Do Repeat Yourself: Understanding Sufficient Conditions for Restricted Chase Non-Termination

Lukas Gerlach[1] David Carral[2]

[1]Knowledge-Based Systems, TU Dresden
[2]LIRMM, Inria, University of Montpellier, CNRS

05.09.2023

International Center
for Computational Logic

~~Running~~ Riding Example

~~Running~~ Riding Example

$x : \text{  } \rightarrow \Big(\exists y.\ y : \text{  } \wedge x \in y \Big) \vee x : \text{  }$

~~Running~~ Riding Example

$$x : \text{  } \rightarrow \left(\exists y.\ y : \text{  } \land x \in y \right) \lor x : \text{  }$$

$$x : \text{  } \rightarrow \exists z.\ z : \text{  } \land x \ni z$$

~~Running~~ Riding Example

$$x : \text{  } \rightarrow \Big(\exists y.\ y : \text{  } \land x \in y \Big) \lor x : \text{  }$$

$$x : \text{  } \rightarrow \exists z.\ z : \text{  } \land x \ni z$$

$$x \in y \rightarrow y \ni x \qquad x \ni y \rightarrow y \in x$$

~~Running~~ Riding Example

The **restricted chase** exhaustively applies *triggers* that are both *loaded* and not *obsolete*.

$$x : \text{  } \rightarrow \Big(\exists y.\ y : \text{  } \land x \in y\Big) \lor x : \text{  }$$

$$x : \text{  } \rightarrow \exists z.\ z : \text{  } \land x \ni z$$

$$x \in y \rightarrow y \ni x \qquad x \ni y \rightarrow y \in x$$

~~Running~~ Riding Example

The **restricted chase** exhaustively applies *triggers* that are both *loaded* and not *obsolete*.

$c : $ 

$$x : \text{  } \rightarrow \left(\exists y.\ y : \text{  } \wedge x \in y\right) \vee x : \text{  }$$

$$x : \text{  } \rightarrow \exists z.\ z : \text{  } \wedge x \ni z$$

$$x \in y \rightarrow y \ni x \qquad x \ni y \rightarrow y \in x$$

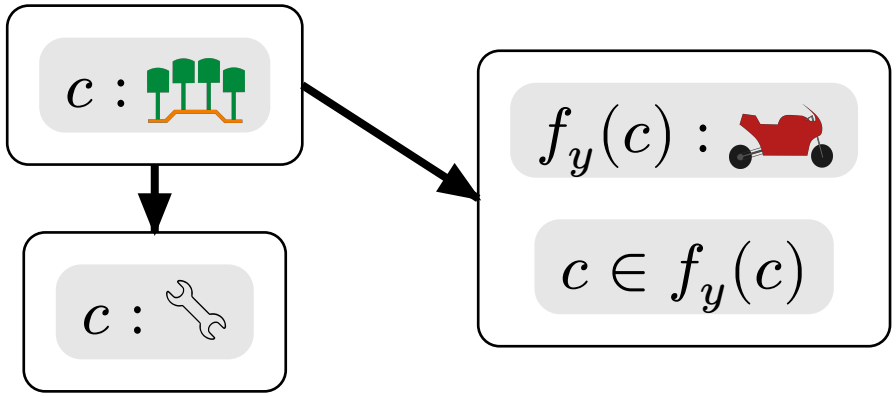
~~Running~~ Riding Example

The **restricted chase** exhaustively applies *triggers* that are both *loaded* and not *obsolete*.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench}$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z$$

$$x \in y \rightarrow y \ni x \qquad x \ni y \rightarrow y \in x$$



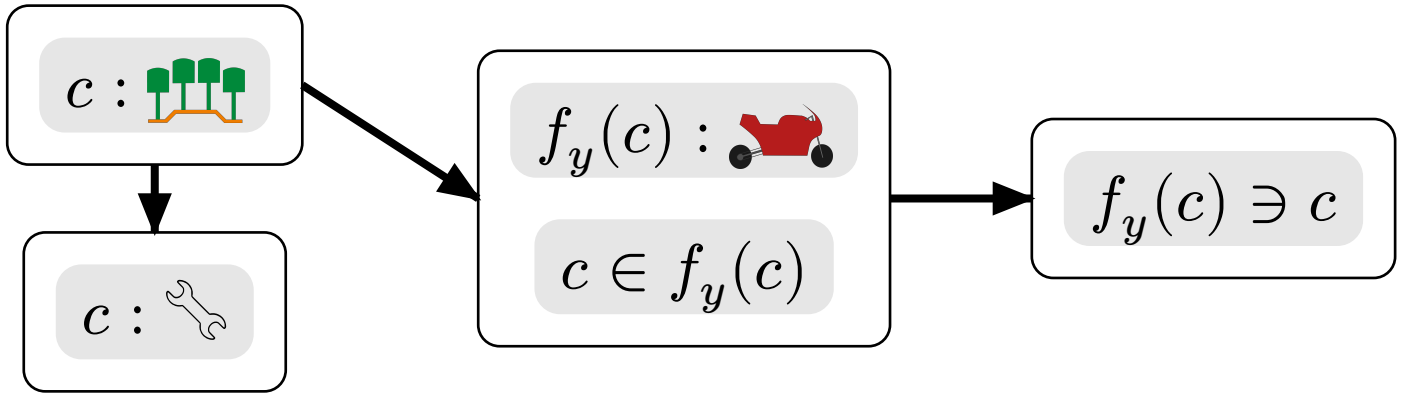
~~Running~~ Riding Example

The **restricted chase** exhaustively applies *triggers* that are both *loaded* and not *obsolete*.

$$x : \text{🌲} \rightarrow \left(\exists y.\ y : \text{🏍} \land x \in y \right) \lor x : \text{🔧}$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \land x \ni z$$

$$x \in y \rightarrow y \ni x \qquad x \ni y \rightarrow y \in x$$



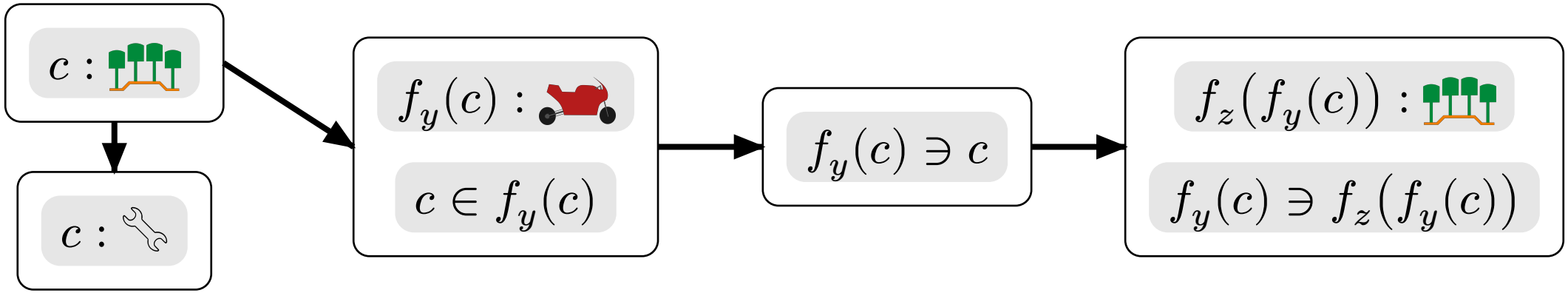
~~Running~~ Riding Example

The **restricted chase** exhaustively applies *triggers* that are both *loaded* and not *obsolete*.

