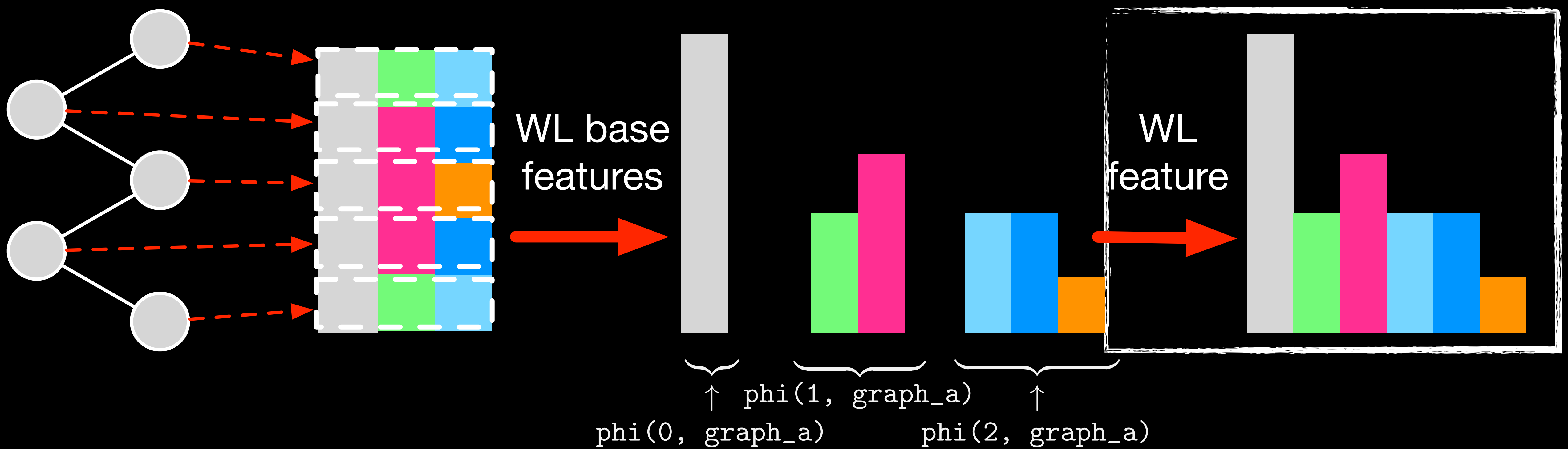

```
:- declare(phi/2, real).
phi(H, Graph):-
  wl_color(H, Graph, V).
```

explicit feature vector at iteration H

```
:- declare(base_kernel/3, real).
base_kernel(H, Graph, GraphPrime):-
  @dot[phi(H, Graph),
       phi(H, GraphPrime)].
```

the base kernel H is the dot product between explicit feature vector at iteration H

