Solving String Constraints: The Case for Constraint Programming

Jun He,[1,2,3] **Pierre Flener**,[1] Justin Pearson,[1] Wei Ming Zhang[2]

[1] Department of Information Technology, Uppsala University, Sweden

[2] School of Information System and Management
National University of Defense Technology, Changsha, China

[3] PLA Academy of National Defense Information, Wuhan, China

CP 2013 — 18 September 2013

UPPSALA UNIVERSITET

Outline

UPPSALA UNIVERSITET

Verification of String-Operating Programs

- String constraints:
  - Formal language membership
  - Length
  - Concatenation
- Important for the security of web applications

  Example: Detecting SQL injection
- Well-known string solvers: HAMPI, KALUZA, SUSHI, ...
- **Observation:** CP excels at fixed-length string constraints.
- **Question:** Are CP solvers competitive with string solvers?

UPPSALA UNIVERSITET

Outline

1 Motivation

2 Constraints for Membership in Formal Languages

3 An Improved Propagator for the CFG Constraint

4 Experimental Evaluation

5 Conclusion and Future Work

UPPSALA UNIVERSITET

Three String Constraints

- The REGULAR constraint (Pesant, 2004)
  - Membership in a regular language
  - Specified by a deterministic finite automaton (DFA)
- The AUTOMATON constraint (Beldiceanu *et al.*, 2004)
  - Membership in a language specified by a (non-)deterministic finite automaton with counters
- The CFG constraint (Sellmann; Quimper & Walsh; 2006)
  - Membership in a context-free language (CFL)
  - Specified by a context-free grammar (CFG)
- Provide user-level extensibility for CP modelling languages
- In this work, we focus on the CFG constraint

Context-Free Grammars

Example (Correctly Bracketed Expressions)

- The following context-free grammar has 3 productions:

$$S \to \ell r,\ S \to SS,\ S \to \ell S r$$

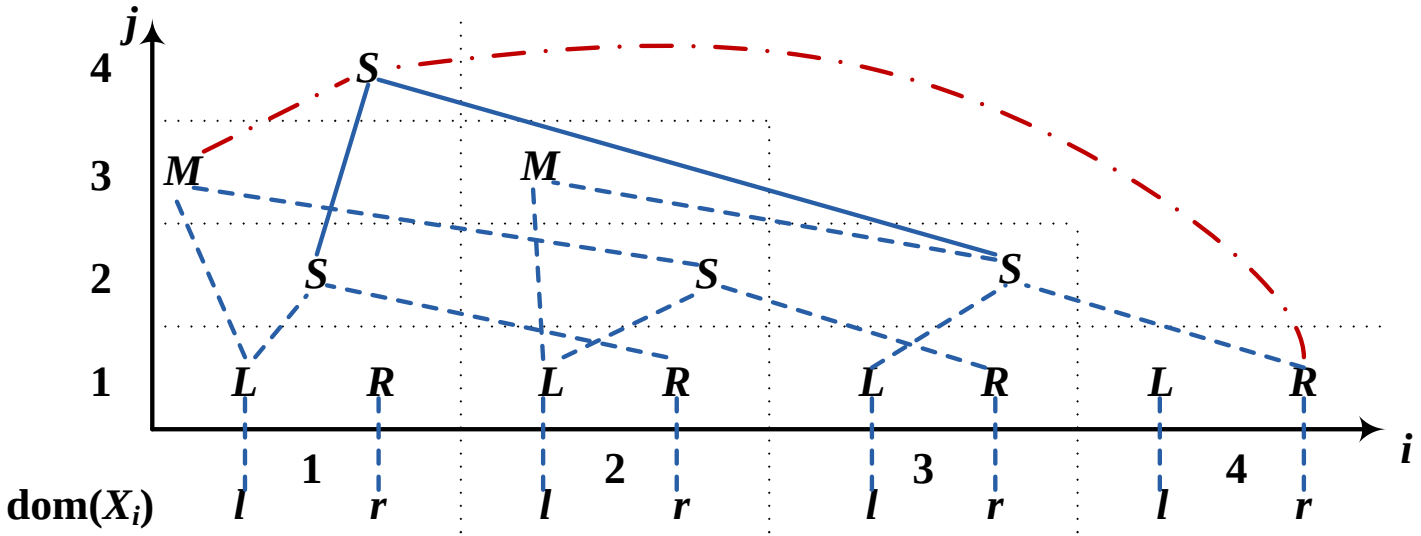
- Its Chomsky normal form G has 6 productions:

$$S \to LR,\ S \to SS,\ S \to MR,\ M \to LS,\ L \to \ell,\ R \to r$$

- The language specified by G contains two four-letter words:
  - $\ell r \ell r$, as $S \to SS \to LRS \to LRLR \to \cdots \to \ell r \ell r$
  - $\ell \ell r r$, as $S \to MR \to LSR \to LLRR \to \cdots \to \ell \ell r r$

Parsing a CFG for Word Length $n = 4$

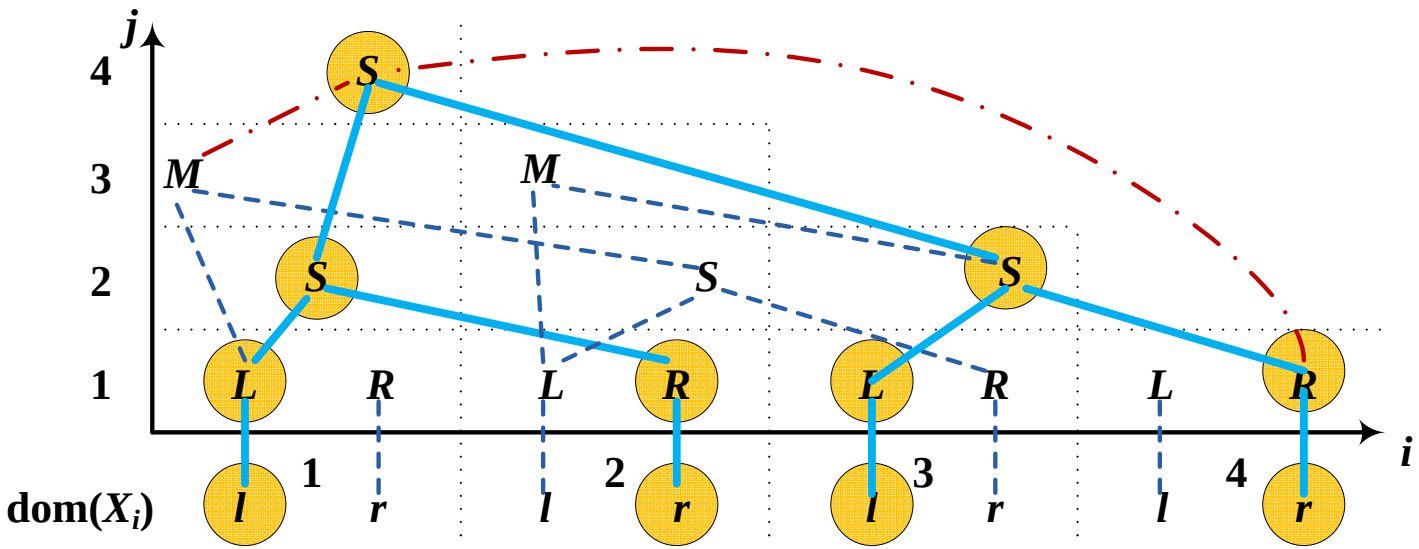
- Let $\langle X_1, \ldots, X_4 \rangle$ be a sequence of $n = 4$ decision variables
- $G: \; S \to LR, \; S \to SS, \; S \to MR, \; M \to LS, \; L \to \ell, \; R \to r$
- Use the Cocke-Younger-Kasami (CYK) dynamic program:



- A (non-)terminal is in the set of the table entry at column i and row j iff it can be generated by the word $X_i X_{i+1} \cdots X_{i+j-1}$
- For a (non-)terminal N, the downward arcs with the same line style denote a production to generate N