$$x : \text{🌲} \rightarrow \left(\exists y.\ y : \text{🏍} \wedge x \in y\right) \vee x : \text{🔧}$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \wedge x \ni z$$

$$x \in y \rightarrow y \ni x \qquad x \ni y \rightarrow y \in x$$



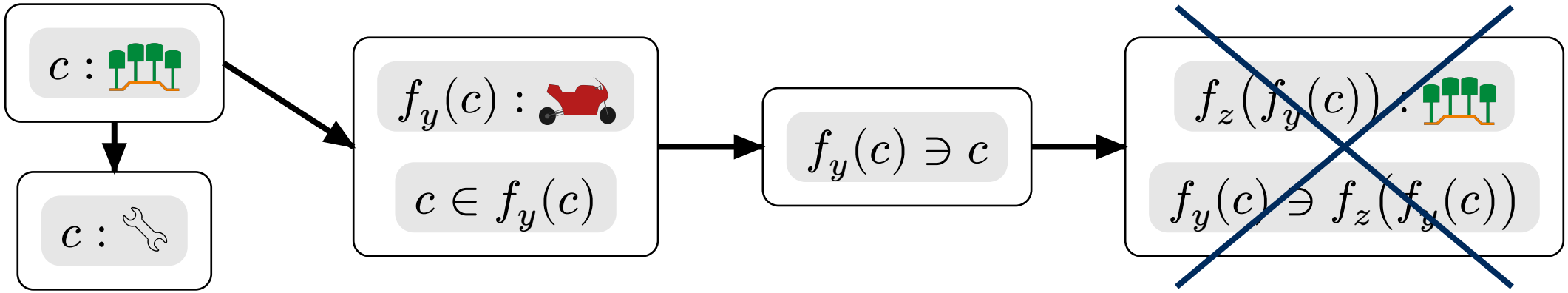
~~Running~~ Riding Example

The **restricted chase** exhaustively applies *triggers* that are both *loaded* and not *obsolete*.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench}$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z$$

$$x \in y \rightarrow y \ni x \qquad x \ni y \rightarrow y \in x$$



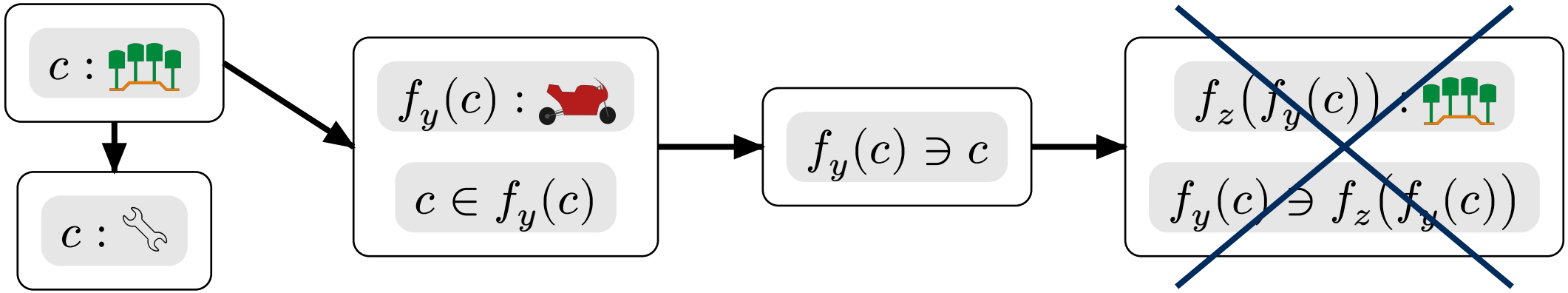
~~Running~~ Riding Example

The **restricted chase** exhaustively applies *triggers* that are both *loaded* and not *obsolete*.

$$x : \blacksquare \rightarrow \left(\exists y.\ y : \blacksquare \wedge x \in y\right) \vee x : \blacksquare$$

$$x : \blacksquare \rightarrow \exists z.\ z : \blacksquare \wedge x \ni z$$

~~$x \in y \rightarrow y \ni x$~~ ~~$x \ni y \rightarrow y \in x$~~



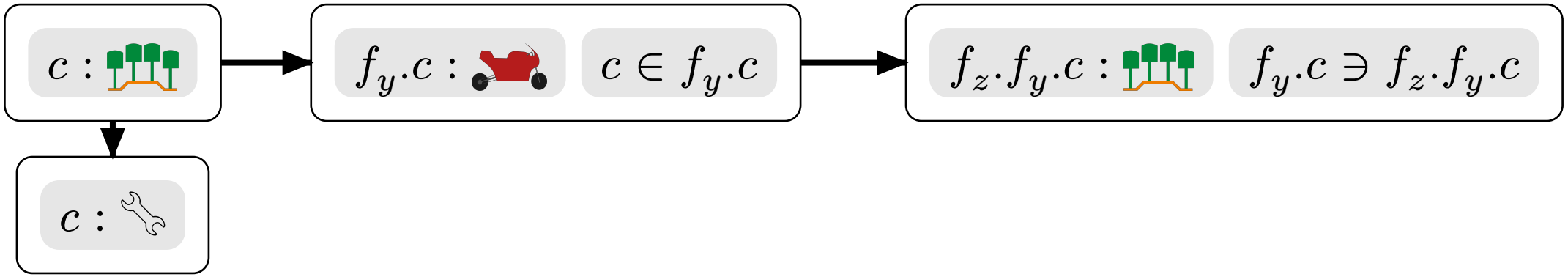
~~Running~~ Riding Example

The **restricted chase** exhaustively applies *triggers* that are both *loaded* and not *obsolete*.

$$x : \blacktree \rightarrow \left(\exists y.\ y : \bikeicon \wedge x \in y\right) \vee x : \wrenchicon$$

$$x : \bikeicon \rightarrow \exists z.\ z : \blacktree \wedge x \ni z$$

$$\cancel{x \in y \rightarrow y \ni x} \qquad \cancel{x \ni y \rightarrow y \in x}$$



