Optimal Dictionary-Based Compression with Answer Set Programming: Encodings and Empirical Analysis

Mutsunori Banbara[1] Hideo Bannai[2] Takashi Horiyama[3]
Dominik Köppl[4] Takuya Mieno[5] Hideotomo Nabeshima[4]

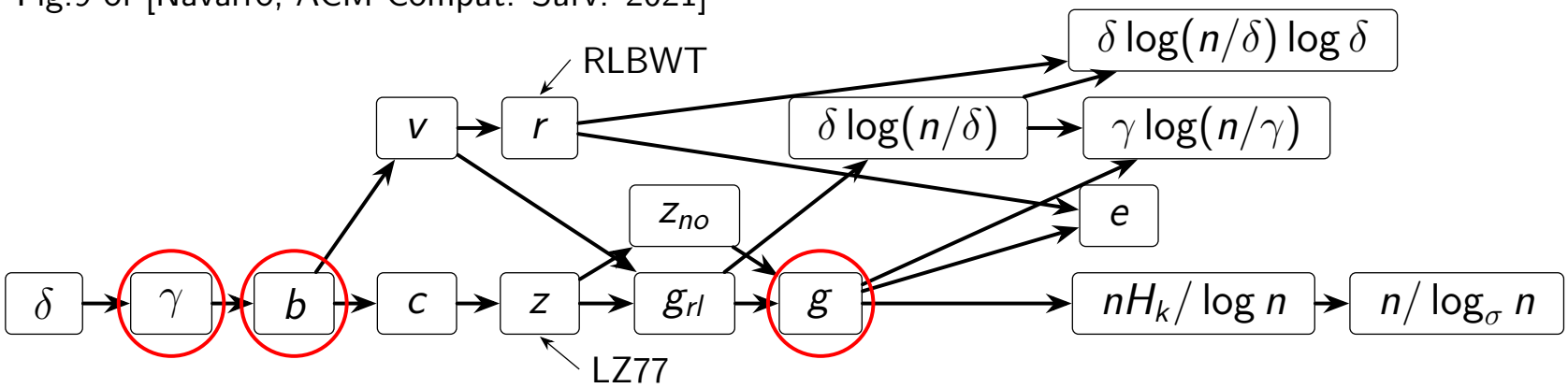
[1]Nagoya University, Japan
[2]Institute of Science Tokyo, Japan
[3]Hokkaido University, Japan
[4]University of Yamanashi, Japan
[5]The University of Electro-Communications, Japan

KR2026@Lisbon

Repetitiveness measures

- Repetitiveness measures represent how compressible the string is.
- Measures based on **dictionary-based compression** are a hot topic in data compression, due to highly repetitive datasets (e.g., versioned documents, genome sets, etc.)

Fig.9 of [Navarro, ACM Comput. Surv. 2021]



- Some measures are **NP-hard** to compute.
- Being able to compute exact values could lead to better understanding of these measures or to improvements of compression algorithms.

Why Answer Set Programming (ASP) ?

ASP is a declarative programming paradigm, combining a rich modeling language with high performance solving capacities.

- ASP is well suited for modeling combinatorial optimization problems, and has been successfully applied in diverse areas of AI.
- Recent advances in ASP suggest promising directions to make it more **scalable** to large graph problems:
  - ASP modulo **acyclicity** [Bomanson et al., 2016]
  - Refined encodings of **acyclicity** constraints [Gebser et al., 2020]
  - Multi-shot ASP solving and theory propagator [Kaminski et al., 2023]

Question

Can a general-purpose ASP approach compete with a more dedicated MaxSAT approach for computing repetitiveness measures?

Our contributions

First systematic application of ASP to dictionary-based compression

- A collection of ASP encodings for computing
  - smallest bidirectional macro scheme (BMS) b
  - smallest straight-line program (SLP) g

  which are all NP-hard to compute.
- Our experiments on the Calgary Corpus show that our approach
  - scales up to strings of length 7000, and
  - competes with a more dedicated MaxSAT encoding [Bannai et al., 2022].

Our contributions

First systematic application of ASP to dictionary-based compression

- A collection of ASP encodings for computing
  - smallest bidirectional macro scheme (BMS) b
  - smallest straight-line program (SLP) g

  which are all NP-hard to compute.
- Our experiments on the Calgary Corpus show that our approach
  - scales up to strings of length 7000, and
  - competes with a more dedicated MaxSAT encoding [Bannai et al., 2022].

Two main use cases of our tool

1. can support theoretical investigations in the stringology/combinatorics on words community.
2. can serve as an offline gold standard generator to evaluate the approximation ratios of linear-time heuristics.

Bidirectional Macro Scheme (BMS) [Storer & Szymanski 1982]

A **bidirectional macro scheme (BMS)** of size f representing a string T is a partition $T = F_1, \ldots, F_f$, where each phrase F_x is encoded as

- a single character (*ground phrase*), or,
- (i_x, ℓ_x) where $F_x = T[i_x..i_x + \ell_x)$

A BMS is *valid*, if T can be reconstructed from the encoding.
($\Leftrightarrow$ the referencing of positions forms a forest rooted at ground phrases)

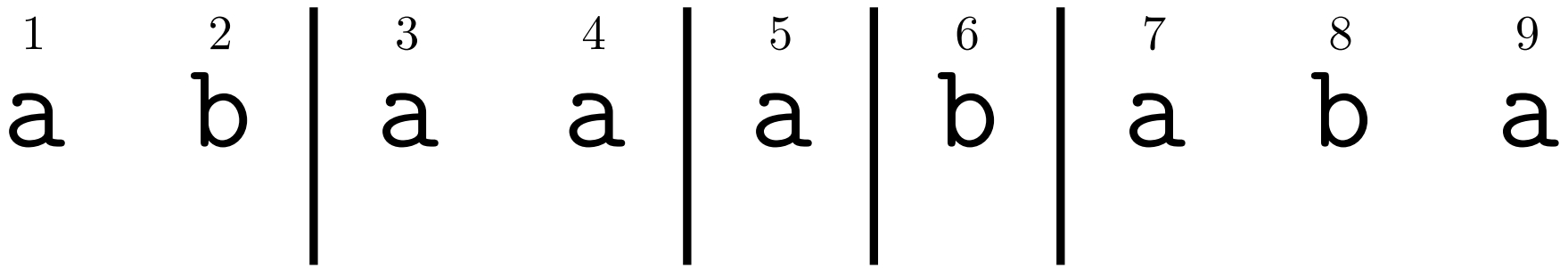
1	2	3	4	5	6	7	8	9
a	b	a	a	a	b	a	b	a

Bidirectional Macro Scheme (BMS) [Storer & Szymanski 1982]

A **bidirectional macro scheme (BMS)** of size f representing a string T is a partition $T = F_1, \ldots, F_f$, where each phrase F_x is encoded as

- a single character (*ground phrase*), or,
- (i_x, ℓ_x) where $F_x = T[i_x..i_x + \ell_x)$

A BMS is *valid*, if T can be reconstructed from the encoding.
($\Leftrightarrow$ the referencing of positions forms a forest rooted at ground phrases)