Parsing a CFG for Word Length $n = 4$

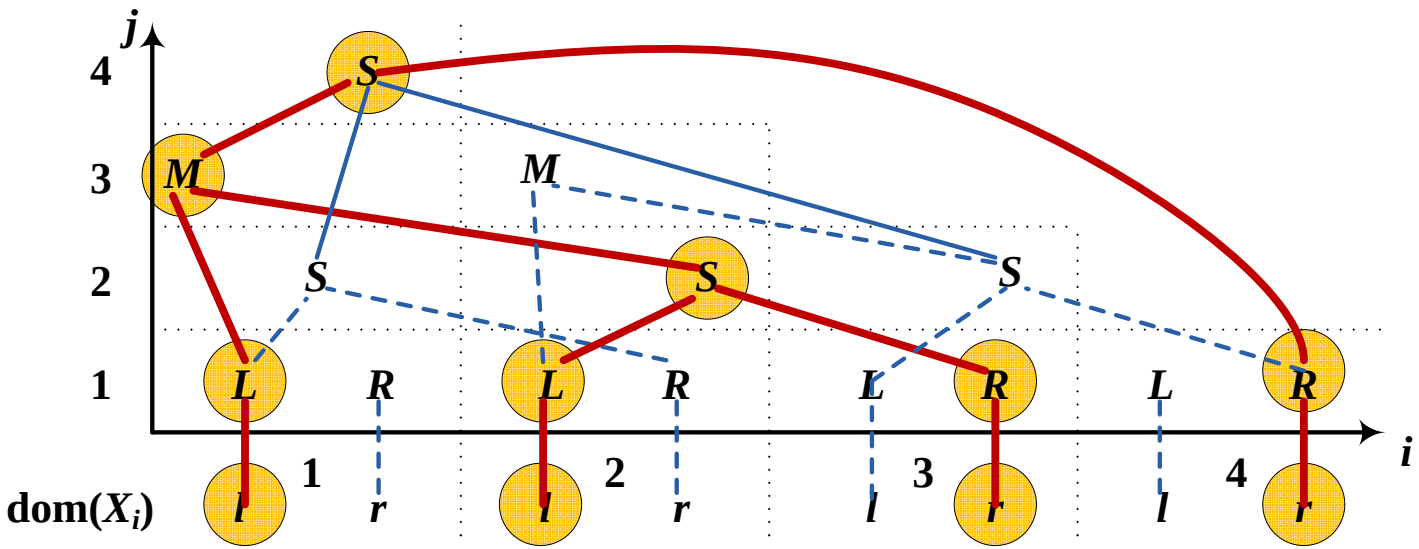
- $G: S \to LR,\ S \to SS,\ S \to MR,\ M \to LS,\ L \to \ell,\ R \to r$



- The parse graph contains two solutions: $\ell r \ell r$ and $\ell\ell rr$
- $S \to SS \to LRS \to LRLR \to \cdots \to \ell r \ell r$

Parsing a CFG for Word Length $n = 4$

- $G: S \to LR,\ S \to SS,\ S \to MR,\ M \to LS,\ L \to \ell,\ R \to r$

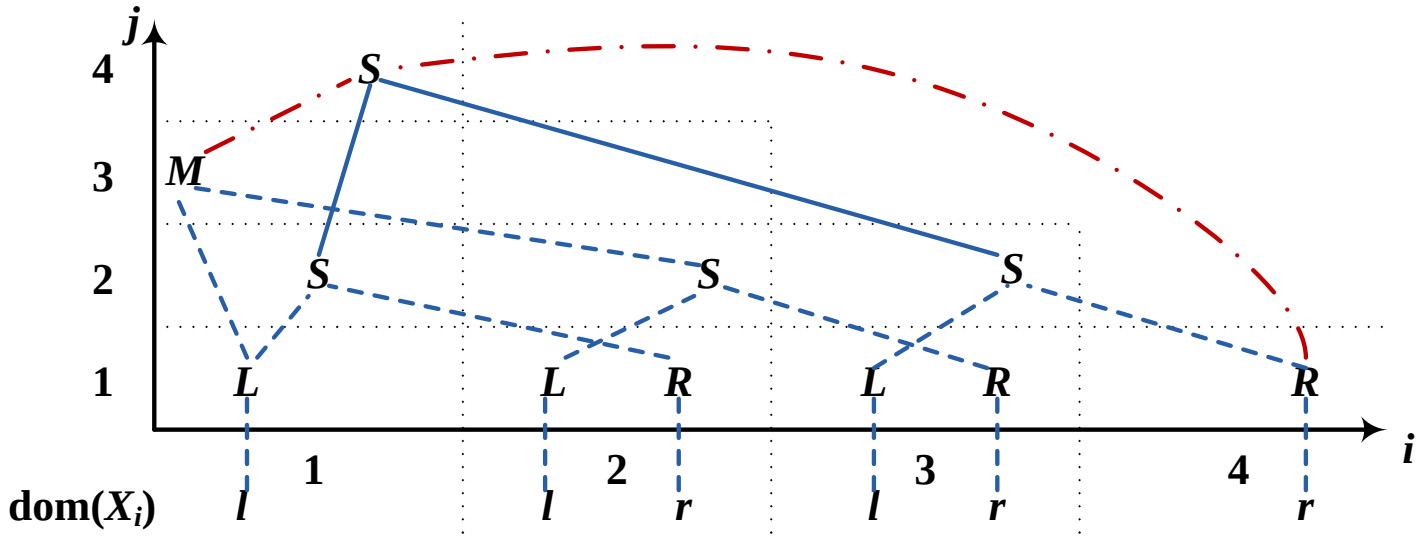


- The parse graph contains two solutions: $\ell r \ell r$ and $\ell\ell rr$
- $S \to MR \to LSR \to LLRR \to \cdots \to \ell\ell rr$

Propagating the $\mathbf{CFG}(X, G)$ Constraint

The CFG propagator of (Sellmann, 2006) uses the parse graph:

- The CYK algorithm constructs the parse graph
- A top-down process removes all (non-)terminals that cannot be reached from the root

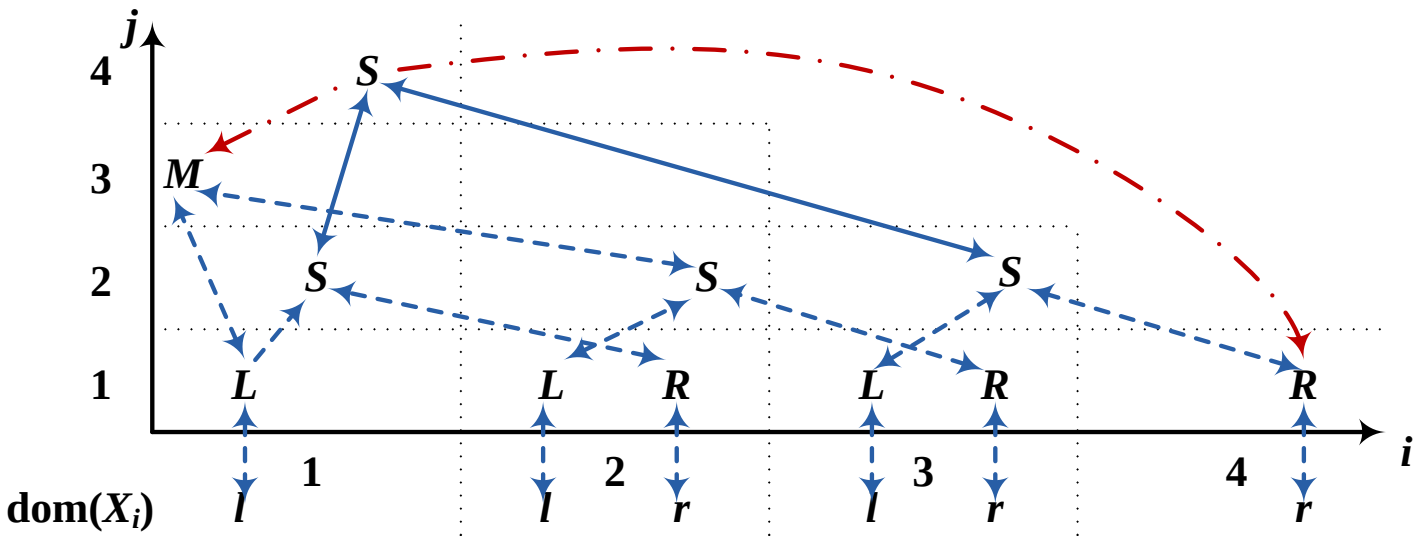


- **Complexity:** $O(n^3\,|G|)$ time & space, with $|X| = n$, for GAC
- **Problem:** Rebuilding the parse graph takes $O(n^3\,|G|)$ time.

