On Extending a Linear Tabling Framework To Support Batched Scheduling

Miguel Areias and Ricardo Rocha
CRACS & INESC-TEC LA
Faculty of Sciences, University of Porto, Portugal
miguel-areias@dcc.fc.up.pt *ricroc@dcc.fc.up.pt*

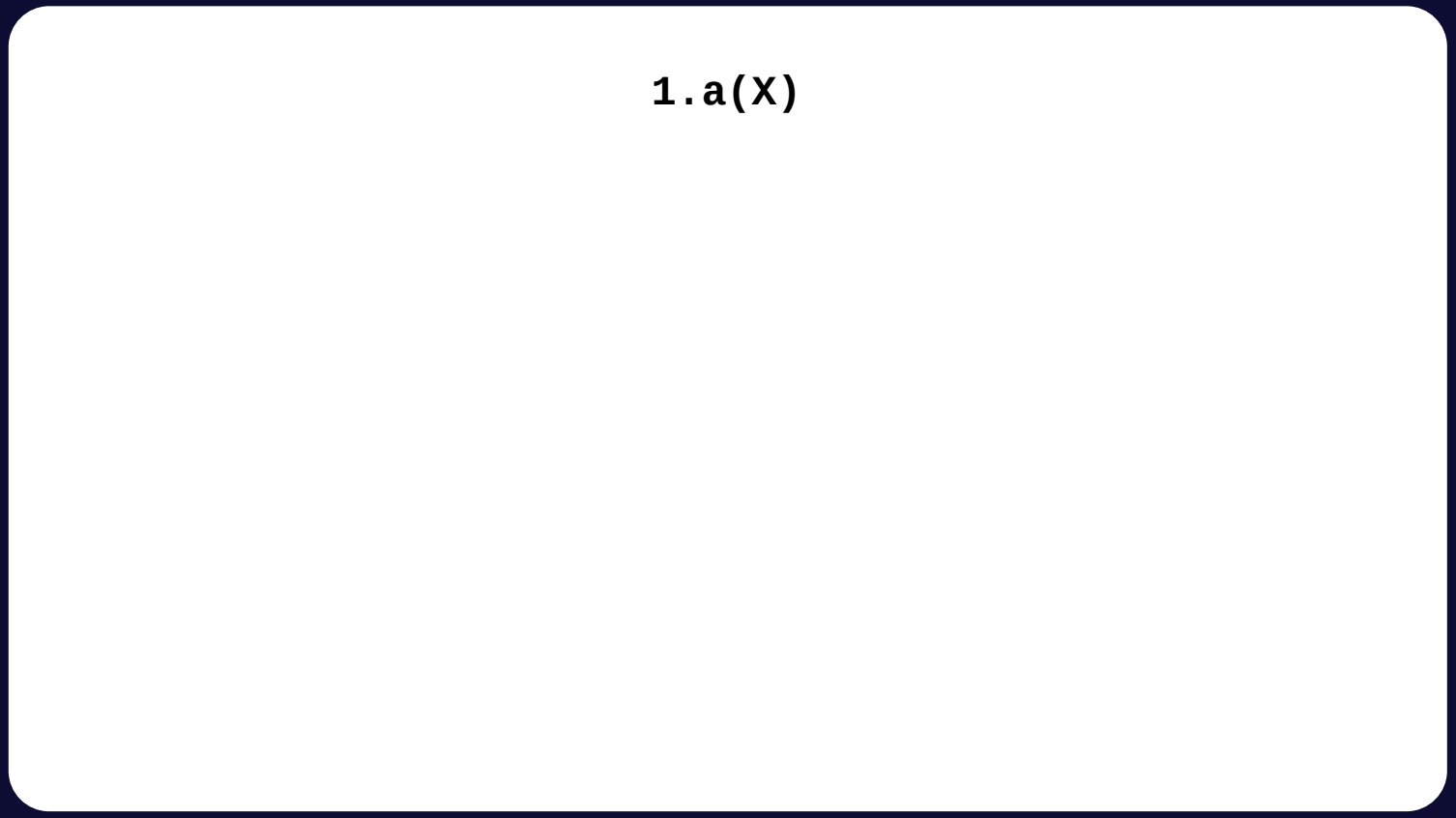
SLATE, Braga - Portugal, June 2012

Prolog and SLD Resolution

- Prolog systems are known to have good performances and flexibility, but they are based on SLD resolution, which limits the potential of the Logic Programing paradigm.
- SLD resolution cannot deal properly with the following situations:
  - **Positive Infinite Cycles** (insufficient expressiveness)
  - **Negative Infinite Cycles** (inconsistence)
  - **Redundant Computations** (inefficiency)

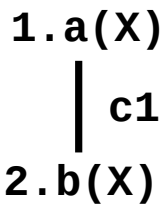
SLD Resolution: Infinite Cycles

```
c1)    a(X) :- b(X).
c2)    a(2).

c3)    b(X) :- a(X).
c4)    b(1).
```



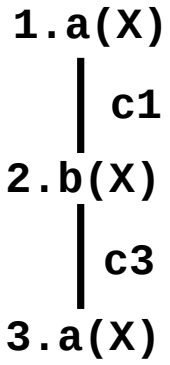
SLD Resolution: Infinite Cycles

```
c1)    a(X) :- b(X).
c2)    a(2).

c3)    b(X) :- a(X).
c4)    b(1).
```



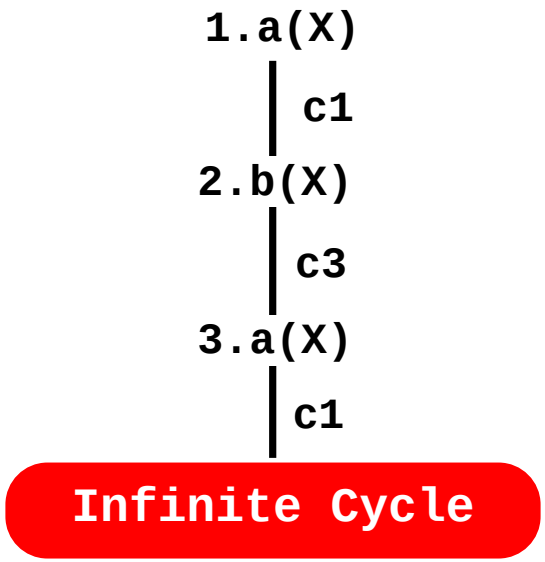
SLD Resolution: Infinite Cycles

```
c1)    a(X) :- b(X).
c2)    a(2).

c3)    b(X) :- a(X).
c4)    b(1).
```



