# Mixed-Strategies for Linear Tabling in Prolog

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

INForum-CoRTA 2010, Braga, Portugal, September 2010

# 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

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

edge(1,2).
edge(2,1).
```

```
path(1,Z)
```

# SLD Resolution: Infinite Cycles

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

edge(1,2).
edge(2,1).
```

```
path(1,Z)
    |
path(1,Y), edge(Y,Z)
    |
Infinite Cycle
```

# SLD Resolution: Infinite Cycles

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

edge(1,2).
edge(2,1).
```

```
path(1,Z)
```

# SLD Resolution: Infinite Cycles

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

edge(1,2).
edge(2,1).
```

```
path(1,Z)
    |
edge(1,Y), path(Y,Z)
    |
path(2,Z)
    |
edge(2,Y), path(Y,Z)
    |
path(1,Z)
    |
Infinite Cycle
```

# 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 and ALS Prolog).

# Linear Tabling

- Arguably, the two most well-known linear tabling strategies are:
  - **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).
  - **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 (ALS Prolog).

# Linear Tabling

- Arguably, the two most well-known linear tabling strategies are:
  - **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).
  - **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 (ALS Prolog).

- In this work, we propose a new linear tabling strategy:
  - **DRS (Dynamic Reordering of Solutions)**: it can be seen as a variant of the DRA strategy, but applied to the consumption of solutions. The key idea is to memorize the solutions leading to consumer calls, the **looping solutions**, and use them as the DRA strategy uses the looping alternatives (Yap Prolog).

# Our Goal

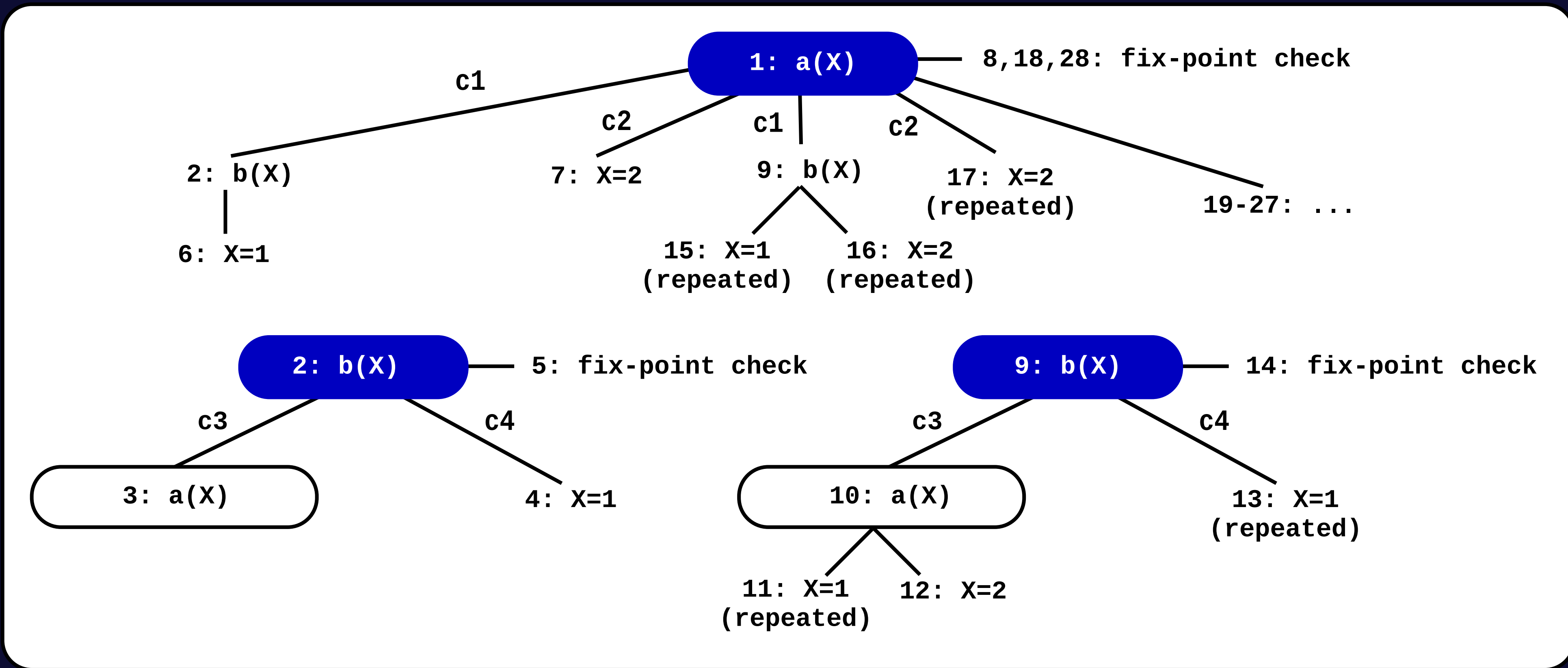
- Implement a framework on top of the **Yap Prolog system**, that supports the **combination of the three strategies**.
- Analyze the **advantages and weaknesses** of each strategy, when used **solely or combined** with the others.