Weisfeiler-Lehman graph kernel



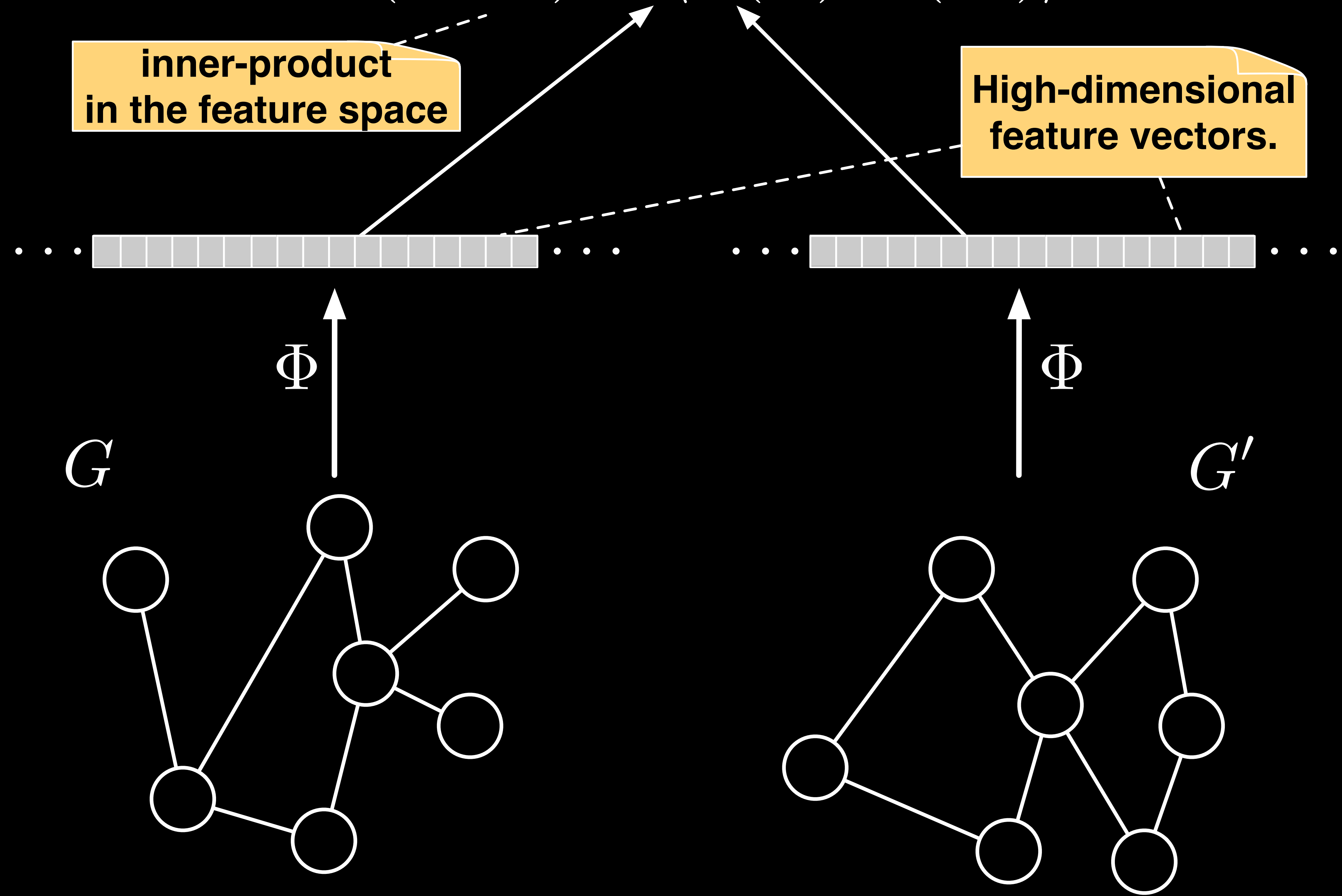

```
:- declare(kernel_wl/3, real).

kernel_wl(0, Graph, GraphPrime):-
  base_kernel(0, Graph, GraphPrime).
```

```
kernel_wl(H, Graph, GraphPrime):-
  H > 0, H1 is H - 1,
  kernel_wl(H1, Graph, GraphPrime).

kernel_wl(H, Graph, GraphPrime):-
  H > 0,
  base_kernel(H, Graph, GraphPrime).
```