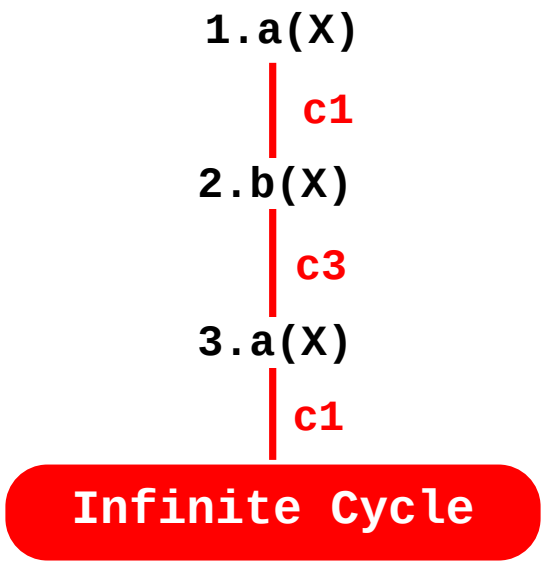
SLD Resolution: Infinite Cycles

```
c1)    a(X) :- b(X).
c2)    a(2).

c3)    b(X) :- a(X).
c4)    b(1).
```



SLD Resolution: Infinite Cycles

```
c1)    a(X) :- b(X).
c2)    a(2).

c3)    b(X) :- a(X).
c4)    b(1).
```



Tabling in Logic Programming

- **Tabling** is an implementation technique that overcomes some of the limitations of SLD resolution.
- Implementations of tabling are currently available in systems like XSB Prolog, **Yap Prolog**, B-Prolog, ALS-Prolog, Mercury and more recently Ciao Prolog.

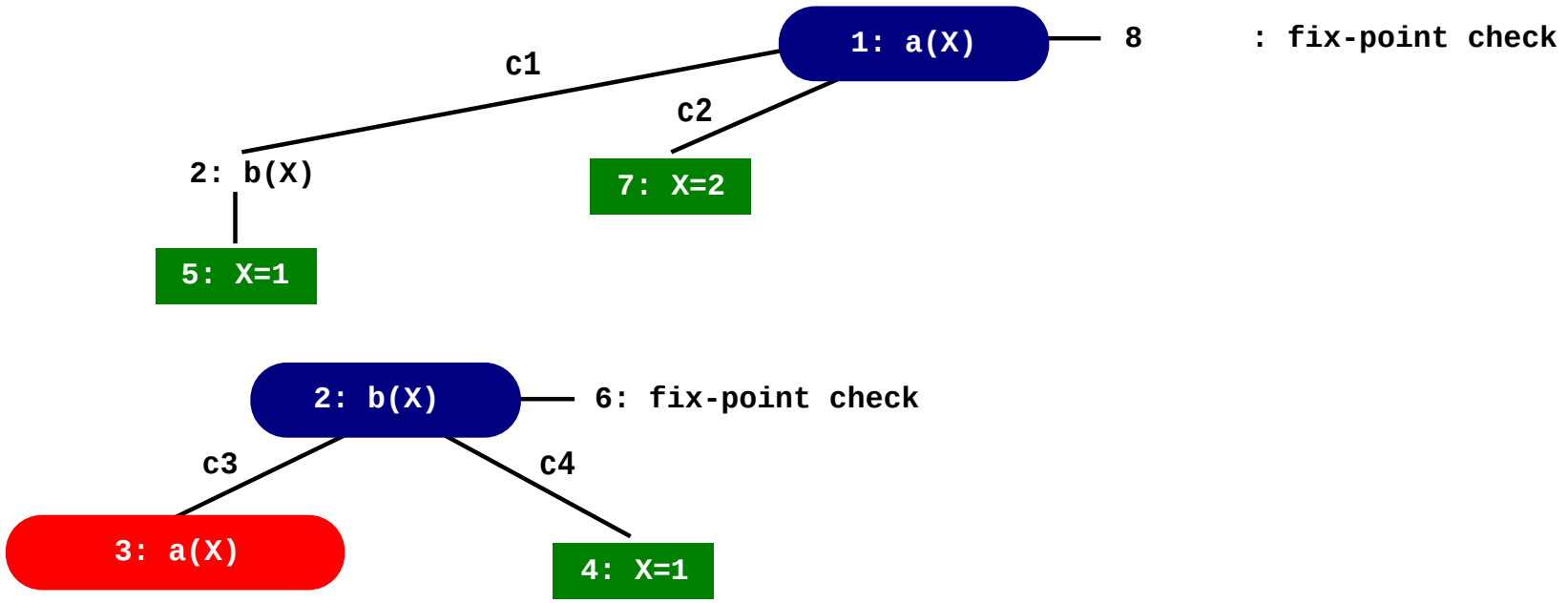
Tabling in Logic Programming

- **Tabling** is an implementation technique that overcomes some of the limitations of SLD resolution.
- Implementations of tabling are currently available in systems like XSB Prolog, **Yap Prolog**, B-Prolog, ALS-Prolog, Mercury and more recently Ciao Prolog.
- In these implementations, we can distinguish two main categories of tabling mechanisms:
  - **Suspension-Based Tabling**: can be seen as a sequence of sub-computations that can be suspended and later resumed, when necessary, to compute fix-points (XSB Prolog, **Yap Prolog**, Mercury and Ciao Prolog).
  - **Linear Tabling**: can be seen as a single execution tree where tabled subgoals use iterative computations, without requiring suspension and resumption, to compute fix-points (B-Prolog, ALS Prolog and **Yap Prolog**).

Linear Tabling Evaluation

```
        :- table a/1, b/1.

c1)     a(X) :- b(X).
c2)     a(2).
c3)     b(X) :- a(X).
c4)     b(1).
```