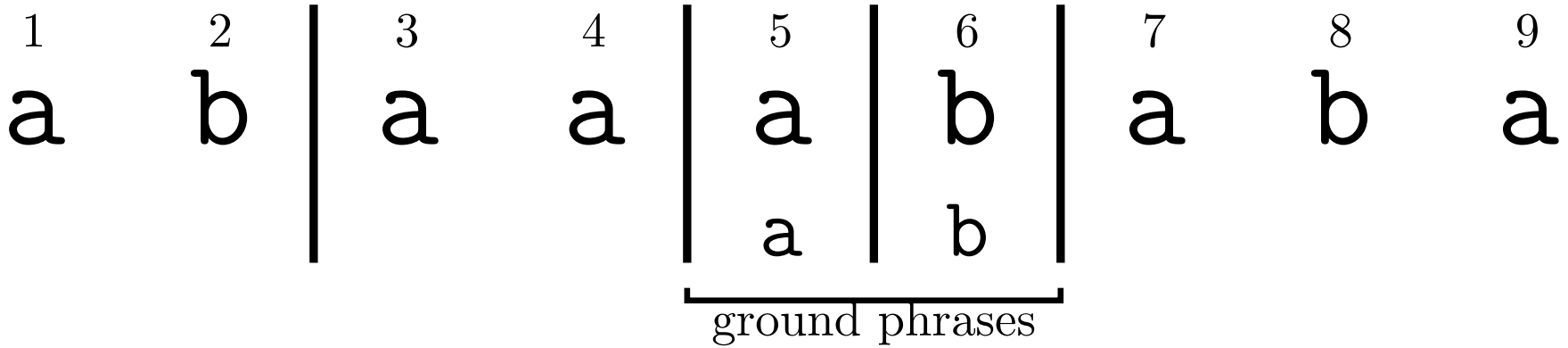


Bidirectional Macro Scheme (BMS) [Storer & Szymanski 1982]

A **bidirectional macro scheme (BMS)** of size f representing a string T is a partition $T = F_1, \ldots, F_f$, where each phrase F_x is encoded as

- a single character (*ground phrase*), or,
- (i_x, ℓ_x) where $F_x = T[i_x..i_x + \ell_x)$

A BMS is *valid*, if T can be reconstructed from the encoding.
($\Leftrightarrow$ the referencing of positions forms a forest rooted at ground phrases)

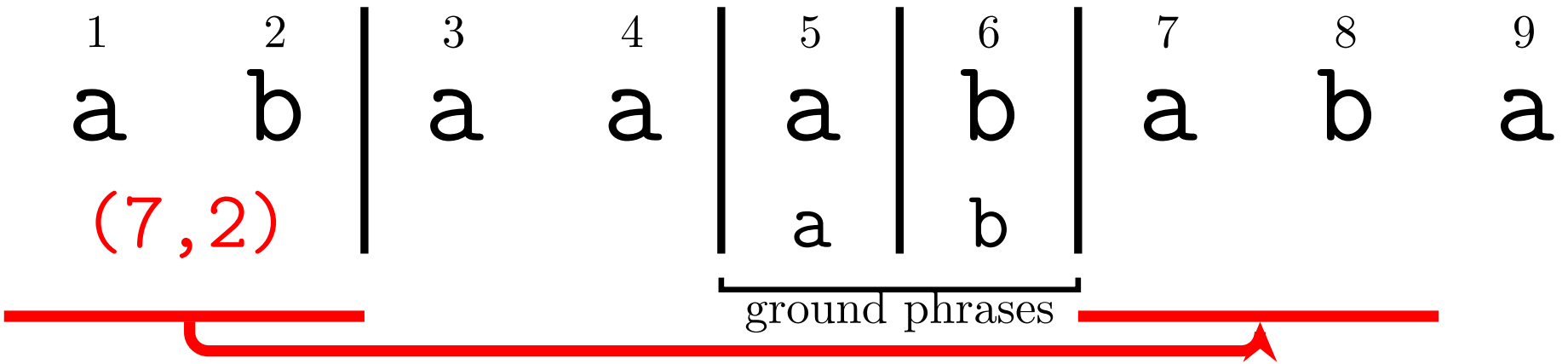


Bidirectional Macro Scheme (BMS) [Storer & Szymanski 1982]

A **bidirectional macro scheme (BMS)** of size f representing a string T is a partition $T = F_1, \ldots, F_f$, where each phrase F_x is encoded as

- a single character (*ground phrase*), or,
- (i_x, ℓ_x) where $F_x = T[i_x..i_x + \ell_x)$

A BMS is *valid*, if T can be reconstructed from the encoding.
($\Leftrightarrow$ the referencing of positions forms a forest rooted at ground phrases)

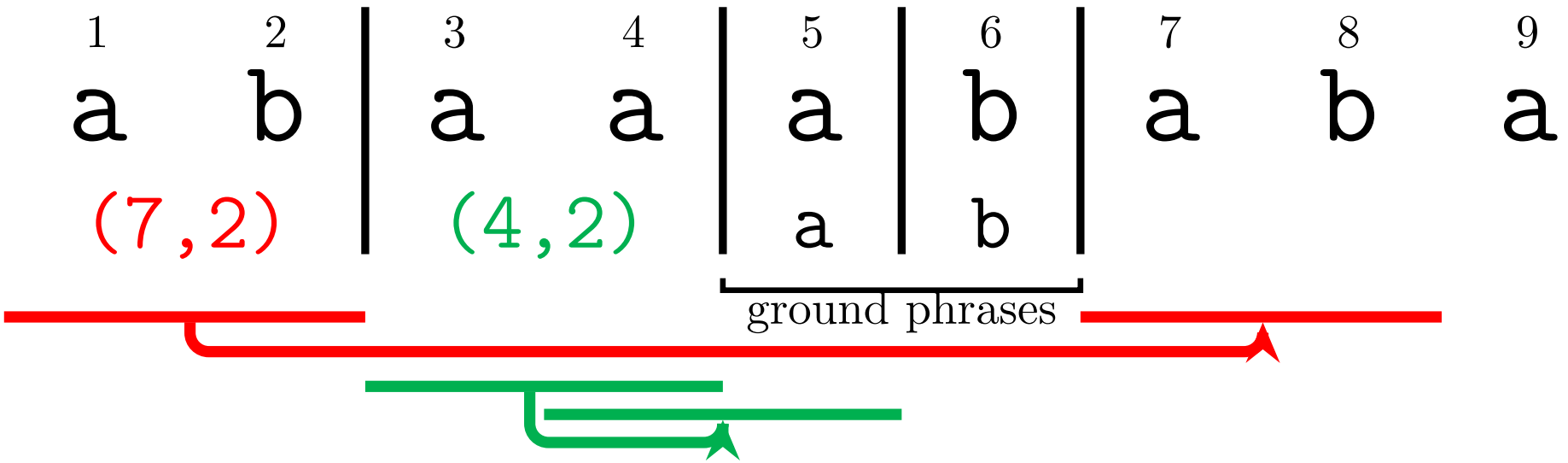


Bidirectional Macro Scheme (BMS) [Storer & Szymanski 1982]

A **bidirectional macro scheme (BMS)** of size f representing a string T is a partition $T = F_1, \ldots, F_f$, where each phrase F_x is encoded as

- a single character (*ground phrase*), or,
- (i_x, ℓ_x) where $F_x = T[i_x..i_x + \ell_x)$

A BMS is *valid*, if T can be reconstructed from the encoding.
($\Leftrightarrow$ the referencing of positions forms a forest rooted at ground phrases)