Propagating the $\mathbf{CFG}(X, G)$ Constraint

The incremental propagator of (Kadıoğlu & Sellmann, 2008-2010) maintains an *over-approximation* of the parse graph in $O(n^2 |G|)$ space, as storing the actual parse graph takes $O(n^3 |G|)$ space:

- For each (non-)terminal, find the *first* low and high supports

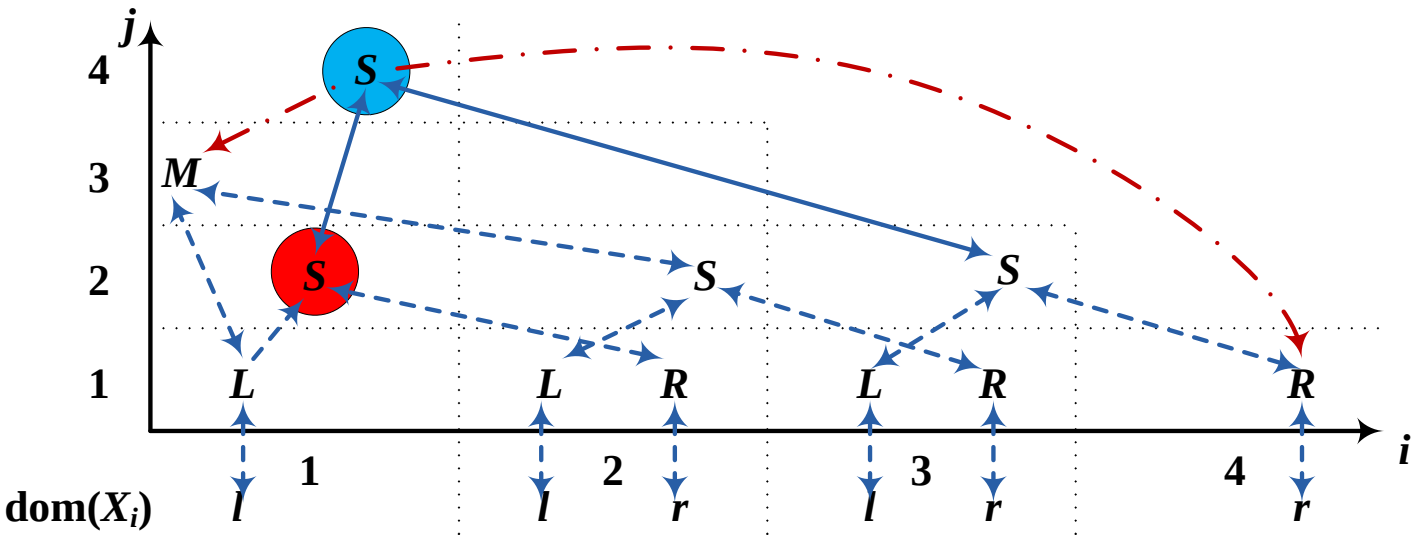


- Find the *next* support whenever the current support is lost
- **Complexity:** $O(n^3 |G|)$ time and $O(n^2 |G|)$ space. Optimal?

Propagating the $\mathbf{CFG}(X, G)$ Constraint

Problem: Exploration of false candidate supports.

- For example, if S is removed, then the next low support for S is found by exploring the following five candidate supports: $\langle S \to LR,\ 2\rangle$, $\langle S \to MR,\ 2\rangle$, $\langle S \to SS,\ 3\rangle$, $\langle S \to LR,\ 3\rangle$, and $\langle S \to MR,\ 3\rangle$
- Only $\langle S \to MR,\ 3\rangle$ is explored if the whole graph is stored.



Reformulating the $\mathbf{CFG}(X, G)$ Constraint

Reformulate a CFG(X, G) constraint (Katsirelos *et al.*, 2009):

- **Observation:** Every language of fixed-size words is regular
- **Idea:** Reformulate G for constant $|X| = n$ into a DFA A, and propagate the REGULAR(X, A) constraint in $O(n|A|)$ time
- **Feature:** The reformulation itself needs a CFG propagator
- **Problem:** What if the resulting DFA is huge? (e.g., forklift)

UPPSALA UNIVERSITET

Outline

1 Motivation

2 Constraints for Membership in Formal Languages

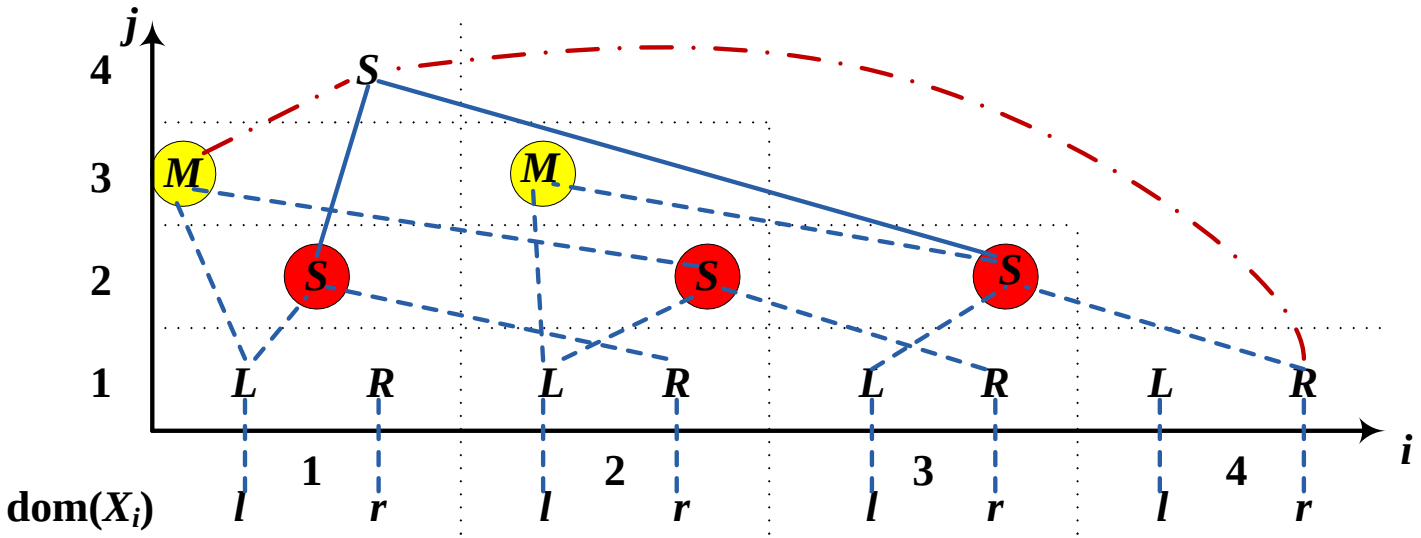
3 An Improved Propagator for the CFG Constraint