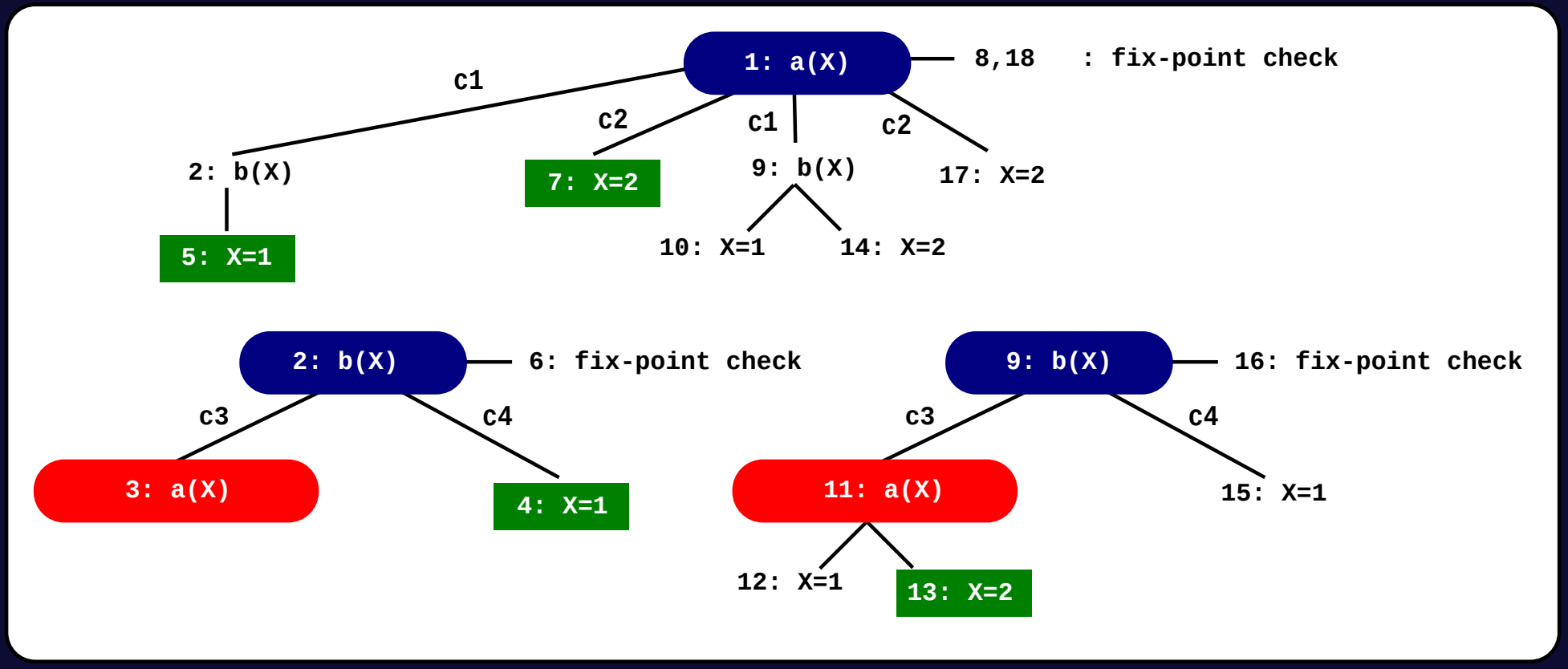
Call	Solutions
1: a(X)	5: X=1 7: X=2
2: b(X)	4: X=1



Linear Tabling Evaluation

```
        :- table a/1, b/1.

c1)     a(X) :- b(X).
c2)     a(2).
c3)     b(X) :- a(X).
c4)     b(1).
```

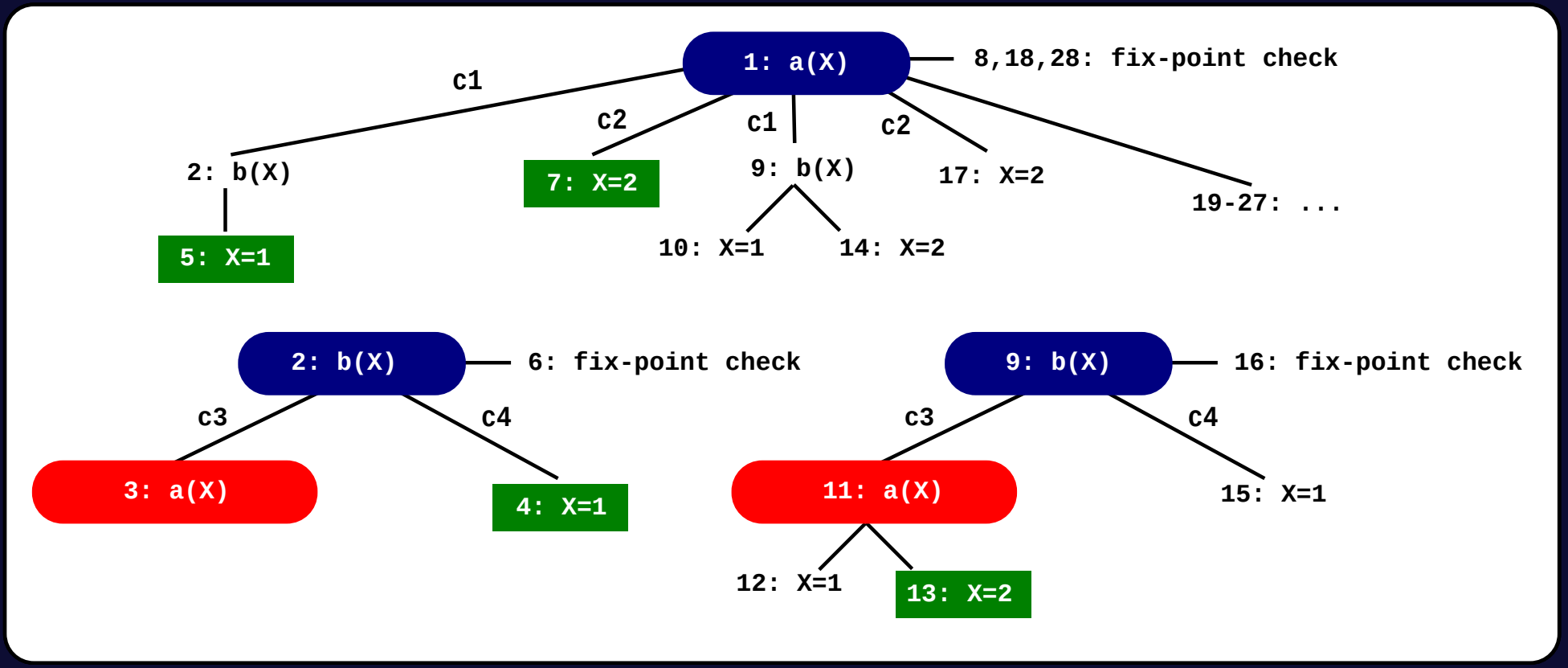
Call	Solutions
1: a(X)	5: X=1 7: X=2
2: b(X)	4: X=1 13: X=2



Linear Tabling Evaluation

```
        :- table a/1, b/1.

c1)     a(X) :- b(X).
c2)     a(2).
c3)     b(X) :- a(X).
c4)     b(1).
```

Call	Solutions
1: a(X)	5: X=1 7: X=2 28: complete
2: b(X)	4: X=1 13: X=2 28: complete