# Standard Evaluation Example

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

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

| Call | Solutions |
|---|---|
| 1: a(X) | 6: X=1<br>7: X=2 |
| 2: b(X) | 4: X=1<br>12: X=2 |

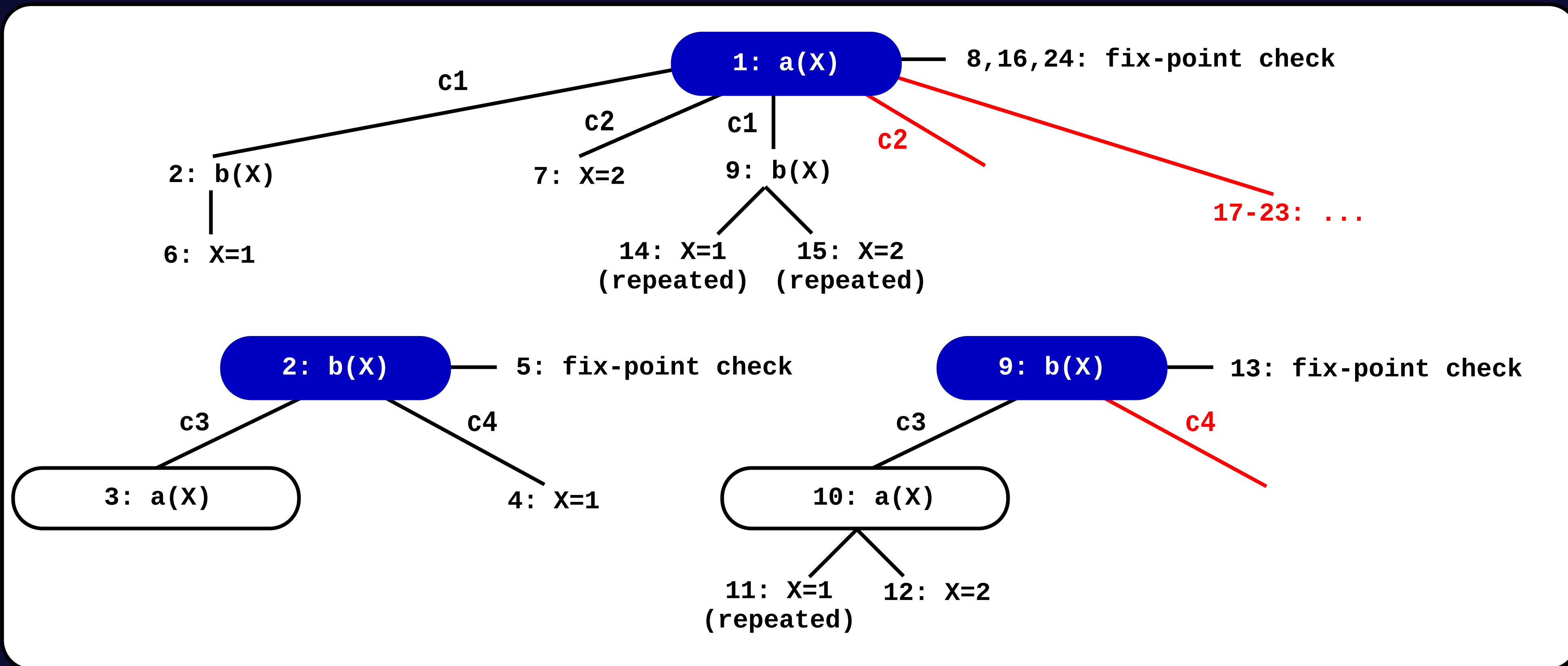


# DRA Evaluation Example

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

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

| Call | Solutions | Looping Alternatives |
|---|---|---|
| 1: a(X) | 6: X=1<br>7: X=2 | 3: c1 |
| 2: b(X) | 4: X=1<br>12: X=2 | 3: c3 |