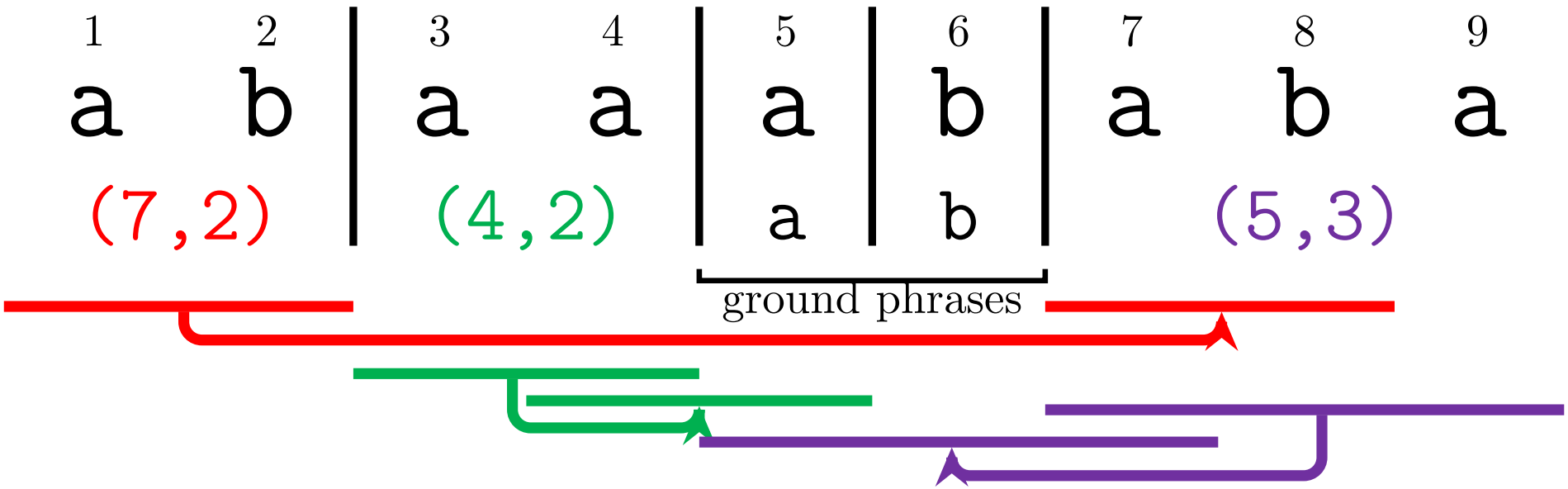


Bidirectional Macro Scheme (BMS) [Storer & Szymanski 1982]

A **bidirectional macro scheme (BMS)** of size f representing a string T is a partition $T = F_1, \ldots, F_f$, where each phrase F_x is encoded as

- a single character (*ground phrase*), or,
- (i_x, ℓ_x) where $F_x = T[i_x..i_x + \ell_x)$

A BMS is *valid*, if T can be reconstructed from the encoding.
($\Leftrightarrow$ the referencing of positions forms a forest rooted at ground phrases)

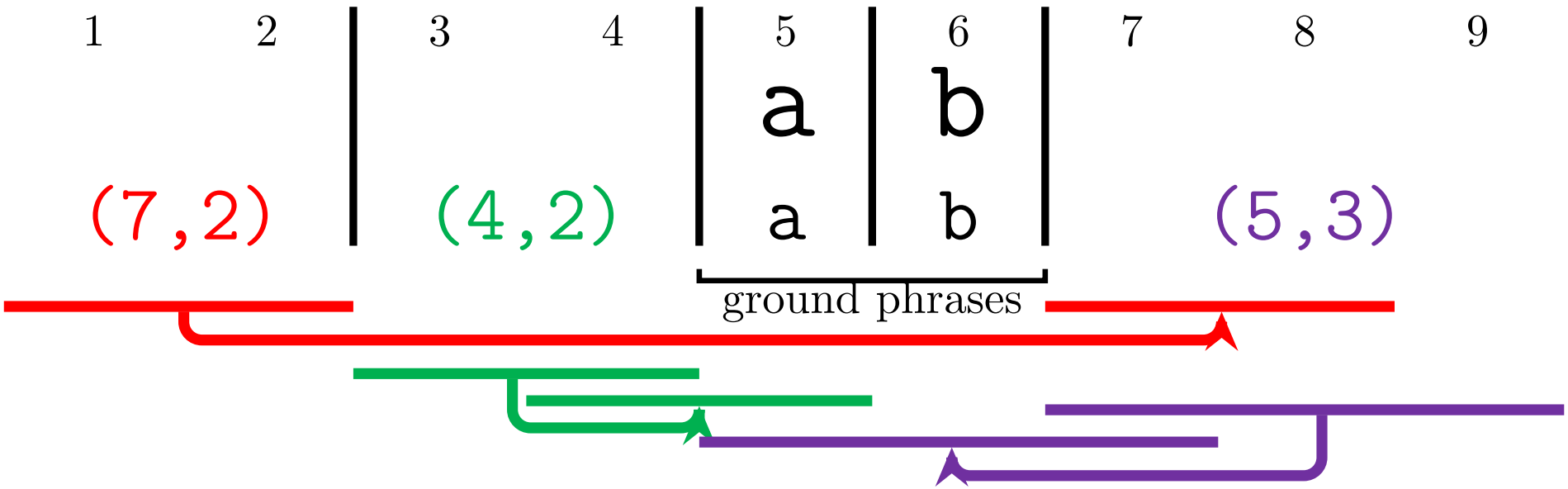


Bidirectional Macro Scheme (BMS) [Storer & Szymanski 1982]

A **bidirectional macro scheme (BMS)** of size f representing a string T is a partition $T = F_1, \ldots, F_f$, where each phrase F_x is encoded as

- a single character (*ground phrase*), or,
- (i_x, ℓ_x) where $F_x = T[i_x..i_x + \ell_x)$

A BMS is *valid*, if T can be reconstructed from the encoding.
($\Leftrightarrow$ the referencing of positions forms a forest rooted at ground phrases)

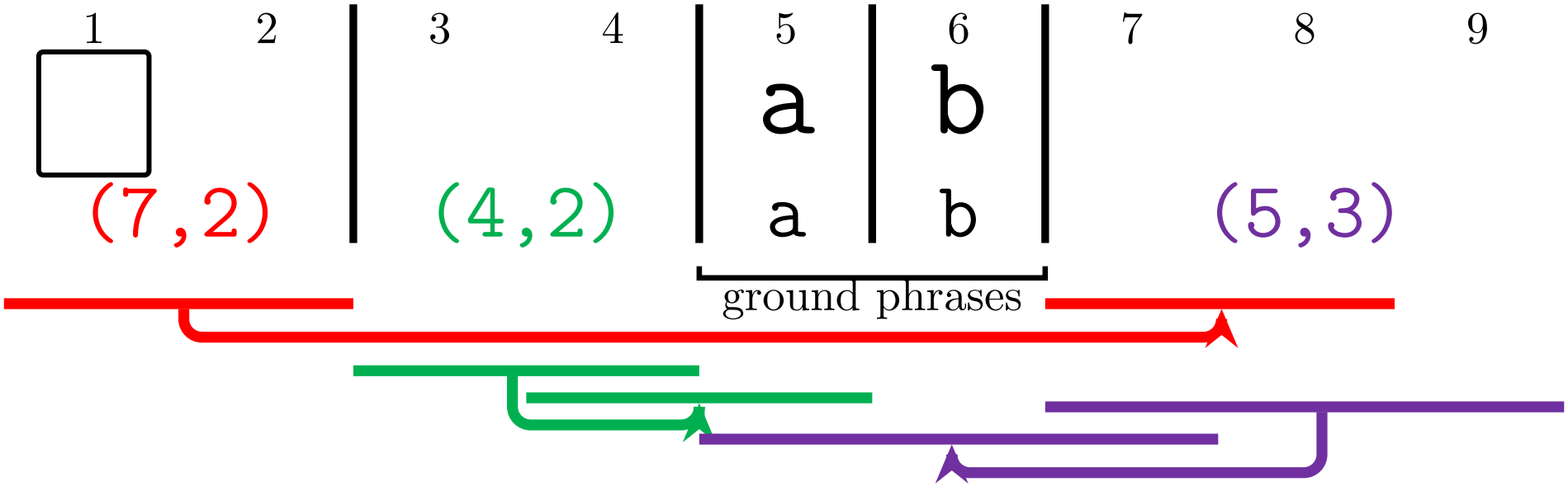


Bidirectional Macro Scheme (BMS) [Storer & Szymanski 1982]