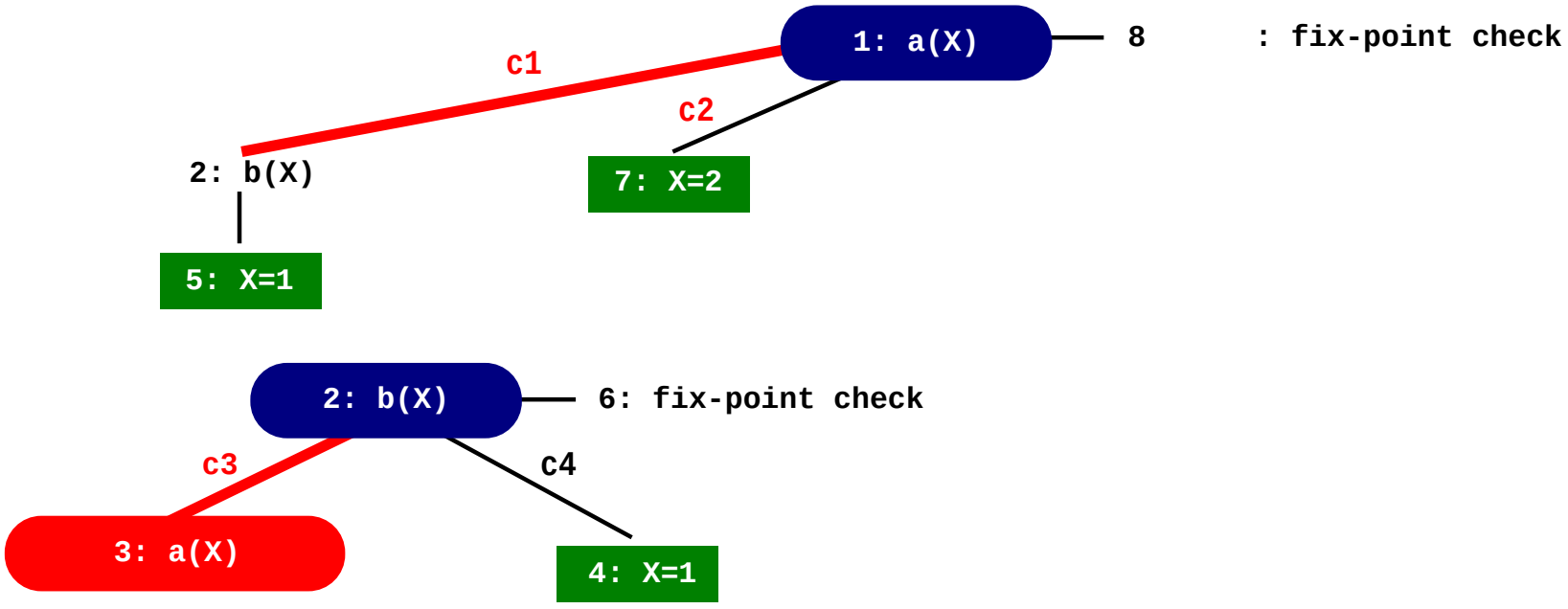
Linear Tabling

- The two most well-known linear tabling strategies are:
  - **DRA (Dynamic Reordering of Alternatives)**: tables not only the answers to tabled subgoals, but also the alternatives leading to repeated calls, the **looping alternatives**. It then uses the looping alternatives to repeatedly recompute them until reaching a fix-point (B-Prolog [actual version], ALS Prolog and **Yap Prolog**).
  - **DRE (Dynamic Reordering of Execution)**: repeated calls, **the followers**, execute from the backtracking point of the former call. A follower is then repeatedly re-executed, until all the available answers and clauses have been exhausted, that is, until a fix-point is reached (B-Prolog [original version] and **Yap Prolog**).

DRA Evaluation

```
        :- table a/1, b/1.

c1)     a(X) :- b(X).
c2)     a(2).
c3)     b(X) :- a(X).
c4)     b(1).
```

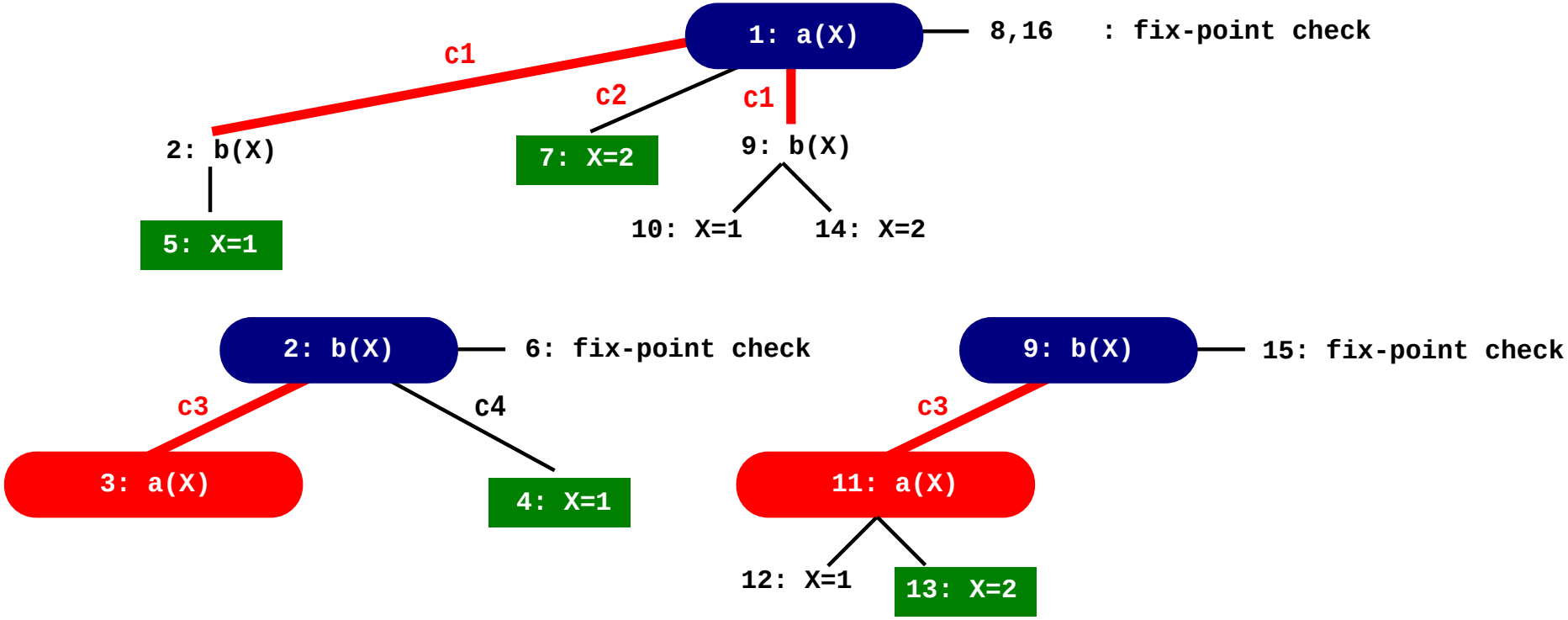
Call	Solutions	Looping Alternatives
1: a(X)	5: X=1 7: X=2	3: c1
2: b(X)	4: X=1	3: c3