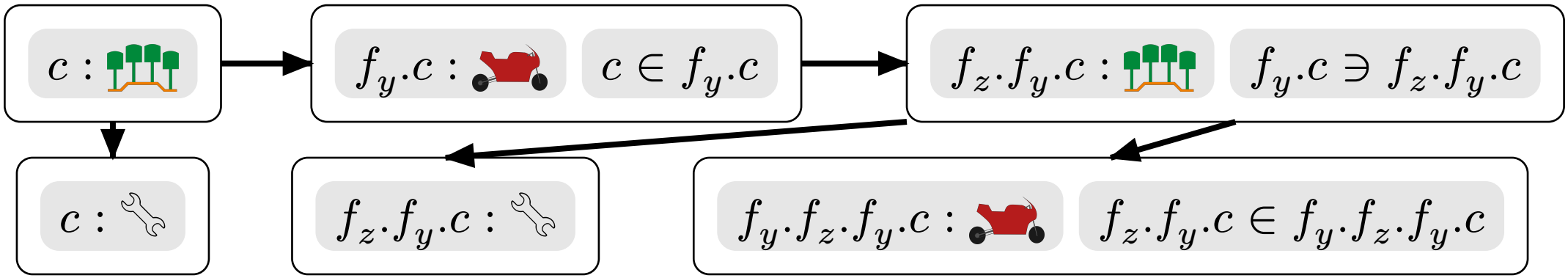
~~Running~~ Riding Example

The **restricted chase** exhaustively applies *triggers* that are both *loaded* and not *obsolete*.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench}$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z$$

~~$x \in y \rightarrow y \ni x$~~ ~~$x \ni y \rightarrow y \in x$~~



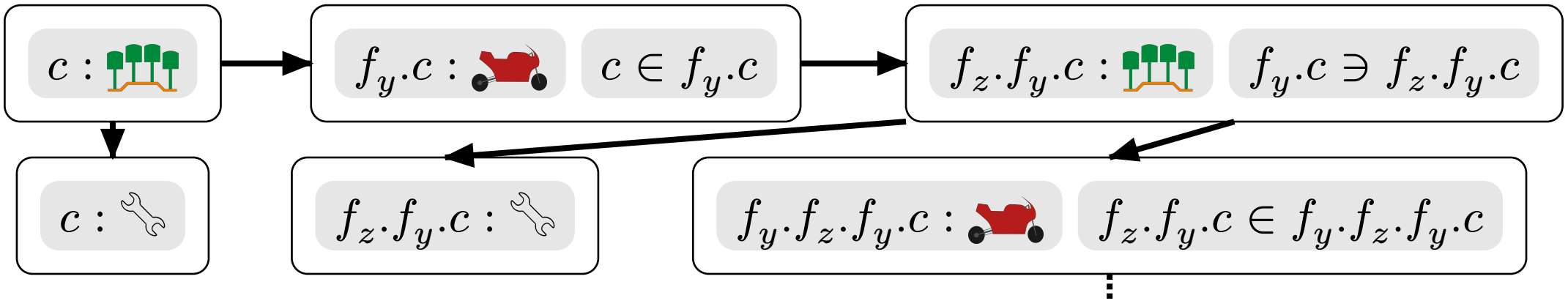
~~Running~~ Riding Example

The **restricted chase** exhaustively applies *triggers* that are both *loaded* and not *obsolete*.

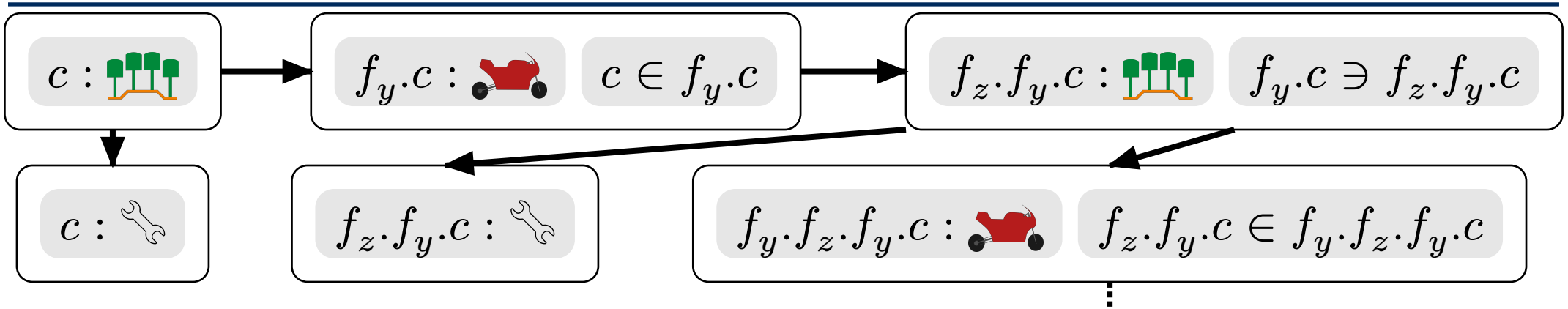
$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench}$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z$$

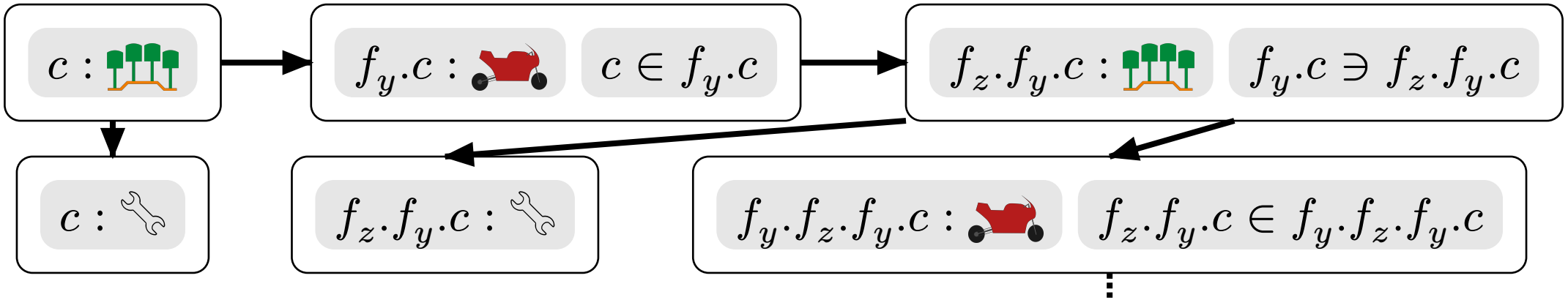
~~$x \in y \rightarrow y \ni x$~~ ~~$x \ni y \rightarrow y \in x$~~



About that Chase...