A **bidirectional macro scheme (BMS)** of size f representing a string T is a partition $T = F_1, \ldots, F_f$, where each phrase F_x is encoded as

- a single character (*ground phrase*), or,
- (i_x, ℓ_x) where $F_x = T[i_x..i_x + \ell_x)$

A BMS is *valid*, if T can be reconstructed from the encoding.
($\Leftrightarrow$ the referencing of positions forms a forest rooted at ground phrases)

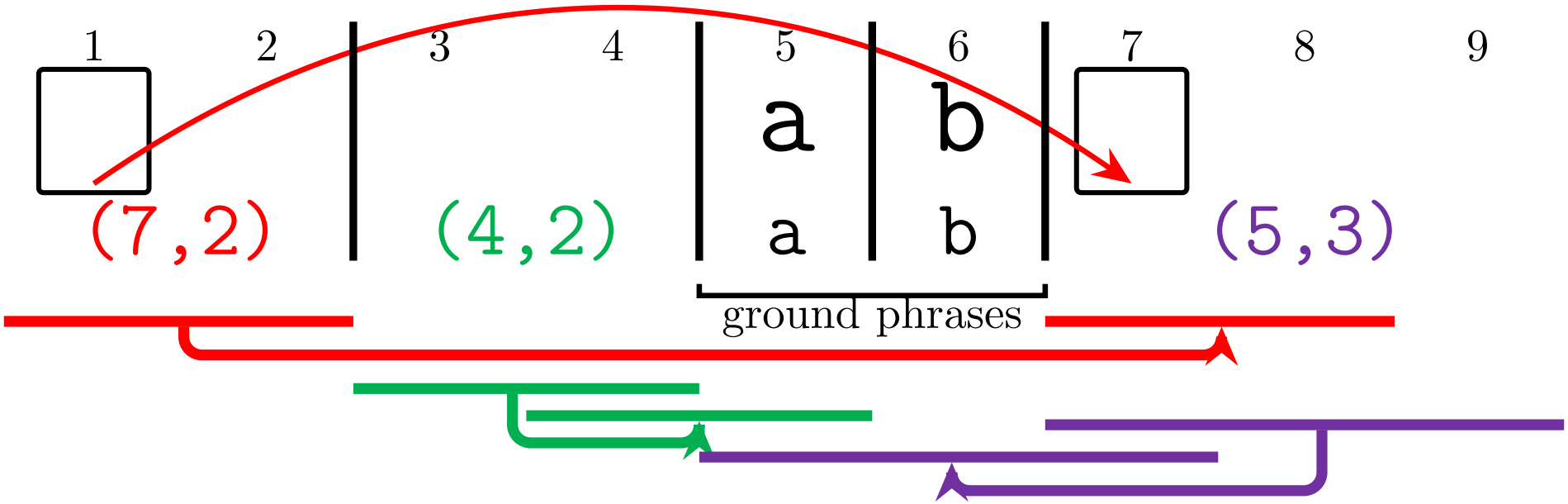


Bidirectional Macro Scheme (BMS) [Storer & Szymanski 1982]

A **bidirectional macro scheme (BMS)** of size f representing a string T is a partition $T = F_1, \ldots, F_f$, where each phrase F_x is encoded as

- a single character (*ground phrase*), or,
- (i_x, ℓ_x) where $F_x = T[i_x..i_x + \ell_x)$

A BMS is *valid*, if T can be reconstructed from the encoding.
($\Leftrightarrow$ the referencing of positions forms a forest rooted at ground phrases)

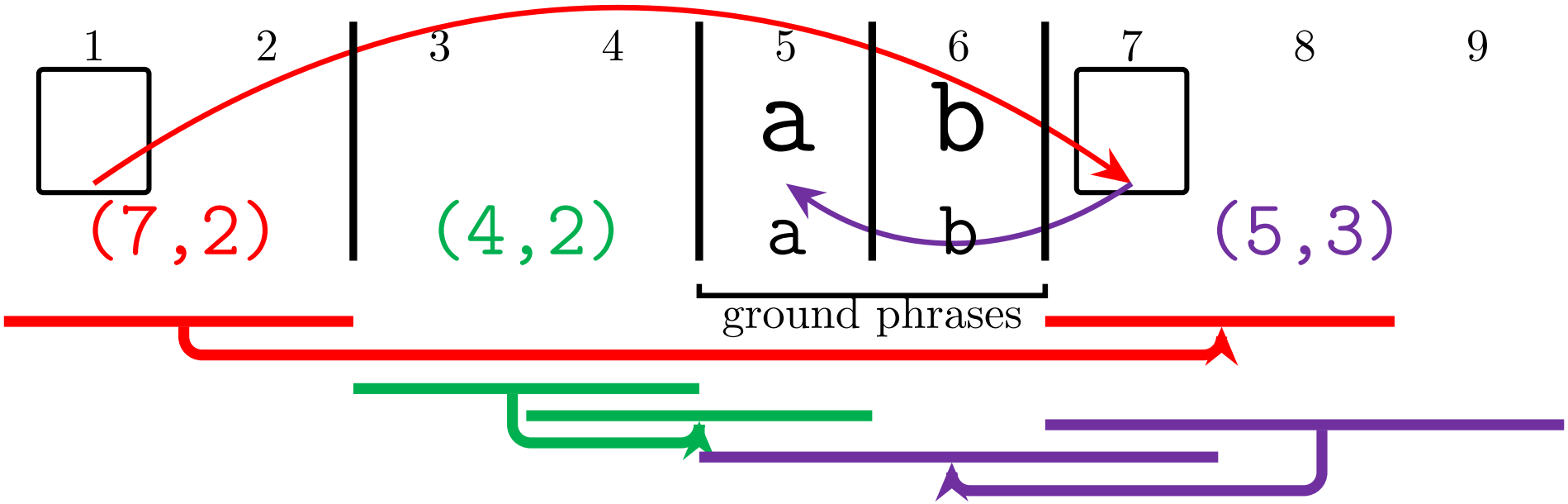


Bidirectional Macro Scheme (BMS) [Storer & Szymanski 1982]

A **bidirectional macro scheme (BMS)** of size f representing a string T is a partition $T = F_1, \ldots, F_f$, where each phrase F_x is encoded as

- a single character (*ground phrase*), or,
- (i_x, ℓ_x) where $F_x = T[i_x..i_x + \ell_x)$

A BMS is *valid*, if T can be reconstructed from the encoding.
($\Leftrightarrow$ the referencing of positions forms a forest rooted at ground phrases)