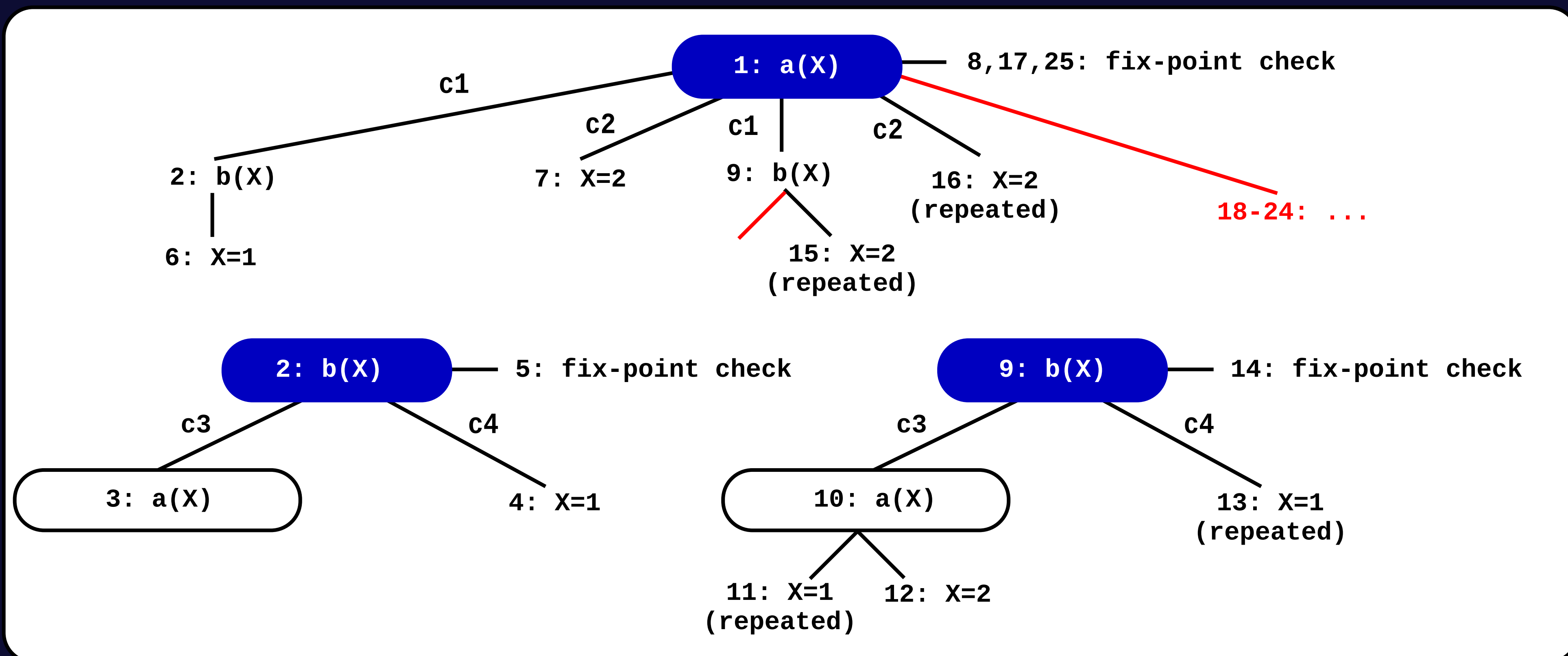


# DRS Evaluation Example

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

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

| Call | Solutions | Looping Solutions |
|---|---|---|
| 1: a(X) | 6: X=1<br>7: X=2 | |
| 2: b(X) | 4: X=1<br>12: X=2 | |