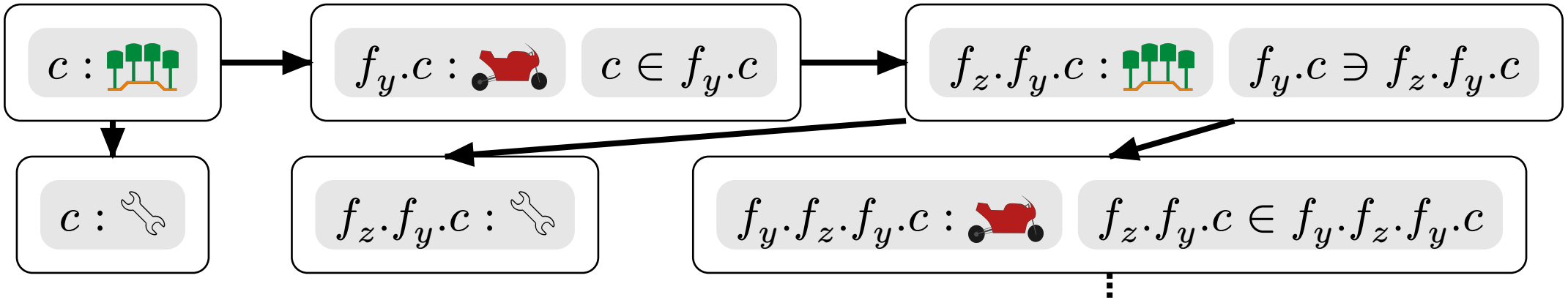


About that Chase...



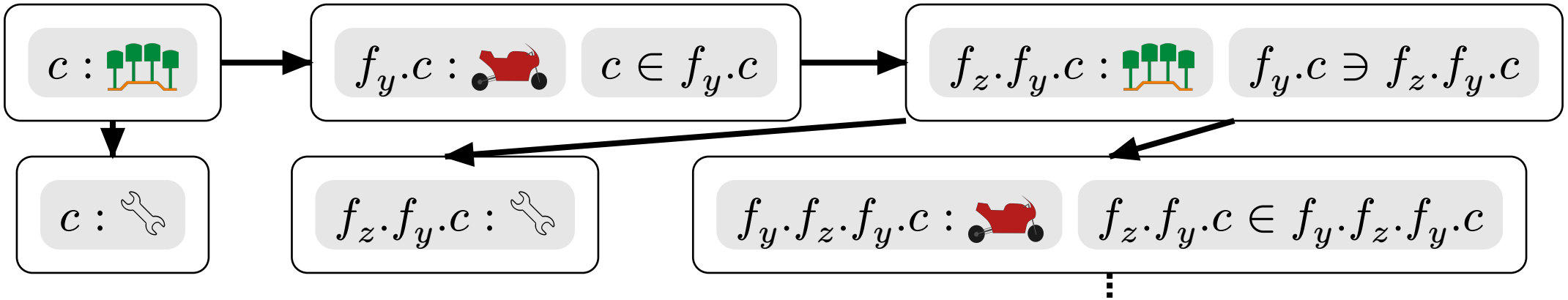
☺ Leaves of tree form *universal model set* (can answer queries) [Bou+16]

About that Chase...



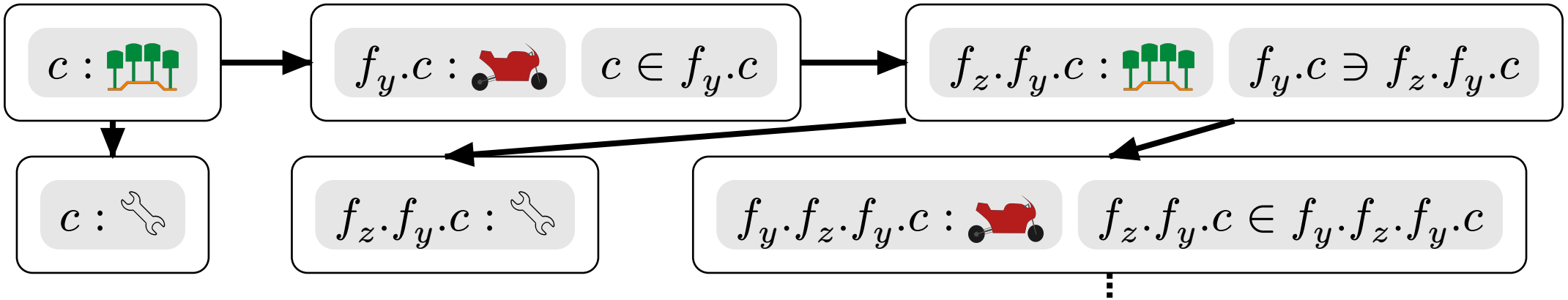
- Leaves of tree form *universal model set* (can answer queries) [Bou+16]
- But query answering/entailment is undecidable [BV81]

About that Chase...



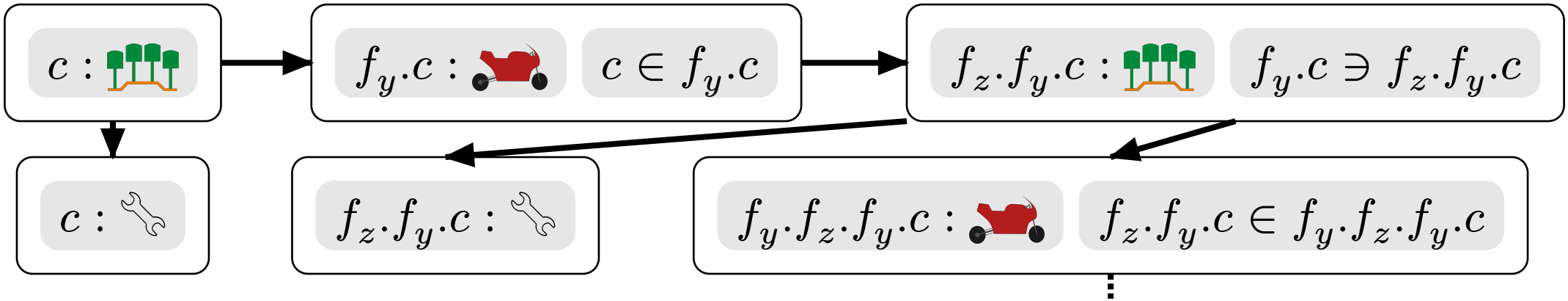
- Leaves of tree form *universal model set* (can answer queries) [Bou+16]
- But query answering/entailment is undecidable [BV81]
- Chase termination is also undecidable [GM14, GO18]

About that Chase...