DRA Evaluation

```
        :- table a/1, b/1.

c1)     a(X) :- b(X).
c2)     a(2).
c3)     b(X) :- a(X).
c4)     b(1).
```

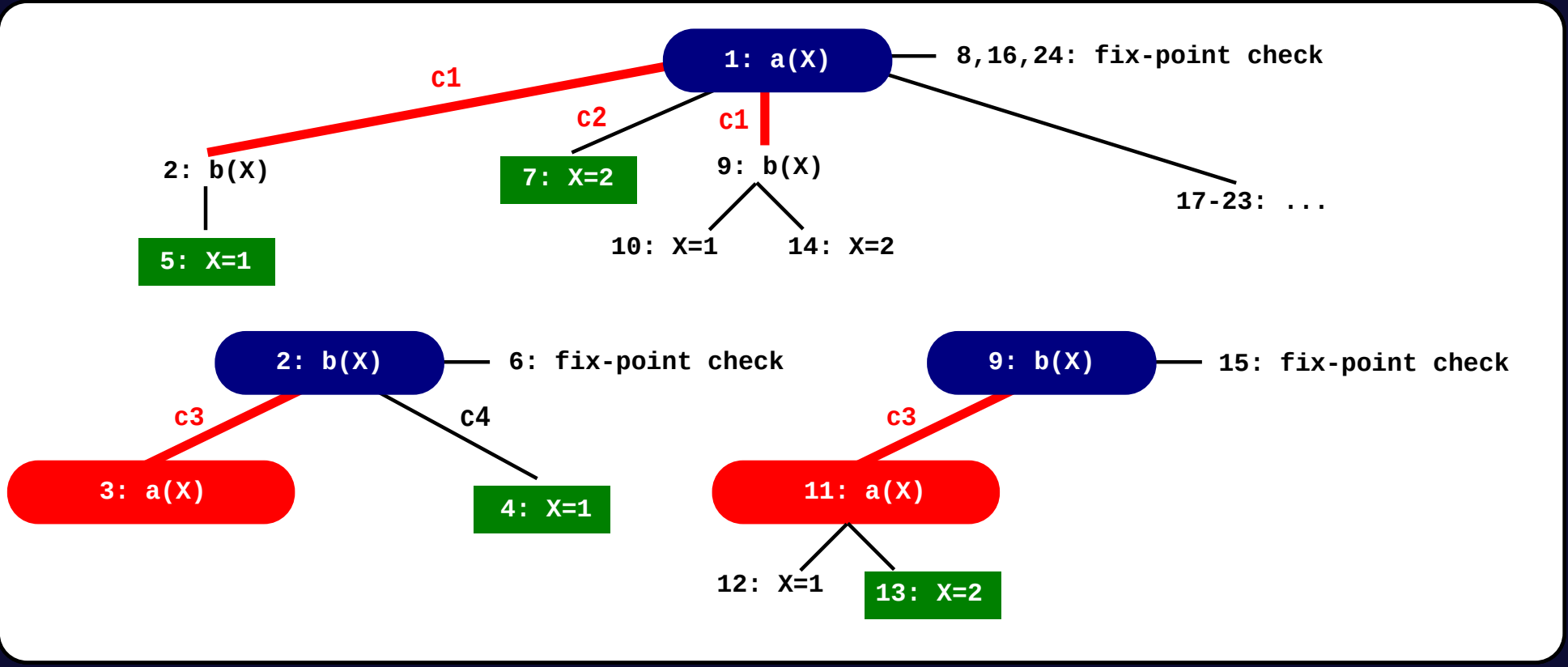
Call	Solutions	Looping Alternatives
1: a(X)	5: X=1 7: X=2	3: c1
2: b(X)	4: X=1 13: X=2	3: c3



DRA Evaluation

```
        :- table a/1, b/1.

c1)     a(X) :- b(X).
c2)     a(2).
c3)     b(X) :- a(X).
c4)     b(1).
```

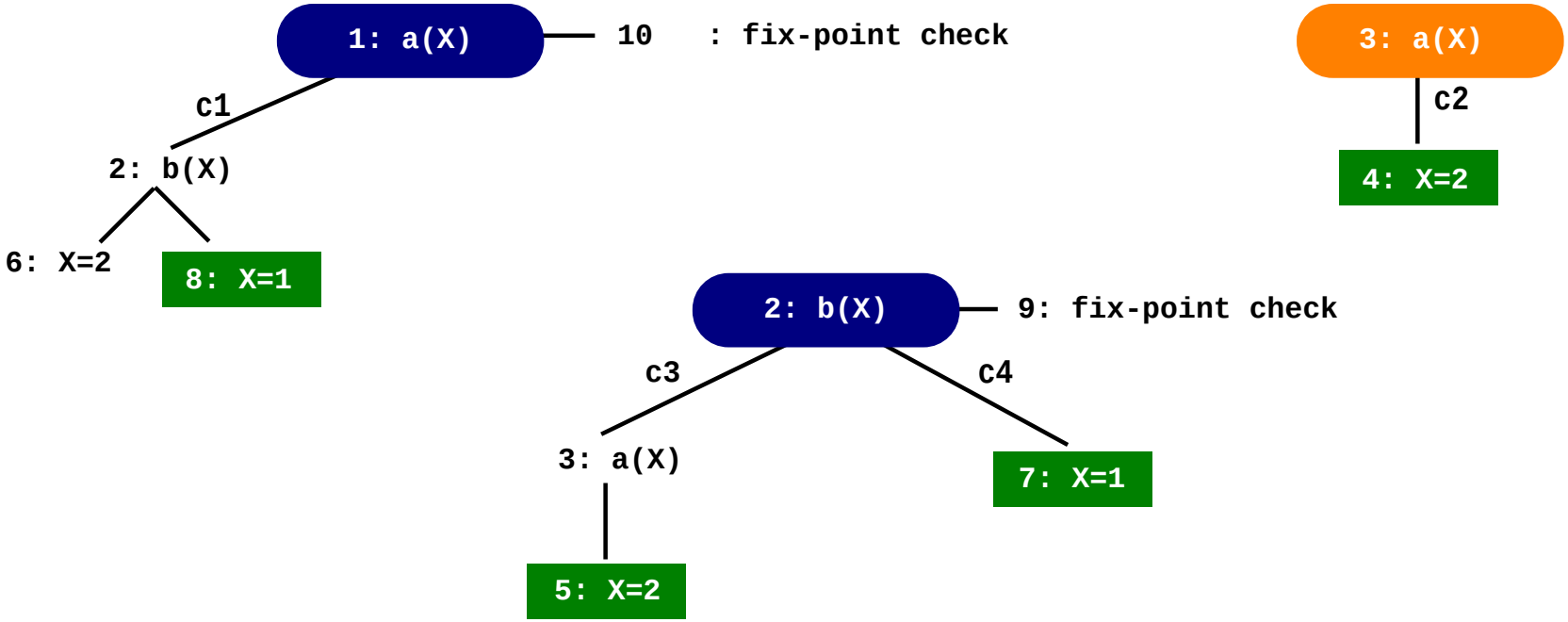
Call	Solutions	Looping Alternatives
1: a(X)	5: X=1 7: X=2 24: complete	3: c1
2: b(X)	4: X=1 13: X=2 24: complete	3: c3