4 Experimental Evaluation

5 Conclusion and Future Work

Revisiting a CFG(X, G) Propagator

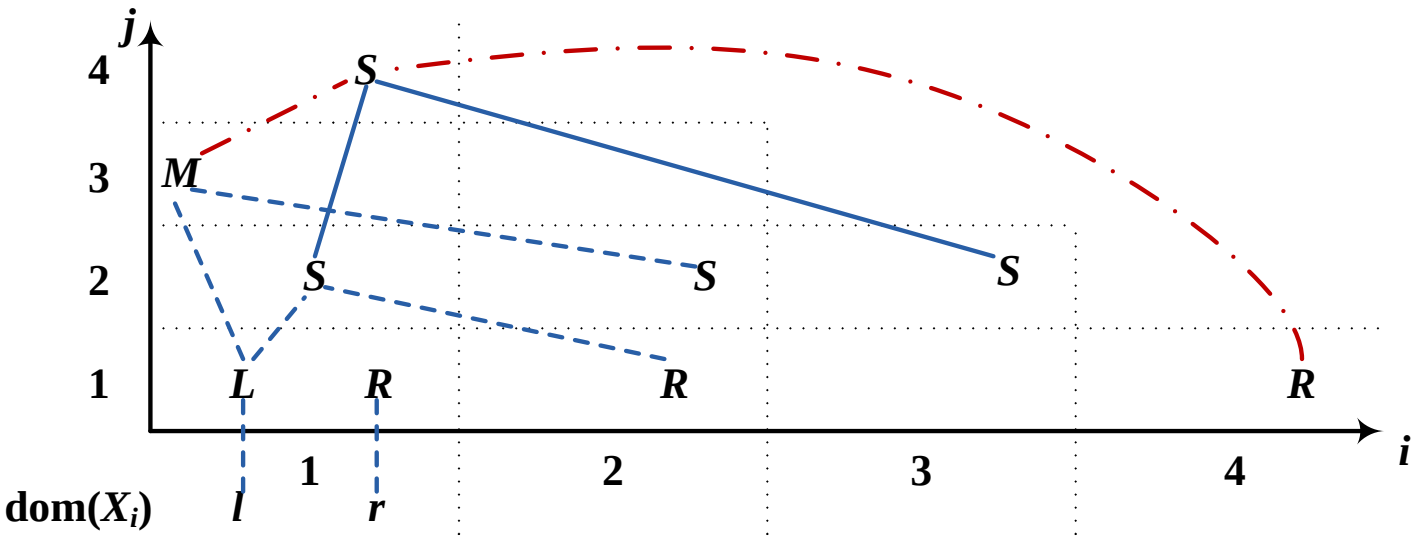
- **Motivation:** The problems of the previous methods
- **Idea:** Improve the incremental CFG propagator of (Kadıoğlu and Sellmann, 2008) by a much *tighter* over-approximation of the parse graph, by using an insight of (Kieżun *et al.*, 2009)



Our Improved CFG Propagator

First Idea: Without loss of generality, storing the *first* column of the CYK table suffices when all variable domains are the same

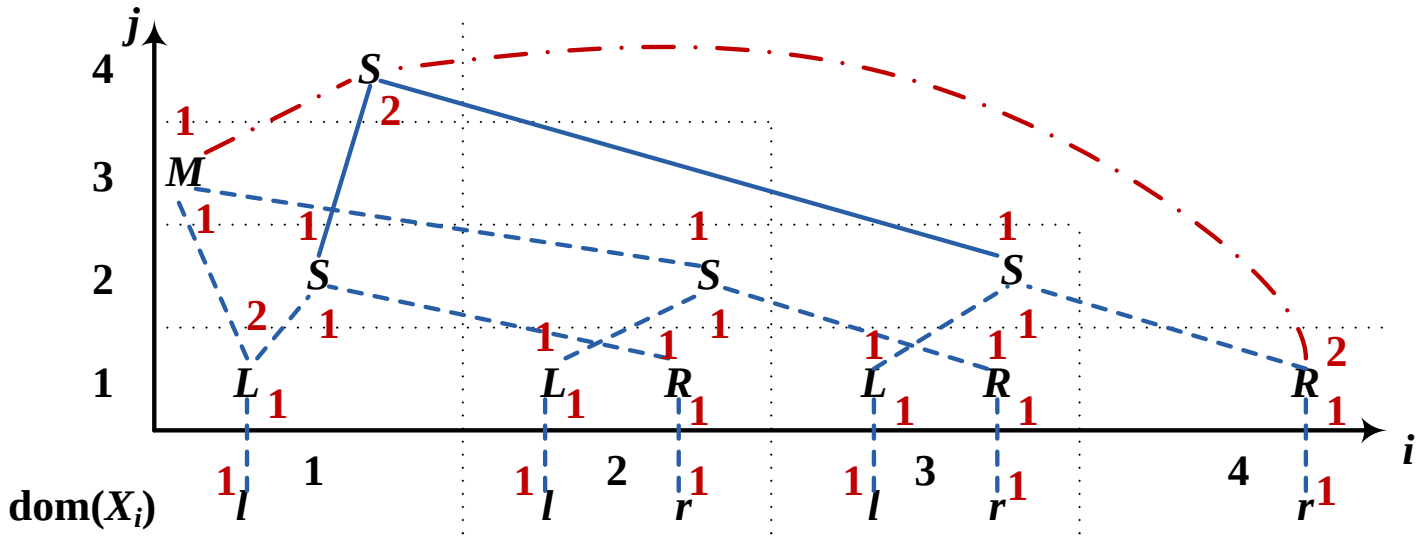
- It takes $O(n|G|)$ space to encode the CYK table
- It takes $O(n^2|G|)$ space to encode all the support sets
- There is no asymptotic space overhead compared to the $O(n^2|G| + |G|)$ propagator of (Kadıoğlu and Sellmann, 2008)



Our Improved CFG Propagator

Second Idea: Use support *counters* instead of tracing supports

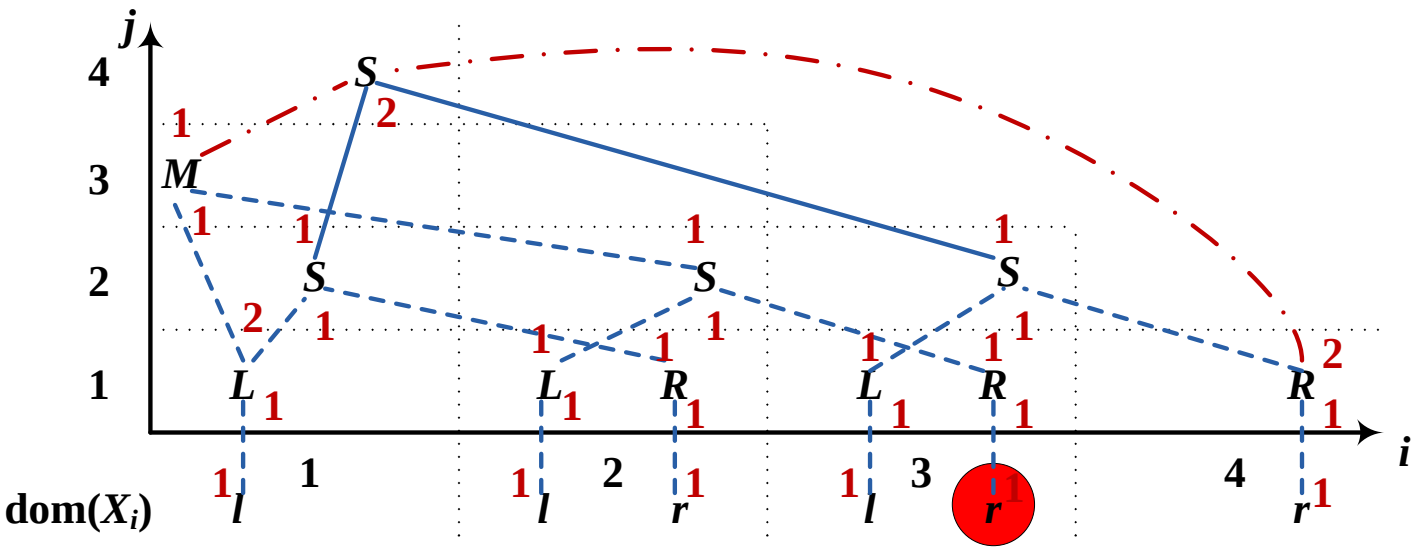
- Maintaining (candidate) support sets is slow
- Maintaining support counters is fast, plus $O(n^2\,|G|)$ space



Complexity: Also $O(n^3\,|G|)$ time and $O(n^2\,|G|)$ space. Optimal?