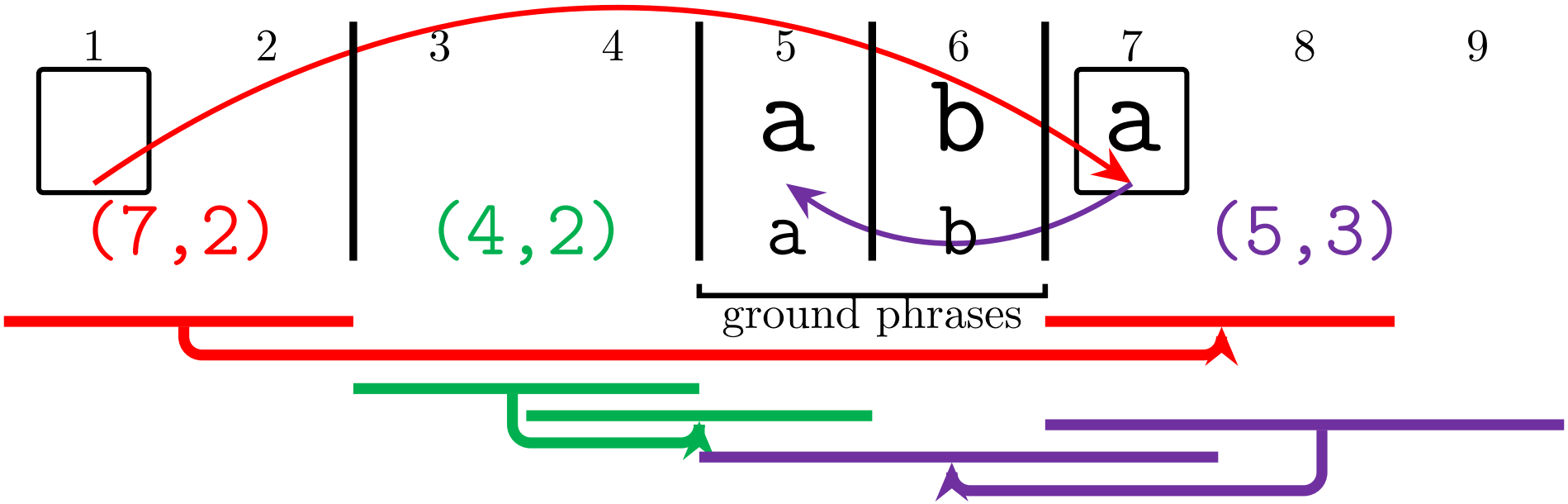


Bidirectional Macro Scheme (BMS) [Storer & Szymanski 1982]

A **bidirectional macro scheme (BMS)** of size f representing a string T is a partition $T = F_1, \ldots, F_f$, where each phrase F_x is encoded as

- a single character (*ground phrase*), or,
- (i_x, ℓ_x) where $F_x = T[i_x..i_x + \ell_x)$

A BMS is *valid*, if T can be reconstructed from the encoding.
($\Leftrightarrow$ the referencing of positions forms a forest rooted at ground phrases)

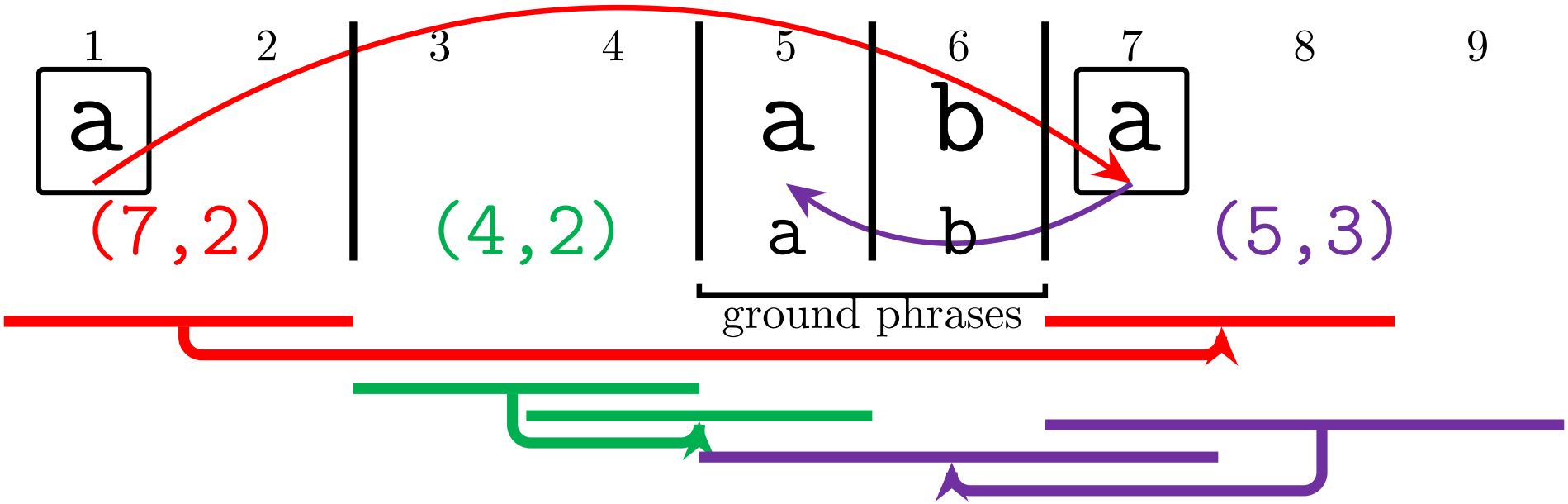


Bidirectional Macro Scheme (BMS) [Storer & Szymanski 1982]

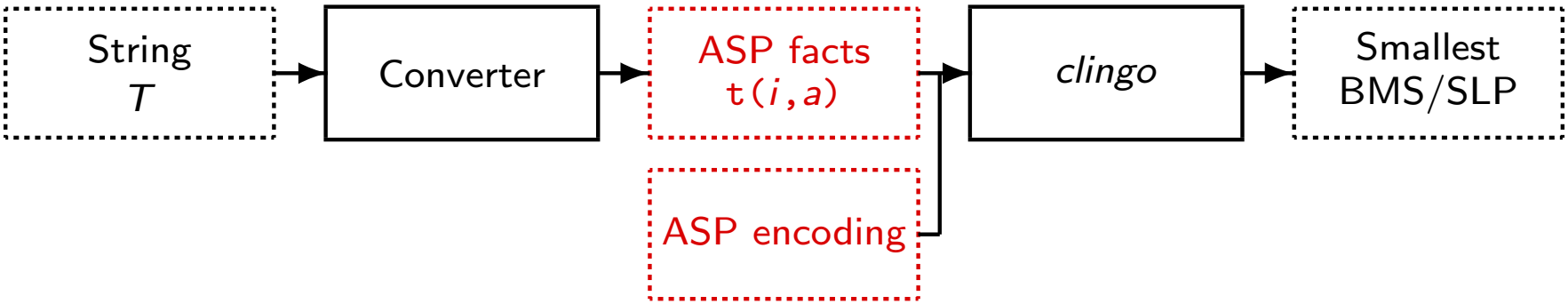
A **bidirectional macro scheme (BMS)** of size f representing a string T is a partition $T = F_1, \ldots, F_f$, where each phrase F_x is encoded as

- a single character (*ground phrase*), or,
- (i_x, ℓ_x) where $F_x = T[i_x..i_x + \ell_x)$

A BMS is *valid*, if T can be reconstructed from the encoding.
($\Leftrightarrow$ the referencing of positions forms a forest rooted at ground phrases)



Our ASP-based approach



- The resulting solver reads a string T and converts it into ASP facts in a standard way.
- In turn, these facts are combined with an ASP encoding for BMS/SLP optimization, which is subsequently solved by an ASP solver *clingo*.