- Leaves of tree form *universal model set* (can answer queries) [Bou+16]
- But query answering/entailment is undecidable [BV81]
- Chase termination is also undecidable [GM14, GO18]
- RMFA/RMFC can sometimes detect (non-)termination [CDK17]

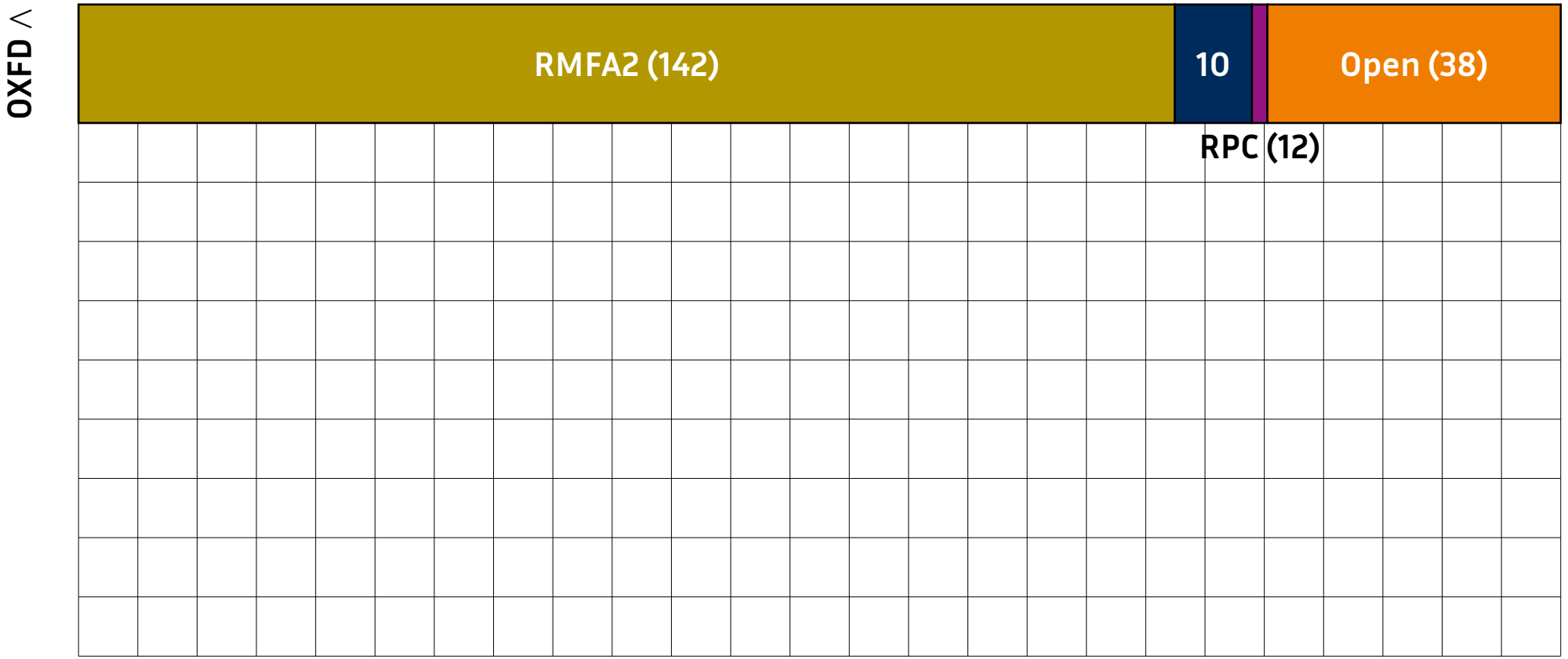
About that Chase...



- Leaves of tree form *universal model set* (can answer queries) [Bou+16]
- But query answering/entailment is undecidable [BV81]
- Chase termination is also undecidable [GM14, GO18]
- RMFA/RMFC can sometimes detect (non-)termination [CDK17]
- We reconsider RMFC with ideas from our previous work [GC23] introducing (D)RPC with a working proof and additional improvements.

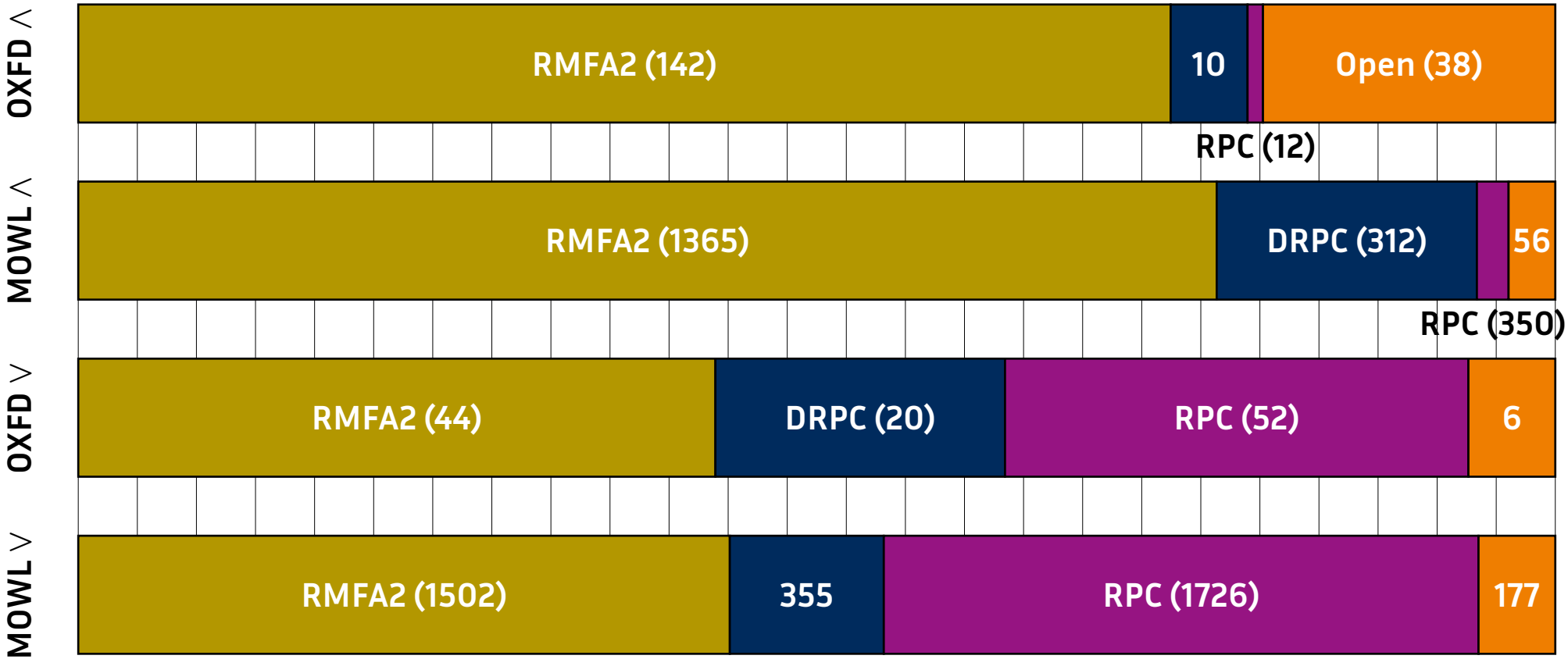
Is it any good?

```
cat my-ruleset | docker run --rm -i registry.gitlab.com/m0nstr/dmfa-checker -t [non_]termination -cv restricted -disj [-depth 2] ...
```



Is it any good?

```
cat my-ruleset | docker run --rm -i registry.gitlab.com/m0nstr/dmfa-checker -t [non_]termination -cv restricted -disj [-depth 2] ...
```



Ok, so what do we want to do exactly?