accumulate base-kernels of successive iterations

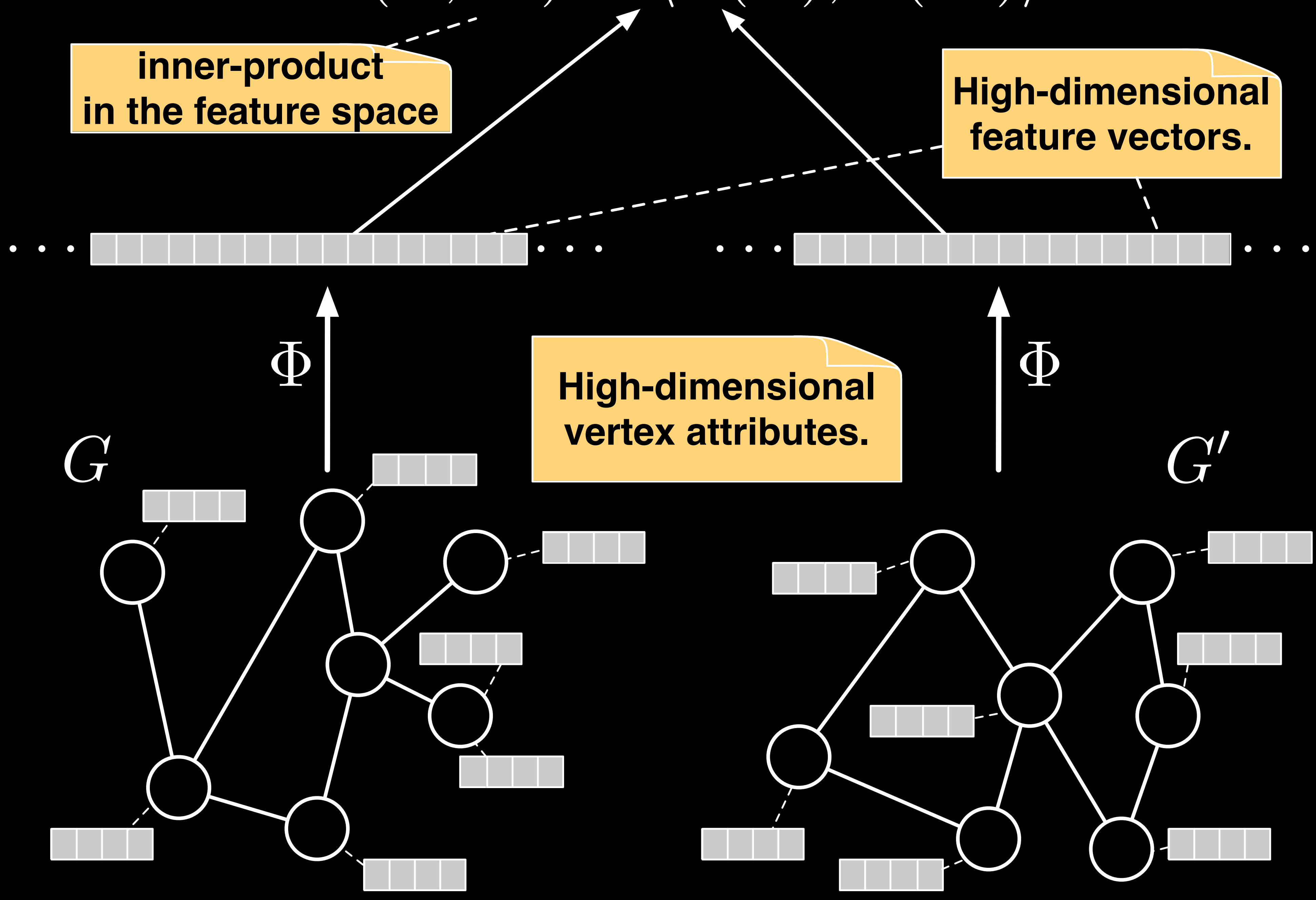
Graph kernels

$$k(G, G') = \langle \Phi(G), \Phi(G') \rangle$$



Graph Kernels with continuous attributes

$$k(G, G') = \langle \Phi(G), \Phi(G') \rangle$$



Graph kernels with continuous attributes

see the ILP2015 paper for details

Outline

- Motivation
- kProlog^{S}
  - Algebraic T_P-operator
  - Tensor operations
- kProlog
  - Algebraic T_P-operator with meta-functions
  - Cyclic programs
- kProlog $^{S[\mathbf{x}]}$
  - Graph kernels
- Conclusions

Novelty related work (1)

ProbLog: probabilistic programming.

- Facts labeled with probabilities.
- Probabilistic Weighted Model Counting.

aProbLog: algebraic generalization of ProbLog.

- Facts labeled with elements of a semiring.
- Algebraic Weighted Model Counting.

kProlog can handle multiple semirings.

- Facts labeled with semiring elements.
- Multiple semirings in the same kProlog program.
- Algebraic Weighted Model Counting is optional (i.e. using the SDD and the BDD semiring).

Novelty related work (2)

kLog: learning with kernels.

- knowledge-based model construction.
- graphicalization declarative specification of graphs.
- can not specify new kernels in the language, allows to plug external graph kernels.

kFOIL: variation of FOIL for learning with kernels.

- can learn simple kernels.
- the kernel defined as the number of clauses that fire in both the interpretations.

kProlog: can declaratively specify kernel features.

- introduction of polynomials for explicit feature extraction.

Conclusions

- kProlog is an algebraic Prolog, and can be used to specify feature spaces and learn with linear separators.
- kProlog is a language that provides a uniform representation for:
  - relational data,
  - background knowledge,
  - kernel design.
- Polynomials and meta-functions allow to specify in kProlog many recent graph kernels.