ASP fact format of string input

1	2	3	4	5	6	7	8	9
a	b	a	a	a	b	a	b	a

ASP fact format

```
#const textlength=9.
t(1,"a").  t(2,"b").  t(3,"a").
t(4,"a").  t(5,"a").  t(6,"b").
t(7,"a").  t(8,"b").  t(9,"a").
```

ASP encoding for BMS optimization

1	2	3	4	5	6	7	8	9
a	b	a	a	a	b	a	b	a
p(1)	p(2)	p(3)	p(4)	p(5)	p(6)	p(7)	p(8)	p(9)

- Atoms
  - `p(I)` for each position `I`: `p(I)` is true $\Leftrightarrow$ `I` is start of phrase
  - `r(I,J)` for positions `I,J`: `r(I,J)` is true $\Leftrightarrow$ position `I` references position `J`
- Weak constraints (optimization statements)

  ```
  #minimize { 1,I : p(I) }.
  ```
- Hard constraints
  1. Adjacent positions should reference adjacent positions, except when phrase starts are involved:

     ```
     :- not p(I), r(I,J), not r(I-1,J-1).
     ```
  2. Referencing forms a forest of rooted trees.

ASP encoding for BMS optimization

1	2	3	4	5	6	7	8	9
a	b	a	a	a	b	a	b	a
p(1)	p(2)	p(3)	p(4)	p(5)	p(6)	p(7)	p(8)	p(9)

- Atoms
  - `p(I)` for each position `I`: `p(I)` is true $\Leftrightarrow$ `I` is start of phrase
  - `r(I,J)` for positions `I,J`: `r(I,J)` is true $\Leftrightarrow$ position `I` references position `J`
- Weak constraints (optimization statements)

  ```
  #minimize { 1,I : p(I) }.
  ```
- Hard constraints
  1. Adjacent positions should reference adjacent positions, except when phrase starts are involved:

     ```
     :- not p(I), r(I,J), not r(I-1,J-1).
     ```
  2. Referencing forms a forest of rooted trees.

     Bottleneck: the validity requires the reference graph is acyclic.

Full encoding for BMS optimization

Base part

```
{ p(1..textlength) }.                    % starting position of phrase
#minimize { 1,I : p(I) }.                        % objective function

% phrase reference
{ r(I,J) ; r(J,I) } 1 :- t(I,C), t(J,C), I < J, J <= textlength.

:- not { r(I,_) } 1, I=1..textlength.     % reference uniqueness
:- not p(I), 0 { r(I,_) } 0, I=1..textlength.    % ground phrase
:- not p(I), r(I,J), not r(I-1,J-1).            % phrase boundary
```

Acyclicity constraints (*reachability* encoding)

```
reachable(I,I) :- 0 { r(I,_) } 0, I=1..textlength.
reachable(X,R) :- r(X,Y), reachable(Y,R).
:- p(I), not 1 { reachable(I,_) } 1.
```

Full encoding for BMS optimization

Base part

```
{ p(1..textlength) }.                    % starting position of phrase
#minimize { 1,I : p(I) }.                        % objective function

% phrase reference
{ r(I,J) ; r(J,I) } 1 :- t(I,C), t(J,C), I < J, J <= textlength.

:- not { r(I,_) } 1, I=1..textlength.     % reference uniqueness
:- not p(I), 0 { r(I,_) } 0, I=1..textlength.    % ground phrase
:- not p(I), r(I,J), not r(I-1,J-1).             % phrase boundary
```

Acyclicity constraints (*closure* encoding)

```
closure(X,Y) :- r(X,Y).
closure(X,Z) :- r(Y,Z), closure(X,Y).
:- closure(X,Y), closure(Y,X).
```

Full encoding for BMS optimization

Base part

```
{ p(1..textlength) }.                    % starting position of phrase
#minimize { 1,I : p(I) }.                        % objective function

% phrase reference
{ r(I,J) ; r(J,I) } 1 :- t(I,C), t(J,C), I < J, J <= textlength.

:- not { r(I,_) } 1, I=1..textlength.     % reference uniqueness
:- not p(I), 0 { r(I,_) } 0, I=1..textlength.    % ground phrase
:- not p(I), r(I,J), not r(I-1,J-1).           % phrase boundary
```

To improve the scalability for large input strings, we can leverage:

1. ASP modulo acyclicity [Bomanson et al., 2016]
2. Refined encodings of acyclicity constraints [Gebser et al., 2020]

Refined encodings of acyclicity constraints

acyclicity encoding via #edge

```
#edge (I,J): r(I,J).
```

- This enforces that a directed graph defined by `r(I,J)` is acyclic.

Refined encodings of acyclicity constraints

acyclicity encoding via #edge

```
#edge (I,J): r(I,J).
```

- This enforces that a directed graph defined by `r(I,J)` is acyclic.

order encoding

```
node(X) :- X=1..textlength.
pair(I,J) :- t(I,C), t(J,C), I != J, node(I), node(J).
order(X,Y) :- pair(X,Y), not r(X,Y).
order(X,Y) :- pair(X,Y), order(Y).
order(X) :- node(X), order(X,Y) : pair(X,Y).
:- node(X), not order(X).
```

- This ensures acyclicity by inductively deriving positions that do not belong to any cycle.

Summary of our encodings

Encoding	Acyclicity constraints	Main point
reachability	reachability from roots	basic baseline
closure	transitive closure	compact but often large after grounding
acyclicity (`#edge`)	ASP modulo acyclicity	compact and scalable
order	inductive acyclicity checking	solver independent refinement

Key point

The advantage of *acyclicity* and *order* over *reachability* and *closure* is mainly due to their smaller size after grounding.

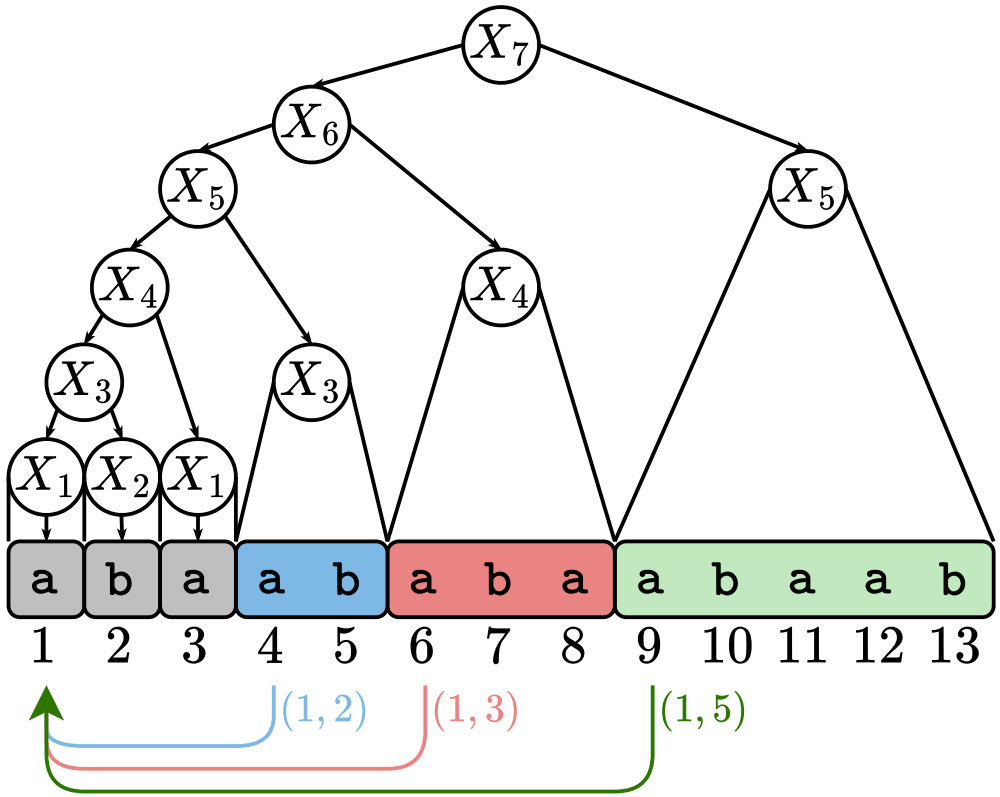
Straight-Line Program (SLP) [Karpinski et al., 1997]

An SLP for string T is a context-free grammar in Chomsky normal form, which derives only T.