Given a rule set R, we ask if there is a *knowledge base* $\langle R, D \rangle$ that only admits infinite chase trees.

Ok, so what do we want to do exactly?

Given a rule set R, we ask if there is a *knowledge base* $\langle R, D \rangle$ that only admits infinite chase trees. Then, we say R is *never-terminating*.

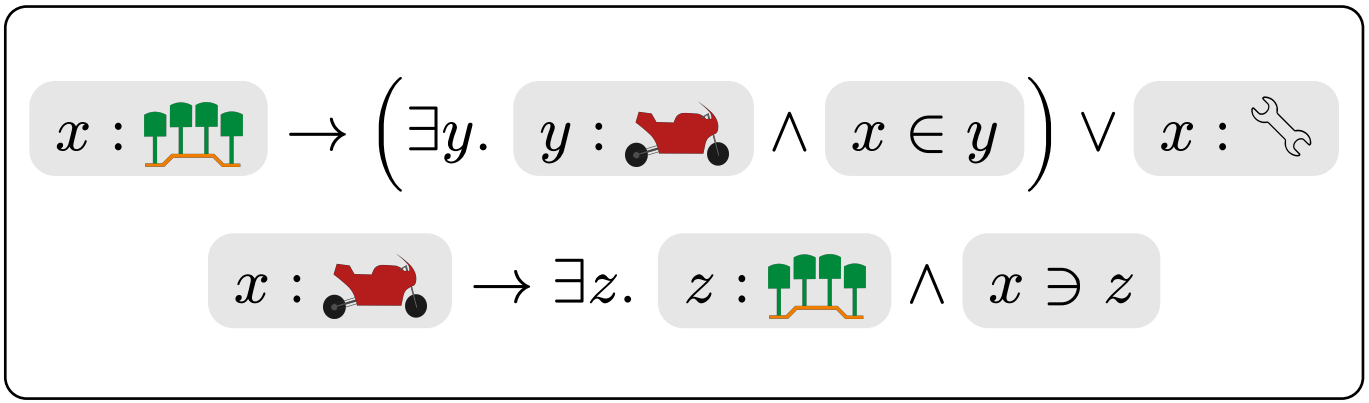
Ok, so what do we want to do exactly?

Given a rule set R, we ask if there is a *knowledge base* $\langle R, D \rangle$ that only admits infinite chase trees. Then, we say R is *never-terminating*. We want to find *sufficient conditions* that guarantee never-termination.

Ok, so what do we want to do exactly?

Given a rule set R, we ask if there is a *knowledge base* $\langle R, D \rangle$ that only admits infinite chase trees. Then, we say R is *never-terminating*. We want to find *sufficient conditions* that guarantee never-termination.

Plan of attack:



Ok, so what do we want to do exactly?

Given a rule set R, we ask if there is a *knowledge base* $\langle R, D \rangle$ that only admits infinite chase trees. Then, we say R is *never-terminating*. We want to find *sufficient conditions* that guarantee never-termination.

Plan of attack:

1. Ease Disjunctions

$$x : \text{🌲} \rightarrow \Big(\exists y.\ y : \text{🏍} \wedge x \in y \Big) \vee x : \text{🔧}$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \wedge x \ni z$$

Ok, so what do we want to do exactly?

Given a rule set R, we ask if there is a *knowledge base* $\langle R, D \rangle$ that only admits infinite chase trees. Then, we say R is *never-terminating*. We want to find *sufficient conditions* that guarantee never-termination.

Plan of attack:

1. Ease Disjunctions
2. Cyclicity Sequences

$$x : \text{🌲} \rightarrow \Big(\exists y.\ y : \text{🏍} \land x \in y \Big) \lor x : \text{🔧}$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \land x \ni z$$

Ok, so what do we want to do exactly?

Given a rule set R, we ask if there is a *knowledge base* $\langle R, D \rangle$ that only admits infinite chase trees. Then, we say R is *never-terminating*. We want to find *sufficient conditions* that guarantee never-termination.

Plan of attack:

1. Ease Disjunctions
2. Cyclicity Sequences
3. Unblockability

$x : \text{  } \rightarrow \left(\exists y.\ y : \text{  } \land x \in y \right) \lor x : \text{  }$

$x : \text{  } \rightarrow \exists z.\ z : \text{  } \land x \ni z$

Ok, so what do we want to do exactly?

Given a rule set R, we ask if there is a *knowledge base* $\langle R, D \rangle$ that only admits infinite chase trees. Then, we say R is *never-terminating*. We want to find *sufficient conditions* that guarantee never-termination.

Plan of attack:

1. Ease Disjunctions
2. Cyclicity Sequences
3. Unblockability
4. Cyclicity Prefixes

$x : \text{  } \rightarrow \Big(\exists y.\ y : \text{  } \wedge x \in y \Big) \vee x : \text{  }$

$x : \text{  } \rightarrow \exists z.\ z : \text{  } \wedge x \ni z$

Ok, so what do we want to do exactly?

Given a rule set R, we ask if there is a *knowledge base* $\langle R, D \rangle$ that only admits infinite chase trees. Then, we say R is *never-terminating*. We want to find *sufficient conditions* that guarantee never-termination.

Plan of attack:

1. Ease Disjunctions
2. Cyclicity Sequences
3. Unblockability
4. Cyclicity Prefixes
5. (D)RPC

$$x : \text{trees} \rightarrow \Big(\exists y.\ y : \text{bike} \wedge x \in y \Big) \vee x : \text{wrench}$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{trees} \wedge x \ni z$$

Ok, so what do we want to do exactly?

Given a rule set R, we ask if there is a *knowledge base* $\langle R, D \rangle$ that only admits infinite chase trees. Then, we say R is *never-terminating*. We want to find *sufficient conditions* that guarantee never-termination.

Plan of attack:

1. *Ease Disjunctions*
2. Cyclicity Sequences
3. Unblockability
4. Cyclicity Prefixes
5. (D)RPC

Head Choice [GC23] fixes disjunct for each rule.

$$x : \text{tree} \rightarrow \Big(\exists y.\ y : \text{bike} \wedge x \in y \Big) \vee x : \text{wrench}$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z$$

2. Cyclicity Sequences

R is never-terminating if for some KB and HC there exists a *cyclicity sequence*, i.e. a *loaded*, *growing*, and *g-unblk.* trigger sequence.

$$x : \text{🌲} \rightarrow \left(\exists y.\ y : \text{🏍} \ \wedge\ x \in y \right) \vee x : \text{🔧} \qquad (1)$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \ \wedge\ x \ni z \qquad (2)$$

2. Cyclicity Sequences

R is never-terminating if for some KB and HC there exists a *cyclicity sequence*, i.e. a *loaded*, *growing*, and *g-unblk.* trigger sequence.

$$x : \text{trees} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y \right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{trees} \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.