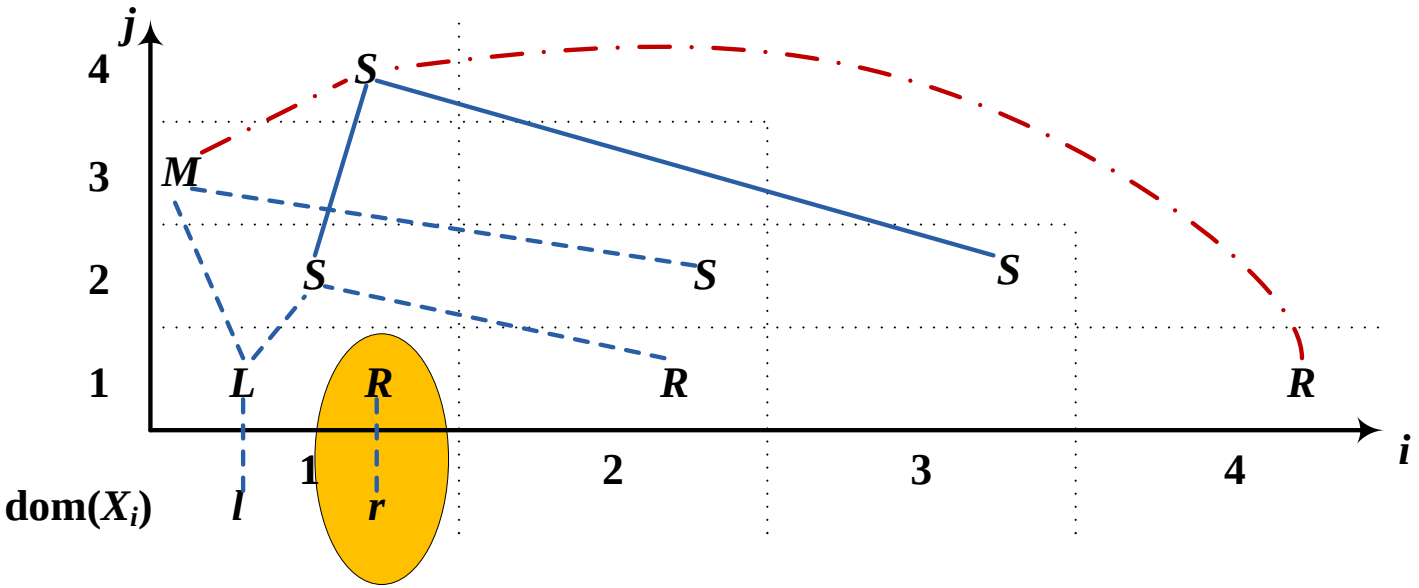
Our Improved CFG Propagator

Example: The right bracket r is removed from dom(X_3)



Our Improved CFG Propagator

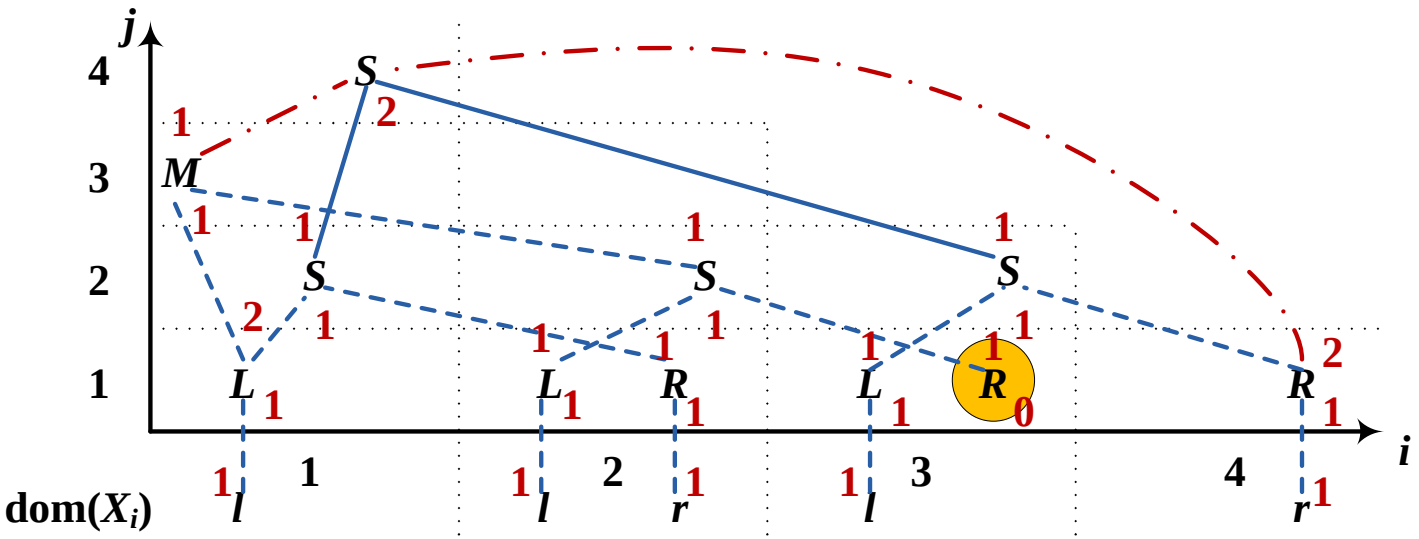
Example: The right bracket r is removed from dom(X_3)



In the over-approximation of the high-support set of r, we find R

Our Improved CFG Propagator

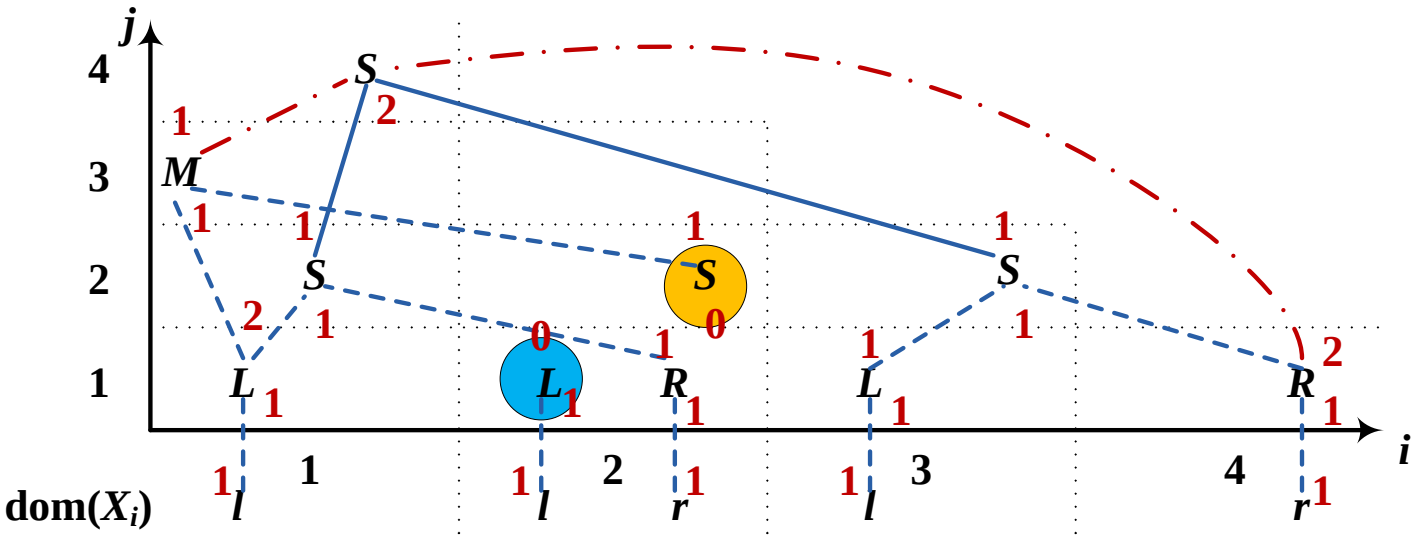
Example: The right bracket r is removed from dom(X_3)