- All production rules are of the form $X \to AB$ or $X \to c$.
- SLP of size g induces a BMS of size $g - \sigma + 1$ [Rytter, 2003]

A smallest SLP of size 7 for $T = \mathtt{abaababaabaab}$

$$
\begin{aligned}
X_7 &\to X_6 X_5 \\
X_6 &\to X_5 X_4 \\
X_5 &\to X_4 X_3 \\
X_4 &\to X_3 X_1 \\
X_3 &\to X_1 X_2 \\
X_2 &\to \mathtt{b} \\
X_1 &\to \mathtt{a}
\end{aligned}
$$



ASP encoding for SLP optimization

A BMS with additional constraints in the referencing induces an SLP [Bannai et al.,2022].

The main difference from our encoding for BMS is as follows:

- Additional atoms
  - The atom `q(J,L)` represents that $T[\mathtt{J}..\mathtt{J}+\mathtt{L})$ corresponds to an internal node of the partial parse tree that is referenced by at least one phrase in the grammar.
- Additional constraints
  1. Every reference must start and end at phrase boundaries.
     ```
     :- not p(J), q(J,L).    :- not p(J+L), q(J,L).
     ```
  2. All references must either be nested or disjoint.
     ```
     :- q(I,L), q(J,M), I < J, J < I+L, I+L < J+M.
     ```
- No acyclicity constraints
  - `r(I,J)` is restricted so that position `I` can only reference previous position `J`, but not vice versa.

Experiments

We evaluate our approach for computing the smallest BMS.

- **Benchmark**: we use all instances of the Calgary Corpus
  - Plain text, source code, manuals, articles, and some binary files
  - Prefixes of 100, 300, 500, and 1000 characters from each instance
  - 46 instances for each length (184 instances in total)
- **Encodings**:
  - Our ASP encodings (*closure*, *reachability*, *acyclicity*, and *order*)
  - MaxSAT encoding [Bannai et al., 2022]
- **Solvers**:
  - *clingo* version 5.7.1
  - Two top solvers from the MaxSAT Evaluation 2024 and PySAT
- **Timeout**:
  - 3600-second timeout per instance
  - The timeout includes grounding in the ASP setting but excludes encoding time for MaxSAT.

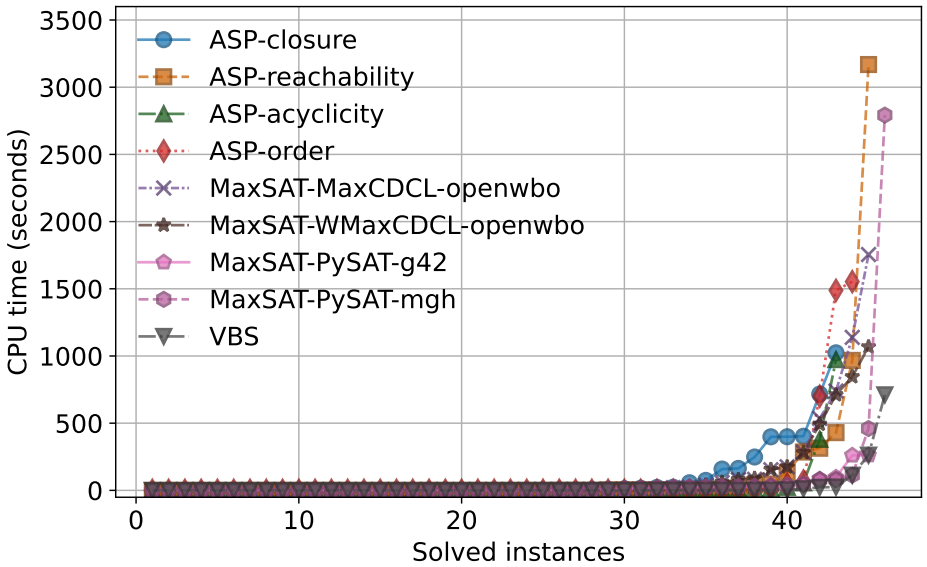
The number of solved instances

Type	*Encoding* / Solver	len100	len300	len500	len1000	all
ASP	*closure*	43	24	13	1	81
	reachability	45	28	6	2	81
	acyclicity	43	30	18	10	101
	order	44	31	17	11	103
MaxSAT	MaxCDCL-openwbo	45	22	2	2	71
	WMaxCDCL-openwbo	45	23	2	2	72
	PySAT-g42	45	23	2	2	72
	PySAT-mgh	46	29	3	2	80
VBS		46	39	24	13	122

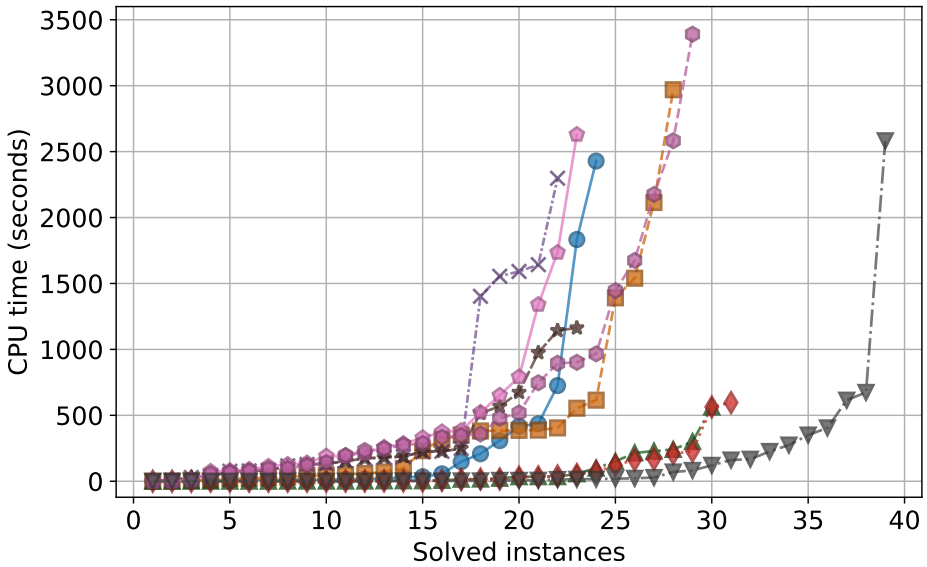
- ASP-*order* solved the largest number of instances (103), followed by ASP-*acyclicity* via #edge (101).
- These two solved 10 and 11 instances of length 1000 (better scalability), whereas the other methods solved only one or two.