2. Cyclicity Sequences

R is never-terminating if for some KB and HC there exists a *cyclicity sequence*, i.e. a *loaded*, *growing*, and *g-unblk.* trigger sequence.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y \right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.

For the DB $c : \text{tree}$, we find the following cyclicity sequence:

2. Cyclicity Sequences

R is never-terminating if for some KB and HC there exists a *cyclicity sequence*, i.e. a *loaded*, *growing*, and *g-unblk.* trigger sequence.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.

For the DB $c : \text{tree}$, we find the following cyclicity sequence:

$\langle (1), [x/c] \rangle$

2. Cyclicity Sequences

R is never-terminating if for some KB and HC there exists a *cyclicity sequence*, i.e. a *loaded*, *growing*, and *g-unblk.* trigger sequence.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.

For the DB $c : \text{tree}$, we find the following cyclicity sequence:

$\langle (1), [x/c] \rangle$ $\langle (2), [x/f_y.c] \rangle$

2. Cyclicity Sequences

R is never-terminating if for some KB and HC there exists a *cyclicity sequence*, i.e. a *loaded*, *growing*, and *g-unblk.* trigger sequence.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.

For the DB $c : \text{tree}$, we find the following cyclicity sequence:

$\langle (1), [x/c] \rangle$ $\langle (2), [x/f_y.c] \rangle$ $\langle (1), [x/f_z.f_y.c] \rangle$

2. Cyclicity Sequences

R is never-terminating if for some KB and HC there exists a *cyclicity sequence*, i.e. a *loaded*, *growing*, and *g-unblk.* trigger sequence.

$$x : \blacksquare \rightarrow \left(\exists y.\ y : \blacksquare \wedge x \in y\right) \vee x : \blacksquare \quad (1)$$

$$x : \blacksquare \rightarrow \exists z.\ z : \blacksquare \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.

For the DB $c : \blacksquare$, we find the following cyclicity sequence:

$\langle (1), [x/c] \rangle$ $\langle (2), [x/f_y.c] \rangle$ $\langle (1), [x/f_z.f_y.c] \rangle$ $\langle (2), [x/f_y.f_z.f_y.c] \rangle$

2. Cyclicity Sequences

R is never-terminating if for some KB and HC there exists a *cyclicity sequence*, i.e. a *loaded*, *growing*, and *g-unblk.* trigger sequence.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.

For the DB $c : \text{tree}$, we find the following cyclicity sequence:

$\langle (1), [x/c] \rangle$ $\langle (2), [x/f_y.c] \rangle$ $\langle (1), [x/f_z.f_y.c] \rangle$ $\langle (2), [x/f_y.f_z.f_y.c] \rangle$ ...

2. Cyclicity Sequences

Loaded:
The body of every trigger is satisfied by the previous outputs (and the database).

$$x : \text{  } \rightarrow \Big(\exists y.\ y : \text{  } \wedge x \in y\Big) \vee x : \text{  } \quad (1)$$

$$x : \text{  } \rightarrow \exists z.\ z : \text{  } \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.

For the DB $c : \text{  }$, we find the following cyclicity sequence:

$\langle (1), [x/c] \rangle$ $\langle (2), [x/f_y.c] \rangle$ $\langle (1), [x/f_z.f_y.c] \rangle$ $\langle (2), [x/f_y.f_z.f_y.c] \rangle$ ...

2. Cyclicity Sequences

Growing:
After finitely many steps a new term must be introduced.
This implies infinity.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.

For the DB $c : \text{tree}$, we find the following cyclicity sequence:

$\langle (1), [x/c] \rangle$ $\langle (2), [x/f_y.c] \rangle$ $\langle (1), [x/f_z.f_y.c] \rangle$ $\langle (2), [x/f_y.f_z.f_y.c] \rangle$ ...

2. Cyclicity Sequences

G-Unblockable:
If a trigger is loaded in the HC-branch of any CT, then its HC-output occurs in this branch.

$$x : \blacktriangle \rightarrow \Big(\exists y.\ y : \blacksquare \wedge x \in y\Big) \vee x : \blacktriangledown \quad (1)$$

$$x : \blacksquare \rightarrow \exists z.\ z : \blacktriangle \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.

For the DB $c : \blacktriangle$, we find the following cyclicity sequence:

$\langle (1), [x/c] \rangle$ $\langle (2), [x/f_y.c] \rangle$ $\langle (1), [x/f_z.f_y.c] \rangle$ $\langle (2), [x/f_y.f_z.f_y.c] \rangle$ ...

2. Cyclicity Sequences

R is never-terminating if for some KB and HC there exists a *cyclicity sequence*, i.e. a *loaded*, *growing*, and *g-unblk.* trigger sequence.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.

For the DB $c : \text{tree}$, we find the following cyclicity sequence:

$\langle (1), [x/c] \rangle$ $\langle (2), [x/f_y.c] \rangle$ $\langle (1), [x/f_z.f_y.c] \rangle$ $\langle (2), [x/f_y.f_z.f_y.c] \rangle$ ...

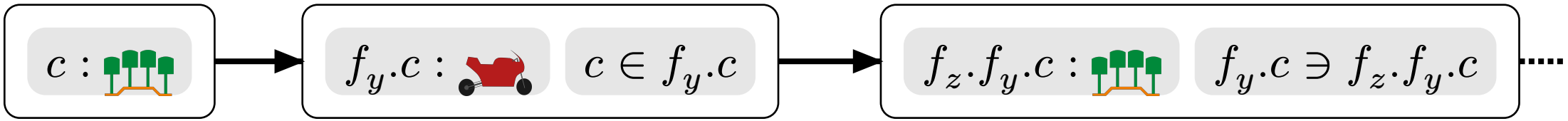
2. Cyclicity Sequences

Loadedness and growth are easily verified for a trigger.

$$x : \text{🌲} \rightarrow \left(\exists y.\ y : \text{🏍} \wedge x \in y \right) \vee x : \text{🔧} \quad (1)$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.



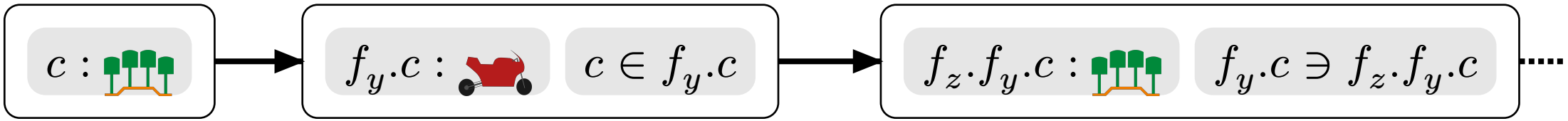
2. Cyclicity Sequences

Loadedness and growth are easily verified for a trigger. **How do we verify g-unblockability?**

$$x : \blacksquare \rightarrow \left(\exists y.\ y : \blacksquare \wedge x \in y\right) \vee x : \blacksquare \quad (1)$$

$$x : \blacksquare \rightarrow \exists z.\ z : \blacksquare \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.