DRE Evaluation

```
        :- table a/1, b/1.

c1)     a(X) :- b(X).
c2)     a(2).
c3)     b(X) :- a(X).
c4)     b(1).
```

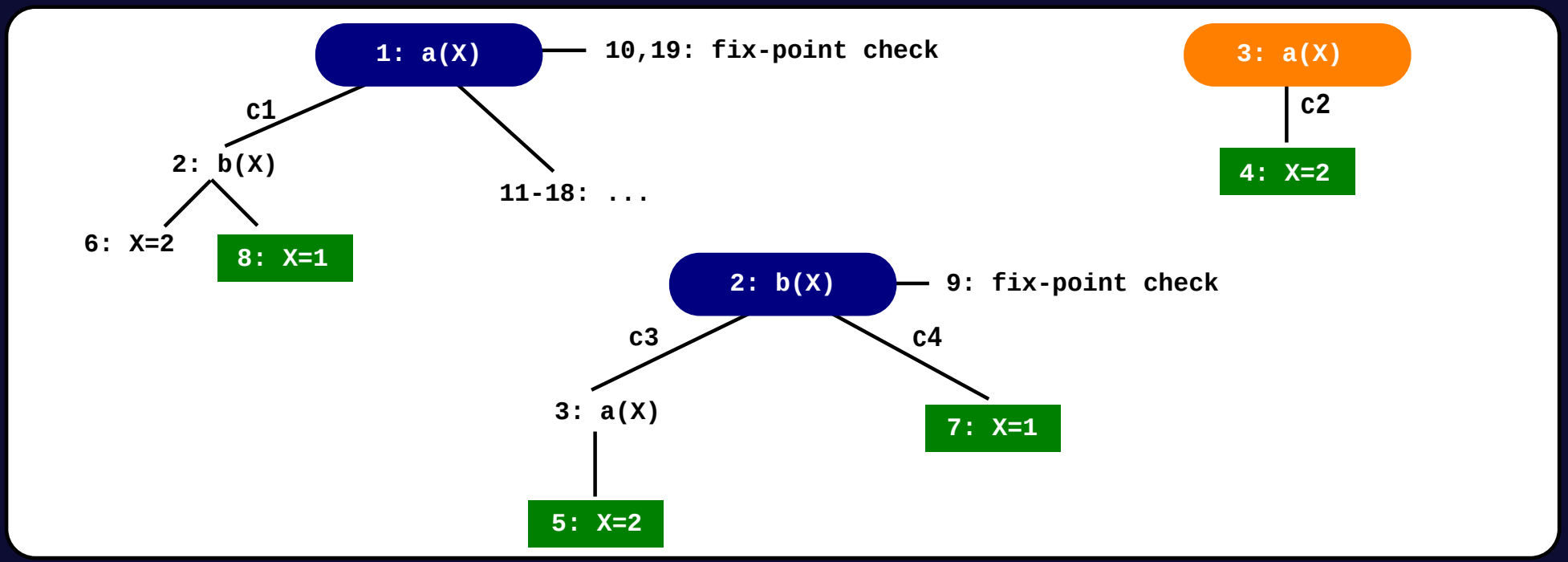
Call	Solutions
1: a(X)	4: X=2 8: X=1
2: b(X)	5: X=2 7: X=1



DRE Evaluation

```
        :- table a/1, b/1.

c1)     a(X) :- b(X).
c2)     a(2).
c3)     b(X) :- a(X).
c4)     b(1).
```

Call	Solutions
1: a(X)	4: X=2 8: X=1 19: complete
2: b(X)	5: X=2 7: X=1 19: complete



Tabling Scheduling Strategies

- The two most successful tabling scheduling strategies are **Local** e **Batched**. They define the behavior of the tabling mechanism whenever it finds a **new answer** for a tabled call.

Scheduling	New Answer	Fix-Point Detection
Local	**Fail** the computation to the **current** choice point.	**Consume all the answers**, propagating them to the context of the **previous** call.
Batched	**Consume the answer**, propagating it immediately to the context of the **previous** call.	**Fail** the computation to the **previous** choice point, since the actual was already fully evaluated.

Some Implementation Details

- Our goal with this work is to:
  - Propose a new linear tabling framework that supports the **combination of DRA with DRE** using batched scheduling. For that we have extended:
    - The **trie** structures inside the table space.

Some Implementation Details

- Our goal with this work is to:
  - Propose a new linear tabling framework that supports the **combination of DRA with DRE** using batched scheduling. For that we have extended:
    - The **trie** structures inside the table space.
    - The **tabled_call** operation to support the consumption of solutions before executing the program clauses on **non leader subgoal calls**.

Some Implementation Details

- Our goal with this work is to:
  - Propose a new linear tabling framework that supports the **combination of DRA with DRE** using batched scheduling. For that we have extended:
    - The **trie** structures inside the table space.
    - The **tabled_call** operation to support the consumption of solutions before executing the program clauses on **non leader subgoal calls**.
    - The **new_solutions** operation to support the immediate propagation of new solutions.

Some Implementation Details

- Our goal with this work is to:
  - Propose a new linear tabling framework that supports the **combination of DRA with DRE** using batched scheduling. For that we have extended:
    - The **trie** structures inside the table space.
    - The **tabled_call** operation to support the consumption of solutions before executing the program clauses on **non leader subgoal calls**.
    - The **new_solutions** operation to support the immediate propagation of new solutions.
    - The **fix-point_check** operation to propagate the solutions to outside the **leader subgoal calls**.