Decrease the low-support counter of R: no low support now

Our Improved CFG Propagator

Example: The right bracket *r* is removed from dom(X_3)

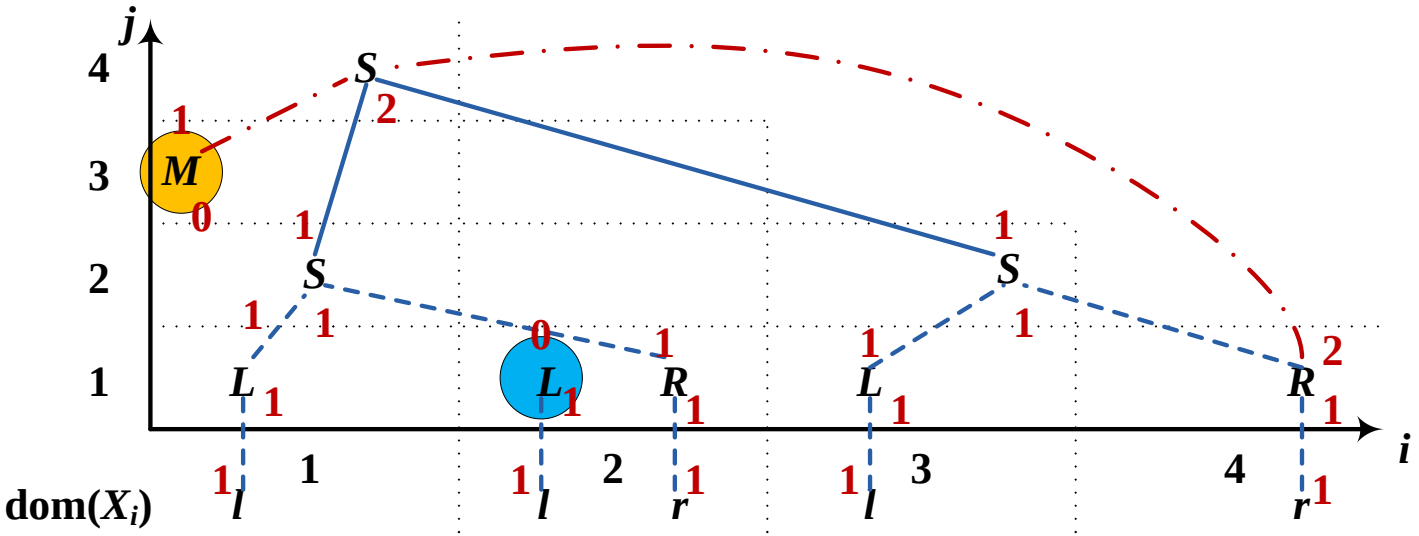


Decrease the low-support counter of *S*: no low support now

Decrease the high-support counter of *L*: no high support now

Our Improved CFG Propagator

Example: The right bracket *r* is removed from dom(X_3)

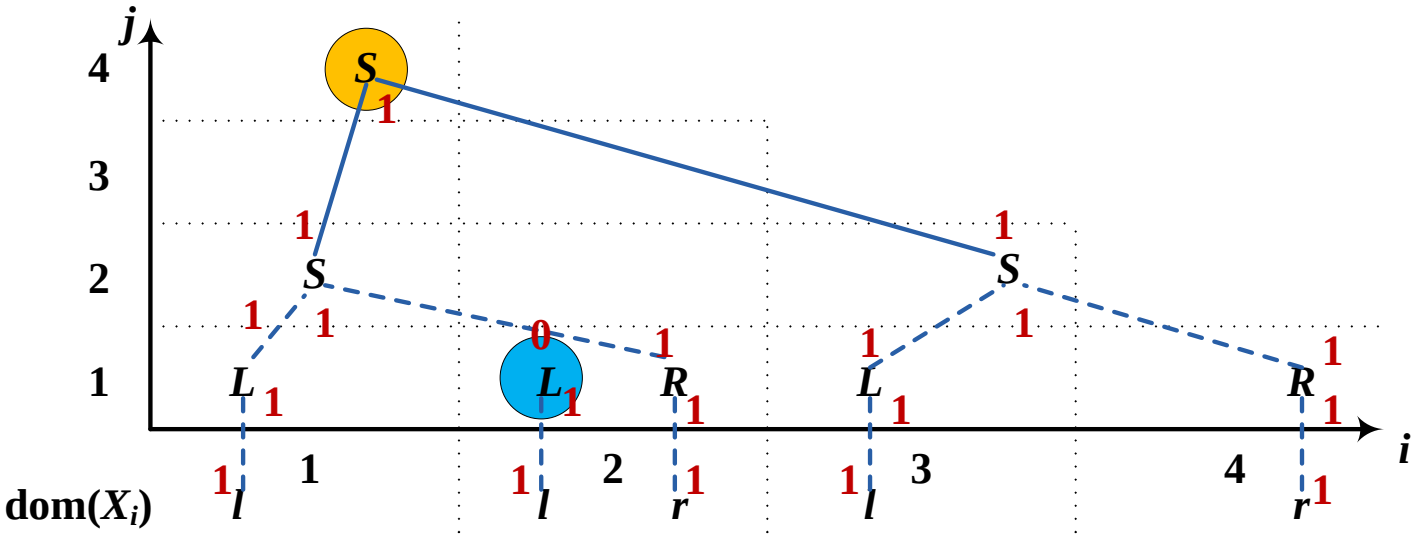


Decrease the low-support counter of *M*: no low support now

Remember: *L* has no high support anymore

Our Improved CFG Propagator

Example: The right bracket *r* is removed from dom(X_3)

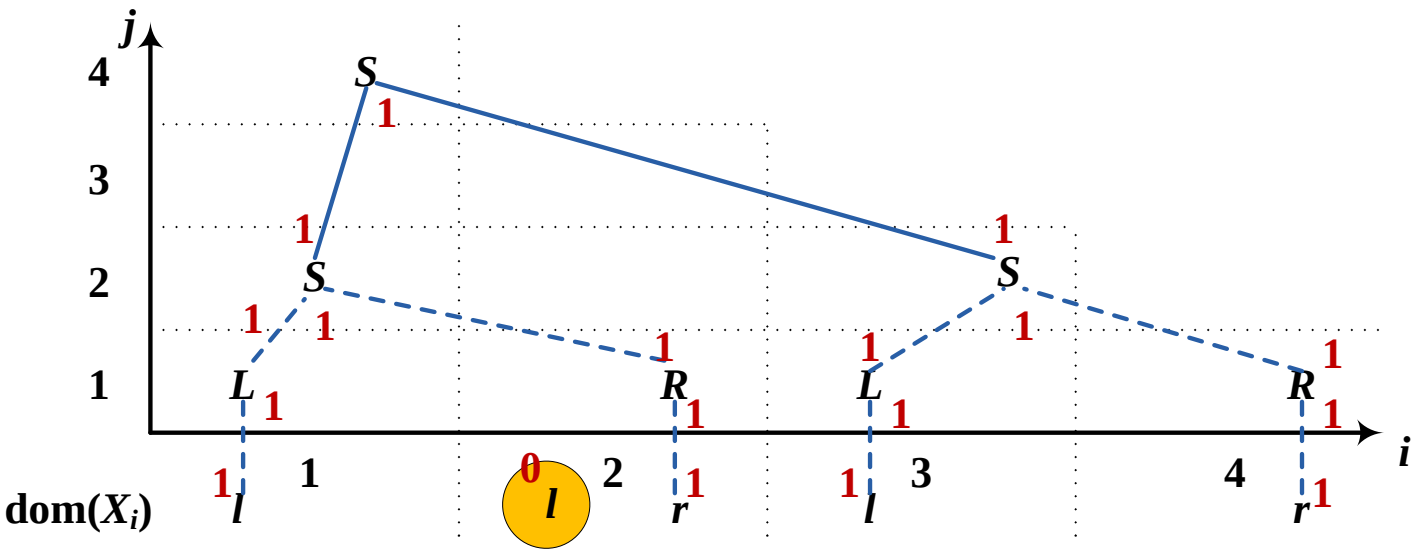


Decrease the low-support counter of *S*

Remember: *L* has no high support anymore

Our Improved CFG Propagator

Example: The right bracket r is removed from dom(X_3)