Cactus plots showing the CPU time

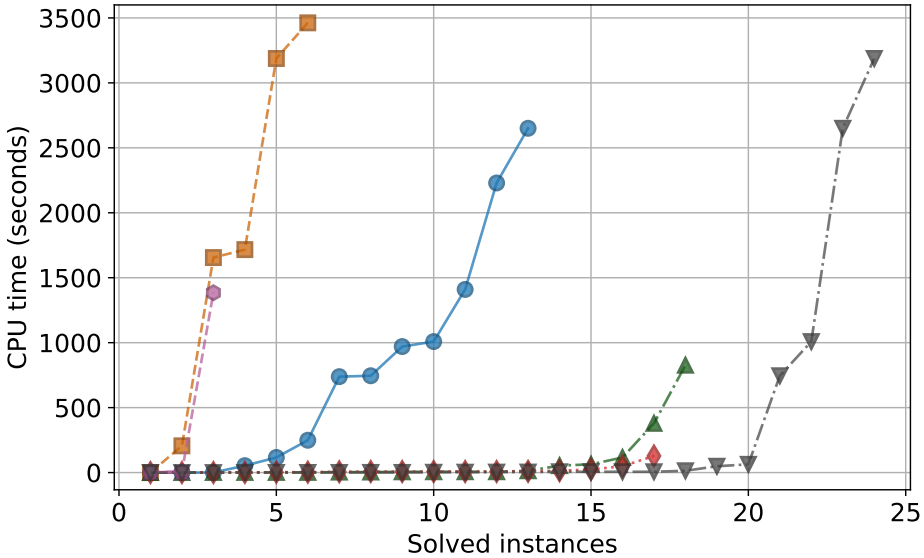
length 100



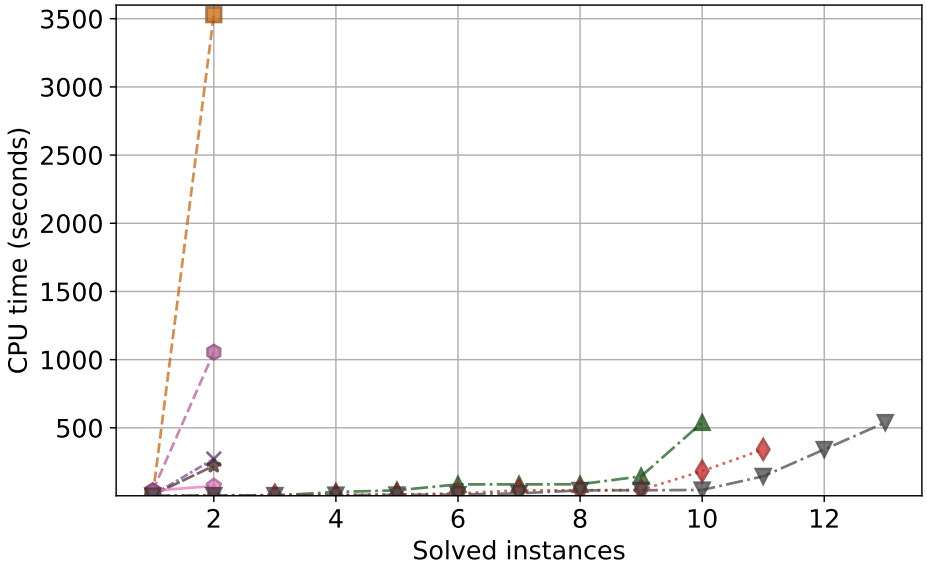
length 300



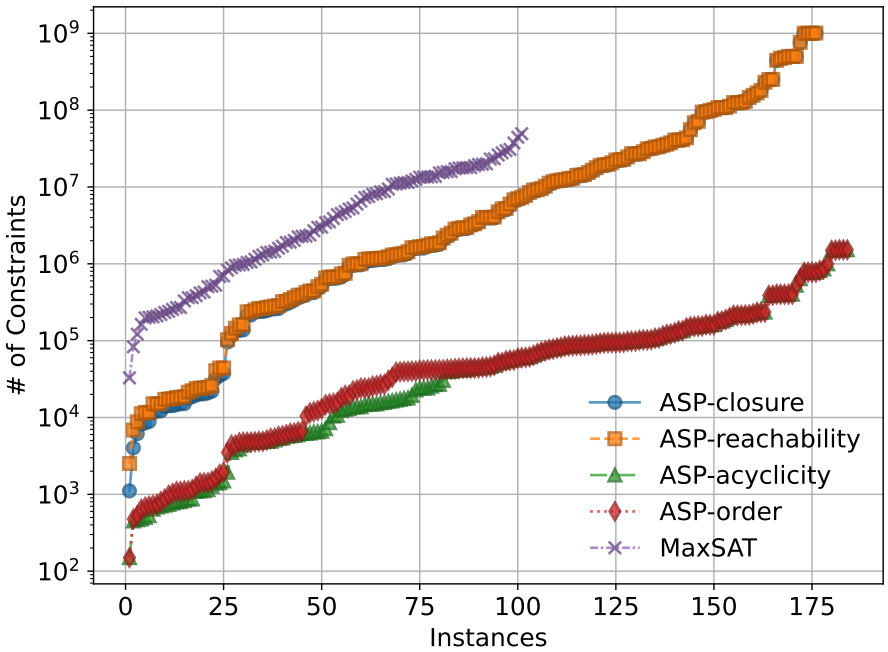
length 500



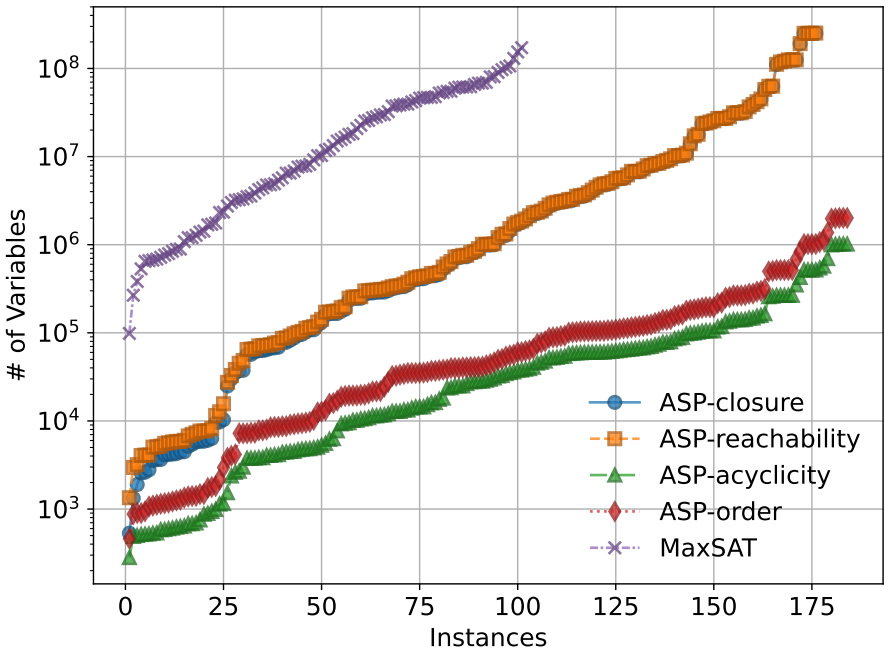
length 1000



The number of constraints and variables



Constraints



Variables

- ASP-*acyclicity* scaled well to longer inputs; the number of constraints and variables is approximately two orders of magnitude smaller than those of ASP-*closure* and ASP-*reachability*.
- Interestingly, ASP-*order* also achieved comparable performance, despite not relying on solver-specific features.

Scalability limits of our ASP-based approach

Our #edge encoding can optimally compress strings of length up to 7,000.

Length	Instance	#char	Optimal BMS size
7,000	aaa.txt	1	2
7,000	pic	2	4
7,000	ptt5	2	4
6,000	random.txt	64	3,550
4,000	book1	67	1,215
3,000	paper2	71	966
2,000	asyoulik.txt	63	716
2,000	paper4	70	698
2,000	pi.txt	10	658

Limit

No encoding was able to find optimal BMSs for strings of length 8,000.

Conclusion

First systematic application of ASP to dictionary-based compression

- A collection of ASP encodings for computing
  - smallest bidirectional macro scheme (BMS)
  - smallest straight-line program (SLP)

  which are all NP-hard to compute.
- Our experiments showed that our approach
  - scales up to strings of length 7000, and
  - competes with a more dedicated MaxSAT encoding.