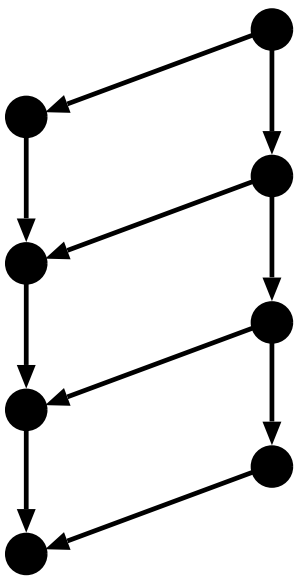
Some Implementation Details

- Our goal with this work is to:
  - Propose a new linear tabling framework that supports the **combination of DRA with DRE** using batched scheduling. For that we have extended:
    - The **trie** structures inside the table space.
    - The **tabled_call** operation to support the consumption of solutions before executing the program clauses on **non leader subgoal calls**.
    - The **new_solutions** operation to support the immediate propagation of new solutions.
    - The **fix-point_check** operation to propagate the solutions to outside the **leader subgoal calls**.
  - Analyze the **advantages and weaknesses** of each strategy, when used **solely or combined** with the other.

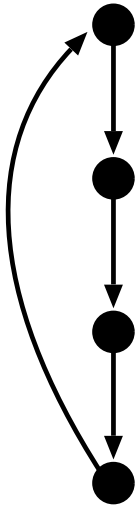
Experimental Results

```
path(X,Z) :- edge(X,Y) , path(Y,Z).
path(X,Z) :- edge(X,Z).
```

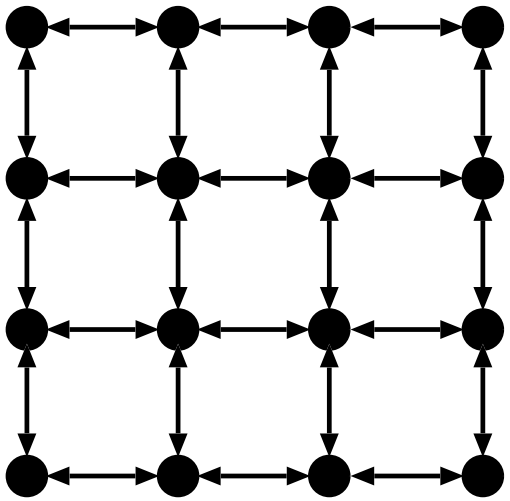
```
path(X,Z) :- sld1 , edge(X,Y) , path(Y,Z) , sld2.
path(X,Z) :- sld3 , edge(X,Z) , sld4.
```



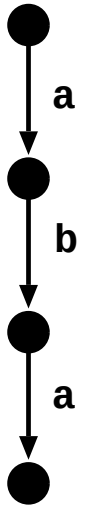
Pyramid (depth 4)



Cycle (depth 4)



Grid (depth 4)



Warren (depth 4)

Experimental Results

Execution Time: Path Right - Pyramid

Depth	Std	DRA		DRE		DRA+DRE		B-Prolog		YapTab	
1000	983	526	**(1.87)**	1,102	(0.89)	658	(1.49)	948	(1.04)	517	(1.90)
2000	3,897	2,071	**(1.88)**	4,380	(0.89)	2,611	(1.49)	5,630	(0.69)	2,013	(1.94)
3000	9,043	4,740	**(1.91)**	10,110	(0.89)	5,920	(1.53)	——	(*n.c.*)	4,561	(1.98)
	Average		**(1.89)**		(0.89)		(1.50)		(0.86)		(1.94)

Number of Executions: Path Right - Pyramid 2000

sld#	Std	DRA	DRE	DRA+DRE	B-Prolog	YapTab
sld1	7,999	**2.00**	1.00	**2.00**	**2.00**	2.00
sld2	37,951,017	**2.38**	0.86	1.73	**2.38**	2.38
sld3	7,999	**2.00**	1.00	**2.00**	**2.00**	2.00
sld4	23,988	**2.00**	1.00	**2.00**	**2.00**	2.00

Experimental Results

Execution Time: Path Right - Cycle

Depth	Std	DRA		DRE		DRA+DRE		B-Prolog		YapTab	
1000	687	539	**(1.27)**	713	(0.96)	563	(1.22)	540	**(1.27)**	362	(1.89)
2000	2,793	2,198	**(1.27)**	2,891	(0.97)	2,286	(1.22)	3,079	(0.91)	1,534	(1.82)
3000	6,048	4,681	**(1.29)**	6,343	(0.95)	4,949	(1.22)	8,678	(0.70)	2,956	(2.05)
	Average		**(1.28)**		(0.96)		(1.22)		(0.96)		(1.92)

Number of Executions: Path Right - Cycle 2000

sld#	Std	DRA	DRE	DRA+DRE	B-Prolog	YapTab
sld1	6,002	1.00	1.00	1.00	1.00	3.00
sld2	18,003,000	**1.29**	1.00	**1.29**	**1.29**	2.25
sld3	6,002	**3.00**	1.00	**3.00**	**3.00**	3.00
sld4	10,000	**2.50**	1.00	**2.50**	**2.50**	2.50

Experimental Results

Execution Time: Path Right - Grid

Depth	Std	DRA		DRE		DRA+DRE		B-Prolog		YapTab	
20	221	166	**(1.33)**	227	(0.97)	174	(1.27)	202	(1.09)	105	(2.10)
30	1,344	1,015	**(1.32)**	1,362	(0.99)	1,036	(1.30)	1,318	(1.02)	605	(2.22)
40	4,578	3,508	**(1.31)**	4,697	(0.97)	3,630	(1.26)	5,995	(0.76)	1,958	(2.34)
	Average		**(1.32)**		(0.98)		(1.28)		(0.96)		(2.22)

Number of Executions: Path Right - Grid 30

sld#	Std	DRA	DRE	DRA+DRE	B-Prolog	YapTab
sld1	2,702	1.00	1.00	1.00	0.18	3.00
sld2	13,851,534	1.29	1.00	**1.30**	0.30	2.21
sld3	2,702	**3.00**	1.00	1.02	**3.00**	3.00
sld4	17,400	**2.50**	1.00	1.27	**2.50**	2.50