Decrease the high-support counter of ℓ: no high support now

Result: The left bracket ℓ is removed from dom(X_2)

UPPSALA UNIVERSITET

Outline

1 Motivation

2 Constraints for Membership in Formal Languages

3 An Improved Propagator for the CFG Constraint

4 Experimental Evaluation

5 Conclusion and Future Work

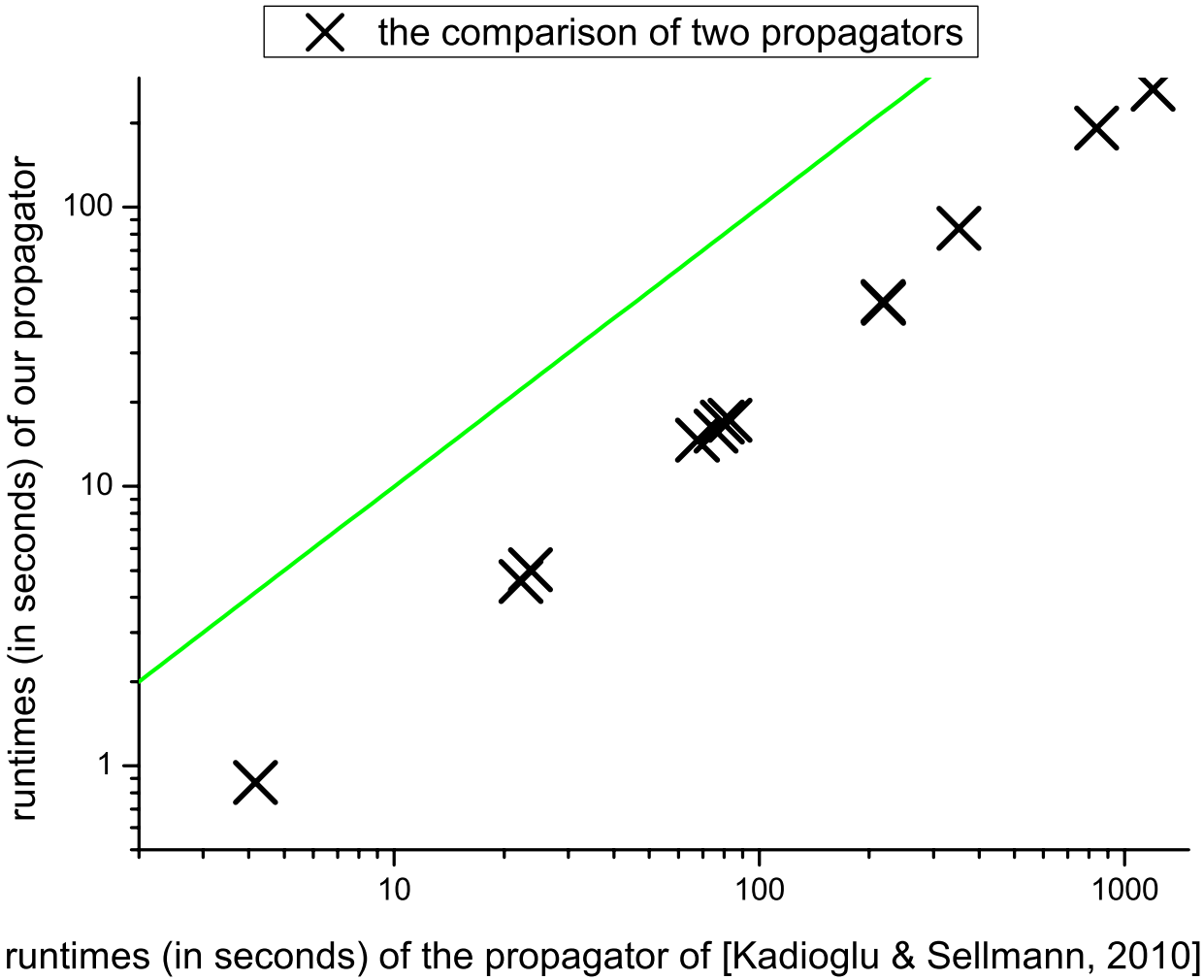
Shift Scheduling (Demassey *et al.*, 2006)

Comparison of CFG propagators under GECODE, for $n = 96$:

96 periods			search tree size			DFA	runtimes in seconds			
instance	a	w	#nodes	#propags	#fails	$\|A\|$	we	DFA_{we}	$\text{DFA}_{\text{[KS10]}}$	[KS10]
1_1	1	1	11	438	1	446	**0.24**	0.49	4.26	3.93
1_2	1	3	133	2123	33	998	**0.90**	3.78	15.38	12.87
1_3	1	4	349	5790	137	998	**1.68**	4.10	19.48	19.49
1_4	1	5	95	1836	7	814	**1.18**	2.41	21.99	20.53
1_5	1	4	71	1332	3	722	**0.92**	1.75	16.95	16.32
1_6	1	5	76	1567	3	722	**1.17**	2.01	21.16	20.17
1_7	1	6	3623	56635	1773	814	7.87	**2.97**	25.56	47.48
1_8	1	2	57	1005	10	998	**0.52**	3.59	10.76	8.47
1_9	1	1	19	460	1	630	**0.22**	0.80	4.41	3.94
1_10	1	7	12699	209988	6305	814	23.31	**4.02**	30.14	100.95
2_1	2	2	46	1414	8	984	**0.93**	1.69	16.76	15.97
2_5	2	4	83	2208	20	1209	**1.02**	3.15	18.51	16.41
2_6	2	5	89	1801	12	1207	**1.35**	2.94	23.03	21.57
2_7	2	6	258	5847	104	944	**1.97**	2.63	32.22	32.03
2_8	2	2	1046	28691	500	1774	**2.86**	7.75	23.09	24.09
2_9	2	1	35	1249	8	1460	**0.63**	4.11	14.21	11.03
2_10	2	7	4690	100007	2302	1506	**7.64**	7.82	43.24	53.90