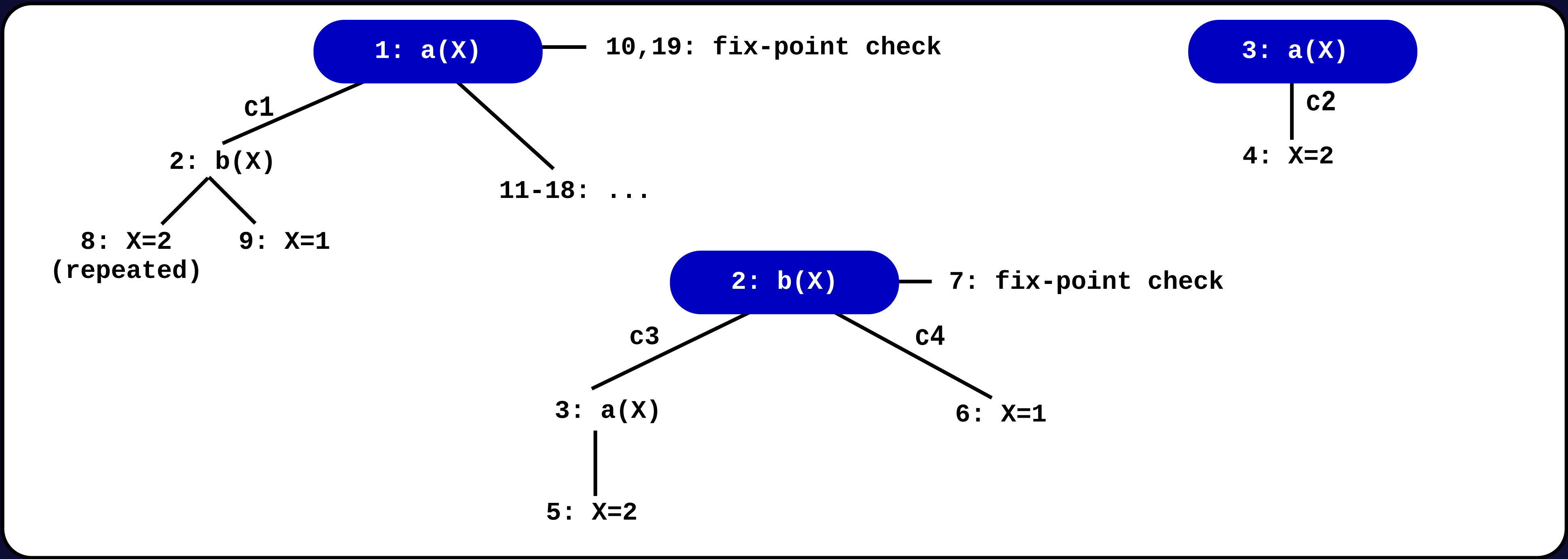


# DRE Evaluation Example

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

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

| Call | Solutions |
|---|---|
| 1: a(X) | 4: X=2<br>9: X=1 |
| 2: b(X) | 5: X=2<br>6: X=1 |



# Experimental Results

| Strategy | Pyramid | | | Cycle | | | Grid | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1000 | 2000 | 3000 | 1000 | 2000 | 3000 | 20 | 30 | 40 |
| **Path Recursive Clause First** | | | | | | | | | |
| DRE | **1.06** | 0.99 | 0.99 | 0.99 | 0.97 | **1.01** | **1.26** | **1.02** | **1.17** |
| DRA | **1.61** | **1.60** | **1.61** | **1.28** | **1.22** | **1.25** | **1.55** | **1.15** | **1.20** |
| DRS | 0.99 | 0.99 | **1.02** | **1.26** | **1.17** | **1.27** | **1.67** | **1.27** | **1.38** |
| DRE+DRA | **1.66** | **1.62** | 1.59 | **1.29** | **1.28** | **1.27** | 1.25 | 1.15 | **1.25** |
| DRE+DRS | 1.03 | 1.00 | 1.00 | **1.30** | 1.17 | **1.28** | 1.44 | **1.28** | 1.37 |
| DRA+DRS | **1.63** | 1.59 | 1.55 | **1.72** | **1.64** | **1.64** | **1.94** | **1.45** | **1.51** |
| DRE+DRA+DRS | 1.65 | 1.57 | 1.55 | 1.70 | **1.65** | 1.62 | 1.61 | 1.26 | 1.44 |
| **Path Recursive Clause Last** | | | | | | | | | |
| DRE | 0.98 | 1.00 | 0.88 | 0.94 | 0.95 | **1.04** | 0.83 | 0.99 | 0.99 |
| DRA | **1.60** | **1.59** | **1.58** | **1.18** | **1.20** | **1.22** | **1.08** | **1.09** | **1.07** |
| DRS | 0.99 | 0.98 | 0.99 | **1.14** | **1.18** | **1.25** | **1.20** | **1.20** | **1.21** |
| DRE+DRA | 1.58 | **1.66** | **1.63** | **1.22** | **1.24** | 1.22 | **1.12** | **1.10** | 1.07 |
| DRE+DRS | 1.00 | **1.01** | **1.01** | **1.22** | **1.23** | 1.23 | 0.95 | 1.14 | 1.14 |
| DRA+DRS | **1.63** | **1.64** | **1.62** | **1.56** | **1.59** | **1.69** | **1.40** | **1.32** | **1.32** |
| DRE+DRA+DRS | 1.59 | 1.57 | 1.56 | **1.61** | 1.55 | 1.60 | 1.36 | **1.33** | 1.30 |

# Conclusions and Further Work

- We have presented a new framework that integrates all possible combinations of the already existent linear tabling strategies **DRA** and **DRE** and the new strategy **DRS**.
- Our experiments for **DRS** strategy showed that the strategy of avoiding the consumption of non-looping solutions in re-evaluation rounds can be quite effective for programs that can benefit from it, with insignificant costs for the other programs.
- Further work will include exploring the impact of applying our strategies to more complex problems, seeking real-world experimental results allowing us to improve and consolidate our current implementation.