Experimental Results

Execution Time: Warren

Depth	Std	DRA		DRE		DRA+DRE		B-Prolog		YapTab	
400	2,673	2,632	(1.02)	42	**(64.26)**	42	**(64.26)**	7,861	(0.34)	21	(126.09)
600	9,496	9,564	(0.99)	109	**(87.28)**	109	**(87.28)**	27,302	(0.35)	58	(162.61)
800	23,163	23,086	(1.00)	205	(112.88)	198	**(116.98)**	67,049	(0.35)	107	(216.88)
	Average		(1.00)		**(87.93)**		**(89.51)**		(0.35)		(168.53)

Number of Executions: Warren 600

sld#/sld#	Std	DRA	DRE	DRA+DRE	B-Prolog	YapTab
sld1/sld3	302	1.00	**100.67**	**100.67**	1.00	302.00
sld2/sld4	18,044,650	1.00	66.98	**100.42**	1.00	201.17
sld5/sld7	302	**302.00**	100.67	**302.00**	**302.00**	302.00
sld6/sld8	90,600	**302.00**	100.67	**302.00**	**302.00**	302.00

Conclusions

- We have presented a **new framework** that integrates all possible combinations of the already existent linear tabling strategies **DRA** and **DRE** using batched scheduling.
- The **combination of DRA with DRE** showed the potential of our framework to **effectively reduce the execution time** of the standard linear tabled evaluation.

Conclusions

- We have presented a **new framework** that integrates all possible combinations of the already existent linear tabling strategies **DRA** and **DRE** using batched scheduling.
- The **combination of DRA with DRE** showed the potential of our framework to **effectively reduce the execution time** of the standard linear tabled evaluation.
- Compared the framework with B-Prolog and YapTab's suspension based system.
  - The **comparison with B-Prolog**, showed that our framework is **always faster**.
  - The **comparison with YapTab** showed that it was possible to **greatly reduce** the difference in the execution time, between both systems on worst case scenarios (Warren set).
- Further work will include the extension to support **Multi-Threading**, add new strategies/optimizations to our framework and seek for new real-world problems that allow us to improve and consolidate our framework.

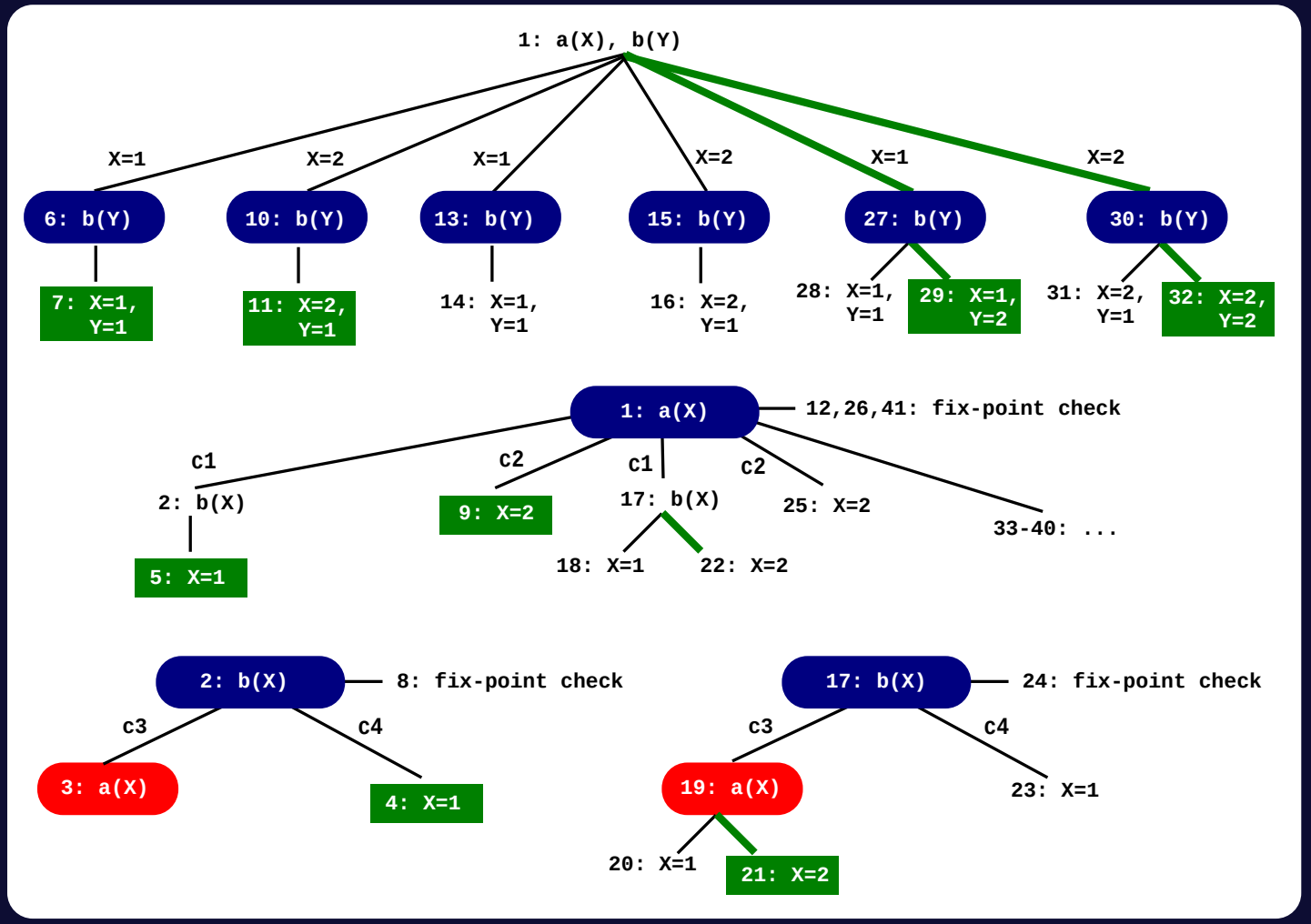
Chronology of Work Done on Linear Tabling

- An Efficient Implementation of Linear Tabling Based on Dynamic Reordering of Alternatives [**PADL-2010**,Madrid-Spain]
- Mixed-Strategies for Linear Tabling in Prolog [**CoRTA-2010**, Braga-Portugal]
- On Combining Linear-Based Strategies for Tabled Evaluation of Logic Programs [**ICLP-2011**, Kentucky-USA]
- On Extending a Linear Tabling Framework to Support Batched Scheduling [**SLATE-2012**, Braga-Portugal]

Propagation of Solutions

```
        :- table a/1, b/1.

c1)     a(X) :- b(X).
c2)     a(2).
c3)     b(X) :- a(X).
c4)     b(1).
```

Call	Solutions
1: a(X)	5: X=1 9: X=2 41: complete
2: b(X)	4: X=1 21: X=2 41: complete



Support for DRA + DRE