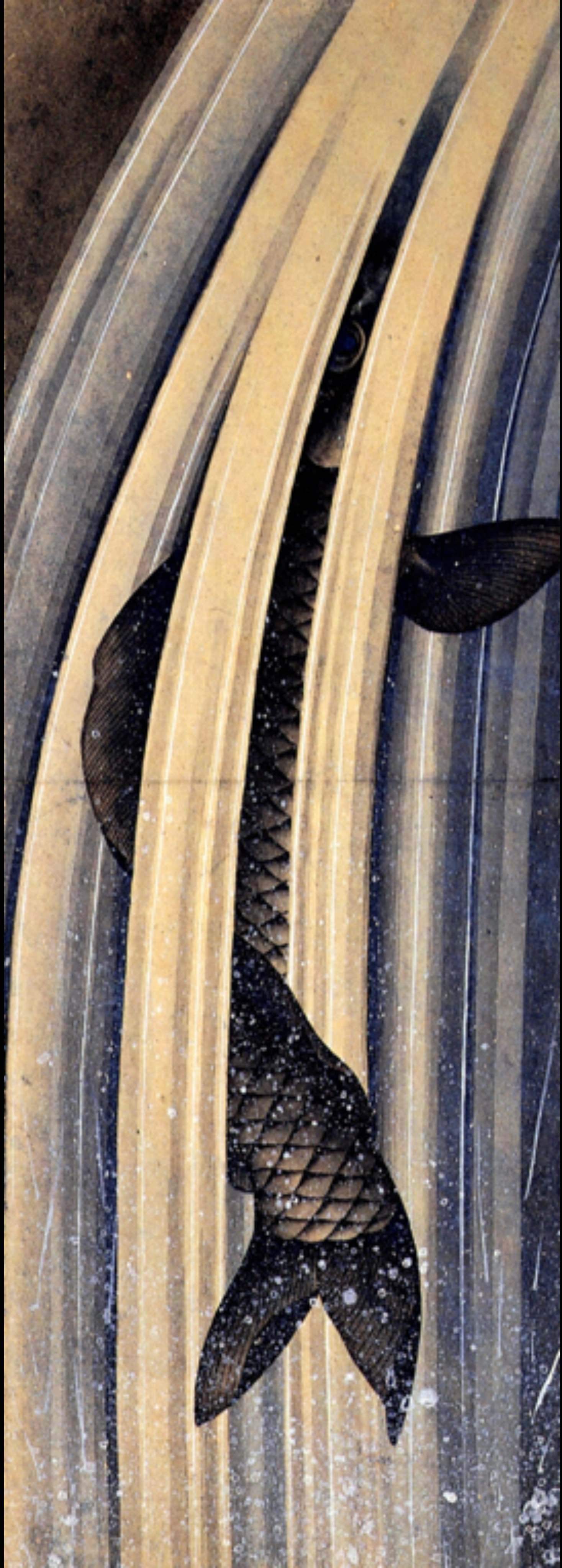
Future work

More declarative specifications of existing graph kernels:

- rational kernels
- shortest path kernels
- …

Kernels on probability distributions (SDD semiring to mimic ProbLog):

- probability product kernels
- Fisher kernel.



Thank you for your attention.

¿ Questions ?

References

[1] L De Raedt, A Kimmig, and H Toivonen. Problog: A probabilistic prolog and its application in link discovery. In IJCAI, 2007.

[2] J Eisner and N W Filardo. Dyna: Extending datalog for modern ai. In Datalog Reloaded. Springer, 2011.

[3] Javier Esparza and Michael Luttenberger. Solving xed-point equations by derivation tree analysis. In Algebra and Coalgebra in Computer Science. Springer, 2011.

[4] Javier Esparza, Stefan Kiefer, and Michael Luttenberger. An extension of newtons method to omega-continuous semirings. In Developments in Language Theory. Springer, 2007.

[5] J Esparza, M Luttenberger, and M Schlund. Fpsolve: A generic solver for fixpoint equations over semirings. In Implementation and Application of Automata. Springer, 2014.

[6] P Frasconi, F Costa, L De Raedt, and K De Grave. klog: A language for logical and relational learning with kernels. Artificial Intelligence, 2014.

[7] Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In Proceedings of the 26th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, 2007.

[8] M Kim and K S Candan. Approximate tensor decomposition within a tensor relational algebraic framework. In Proceedings of the 20th ACM international conference on Information and knowledge management. ACM, 2011.

[9] A Kimmig, G Van den Broeck, and L De Raedt. An algebraic prolog for reasoning about possible worlds. In 25th AAAI Conference on Artificial Intelligence, 2011.

References

[10] N Landwehr, A Passerini, L De Raedt, and P Frasconi. kfoil: Learning simple relational kernels. In AAAI, 2006.
[11] Daniel J Lehmann. Algebraic structures for transitive closure. Theoretical Computer Science, 1977.
[12] M Neumann, N Patricia, R Garnett, and K Kersting. Efficient graph kernels by randomization. In Machine Learning and Knowledge Discovery in Databases. Springer, 2012.
[13] F Orsini, P Frasconi, and L De Raedt. Graph invariant kernels. In Proceedings of the 24th IJCAI, 2015.
[14] M Richardson and P Domingos. Markov logic networks. Machine learning, 2006.
[15] T Sato and Y Kameya. Prism: a language for symbolic-statistical modeling. In IJCAI, 1997.
[16] N Shervashidze, P Schweitzer, E J Van Leeuwen, K Mehlhorn, and K M Borgwardt. Weisfeiler-lehman graph kernels. The Journal of Machine Learning Research, 2011.
[17] J Vlasselaer, G Van den Broeck, A Kimmig, W Meert, and L De Raedt. Anytime inference in probabilistic logic programs with tp-compilation. In Proceedings of the 24th IJCAI, 2015.
[18] J Whaley, D Avots, M Carbin, and M S Lam. Using datalog with binary decision diagrams for program analysis. In Programming Languages and Systems. Springer, 2005.