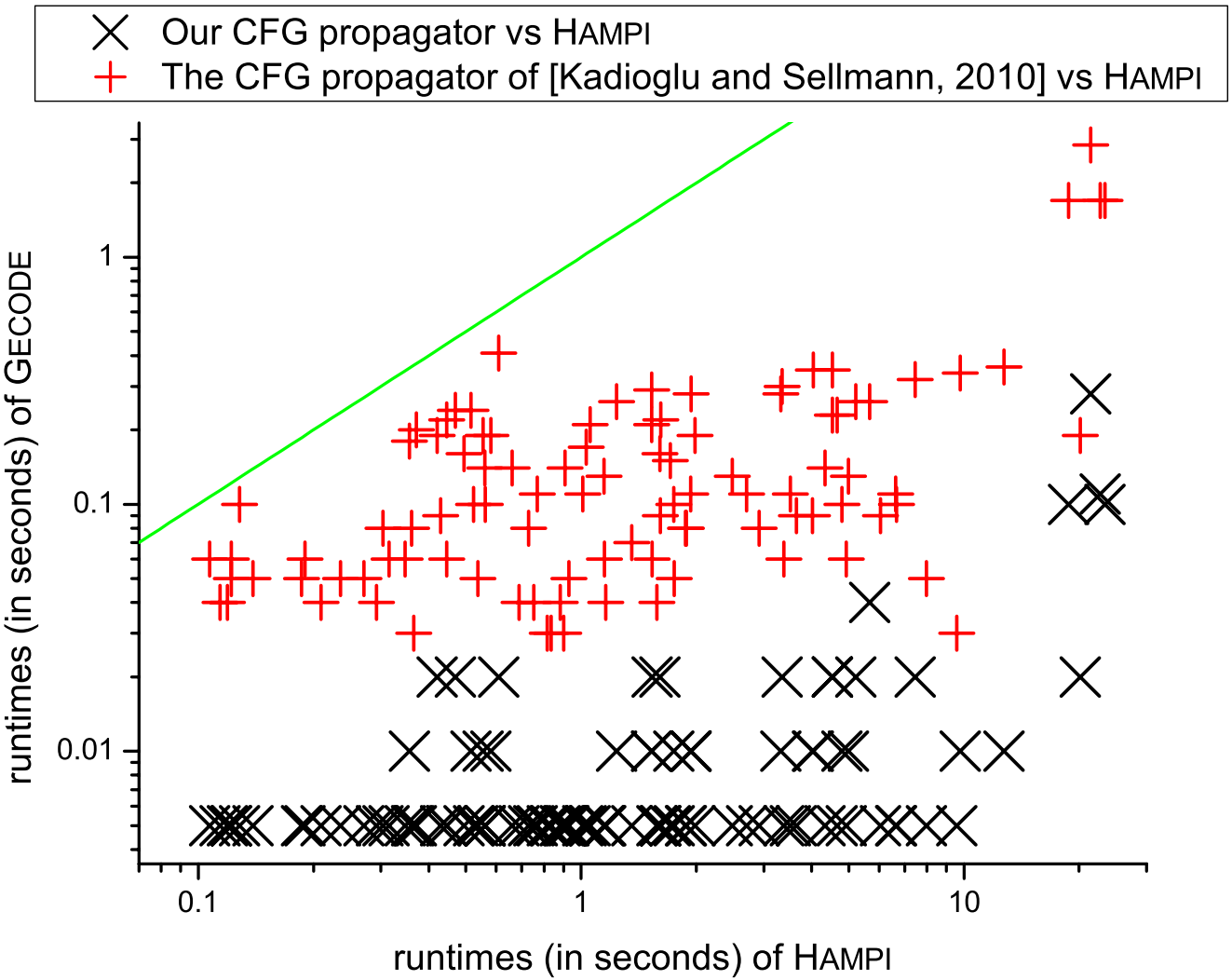
Forklift Scheduling (Gange & Stuckey, 2012)

Comparison of CFG propagators under GECODE, for $n \in [15, 20]$:



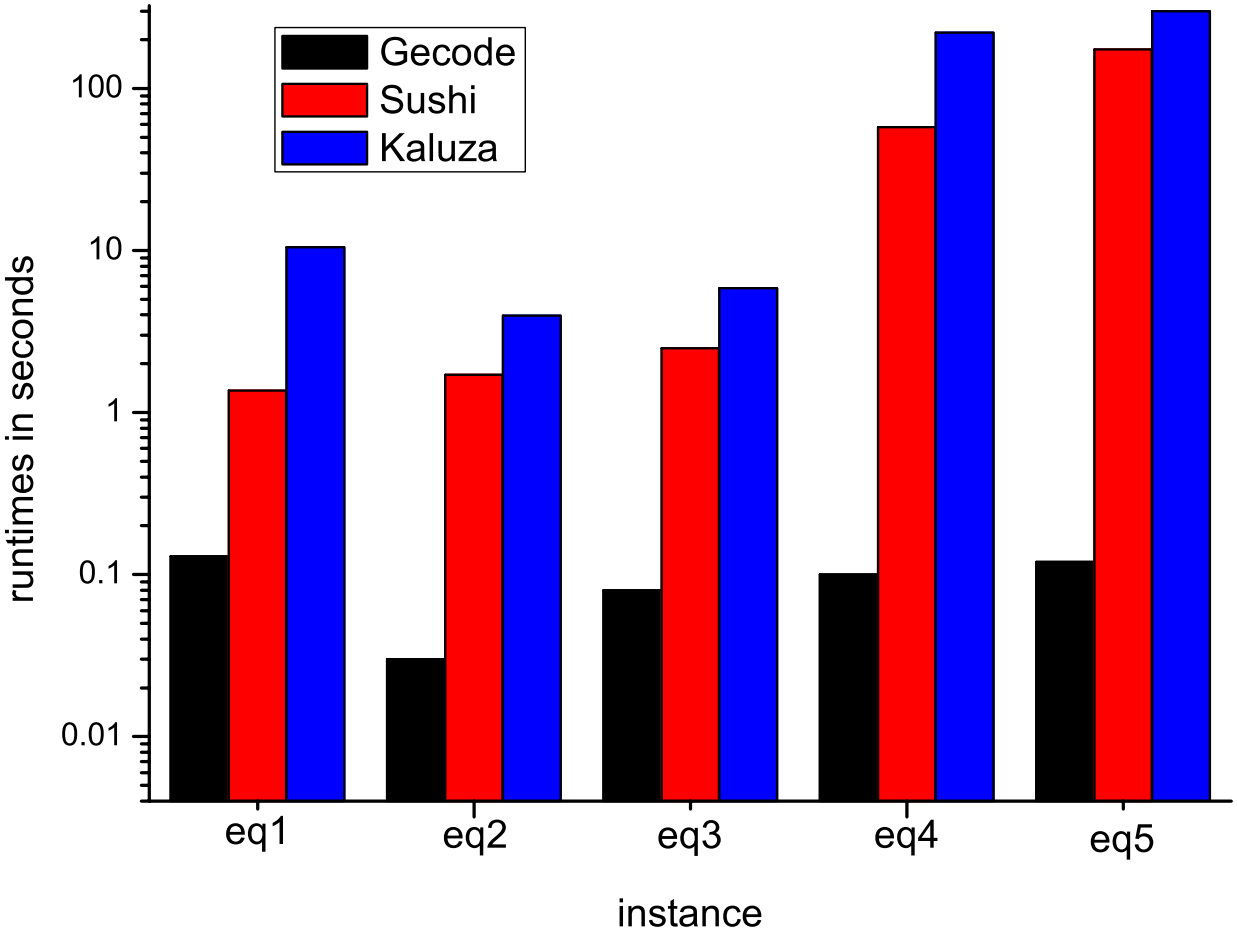
CFL Intersection (Kieżun *et al.*, 2009)

GECODE with the CFG constraint vs HAMPI, for $1 \leq n \leq 50$:



String Equations (Fu *et al.*, 2013)

GECODE with REGULAR constraint vs string solvers, for $n = 37$:



UPPSALA UNIVERSITET

Outline

Conclusion and Future Work

- **Result:** An improved CFG propagator achieving GAC: same (optimal?) complexity, but up to 250 times faster
- **Conjecture:** Useful idea for other CFG propagators?
- **Case for CP** for solving string constraints:
  - Verification benchmarks not previously known to be trivial
  - CP smoothly handles the integration of a cost function, the presence of non-membership constraints (length, GCC, etc), the handling of more than one string (with shared elements), etc
- **Future work:** CP for strings of (un)bounded length

UPPSALA UNIVERSITET

The End

THANKS FOR YOUR ATTENTION!

ANY QUESTIONS?

Special thanks to our sponsors:

- Swedish Research Council (VR): grants 2007-6445 and 2011-6133
- Computing Science Division, Info Tech Department, Uppsala University
- China Scholarship Council (CSC)