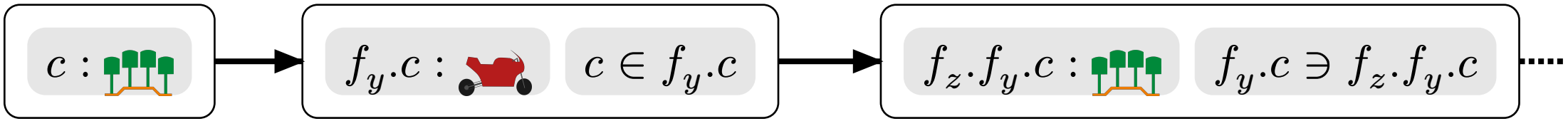
2. Cyclicity Sequences

Theorem.
Checking if a trigger is g-unblockable is undecidable!

$$x : \blacksquare \rightarrow \left(\exists y.\ y : \blacksquare \wedge x \in y\right) \vee x : \blacksquare \quad (1)$$

$$x : \blacksquare \rightarrow \exists z.\ z : \blacksquare \wedge x \ni z \quad (2)$$

Such a sequence is embeddable into every CT.



3. Unique Constant (UC) Unblockability

UC-Overapproximation
before a trigger λ

$$x : \text{🌲} \rightarrow \Big(\exists y.\ y : \text{🏍} \wedge x \in y\Big) \vee x : \text{🔧} \quad (1)$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \wedge x \ni z \quad (2)$$

We show UC-Overapp. before $\langle (2), [x/f_y.c] \rangle$ *:*

3. Unique Constant (UC) Unblockability

UC-Overapproximation
before a trigger λ
1. Birth Facts of λ

$$x : \text{🌲} \rightarrow \Big(\exists y.\ y : \text{🏍} \wedge x \in y\Big) \vee x : \text{🔧} \quad (1)$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \wedge x \ni z \quad (2)$$

We show UC-Overapp. before $\langle (2), [x/f_y.c] \rangle$ *:*

$f_y.c : \text{🏍}$ $\quad c \in f_y.c$

3. Unique Constant (UC) Unblockability

UC-Overapproximation
before a trigger λ
1. Birth Facts of λ
2. Critical Facts of λ

$$x : \text{🌲} \rightarrow \Big(\exists y.\ y : \text{🏍} \wedge x \in y\Big) \vee x : \text{🔧} \quad (1)$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \wedge x \ni z \quad (2)$$

We show UC-Overapp. before $\langle (2), [x/f_y.c] \rangle$ *:*

$$f_y.c : \text{🏍} \qquad c \in f_y.c \qquad \star, c : \text{🏍} \qquad \star, c : \text{🌲} \qquad \star, c : \text{🔧}$$

$$\star \in \star \qquad \star \in c \qquad c \in \star \qquad c \in c \qquad \star \ni \star \qquad \star \ni c \qquad c \ni \star \qquad c \ni c$$

3. Unique Constant (UC) Unblockability

UC-Overapproximation
before a trigger λ
1. Birth Facts of λ
2. Critical Facts of λ
3. Chase with UC-Rules

$$x : \text{🌲} \rightarrow d_y : \text{🏍} \wedge x \in d_y \vee x : \text{🔧} \quad (1)$$

$$x : \text{🏍} \rightarrow d_z : \text{🌲} \wedge x \ni d_z \quad (2)$$

We show UC-Overapp. before $\langle (2), [x/f_y.c] \rangle$ *:*

$f_y.c : \text{🏍}$ $\quad c \in f_y.c$ $\quad \star, c : \text{🏍}$ $\quad \star, c : \text{🌲}$ $\quad \star, c : \text{🔧}$

$\star \in \star$, $\star \in c$, $c \in \star$, $c \in c$ $\quad \star \ni \star$, $\star \ni c$, $c \ni \star$, $c \ni c$

$d_y : \text{🏍}$ $\quad \star, c \in d_y$ $\quad d_z : \text{🌲}$ $\quad \star, c \ni d_z$ $\quad d_z \in d_y$ $\quad d_y \ni d_z$

3. Unique Constant (UC) Unblockability

UC-Overapproximation
before a trigger λ
1. Birth Facts of λ
2. Critical Facts of λ
3. Chase with UC-Rules

$$x : \blacktriangle \rightarrow \left(\exists y.\ y : \blacksquare \wedge x \in y\right) \vee x : \blacktriangledown \quad (1)$$

$$x : \blacksquare \rightarrow \exists z.\ z : \blacktriangle \wedge x \ni z \quad (2)$$

$\langle (2), [x/f_y.c] \rangle$ is applicable to its overapp.

$f_y.c : \blacksquare$ $c \in f_y.c$ $\star, c : \blacksquare$ $\star, c : \blacktriangle$ $\star, c : \blacktriangledown$

$\star \in \star$, $\star \in c$, $c \in \star$, $c \in c$, $\star \ni \star$, $\star \ni c$, $c \ni \star$, $c \ni c$

$d_y : \blacksquare$ $\star, c \in d_y$ $d_z : \blacktriangle$ $\star, c \ni d_z$ $d_z \in d_y$ $d_y \ni d_z$

3. Unique Constant (UC) Unblockability

UC-Overapproximation
before a trigger λ
1. Birth Facts of λ
2. Critical Facts of λ
3. Chase with UC-Rules

$$x : \blacktriangle \rightarrow \left(\exists y.\ y : \blacksquare \wedge x \in y\right) \vee x : \blacktriangledown \quad (1)$$

$$x : \blacksquare \rightarrow \exists z.\ z : \blacktriangle \wedge x \ni z \quad (2)$$

$\langle (2), [x/f_y.c] \rangle$ is uc-unblockable!

$$f_y.c : \blacksquare \quad c \in f_y.c \quad \star, c : \blacksquare \quad \star, c : \blacktriangle \quad \star, c : \blacktriangledown$$

$$\star \in \star \quad \star \in c \quad c \in \star \quad c \in c \quad \star \ni \star \quad \star \ni c \quad c \ni \star \quad c \ni c$$

$$d_y : \blacksquare \quad \star, c \in d_y \quad d_z : \blacktriangle \quad \star, c \ni d_z \quad d_z \in d_y \quad d_y \ni d_z$$

3. Unique Constant (UC) Unblockability

UC-Overapproximation
before a trigger λ
1. Birth Facts of λ
2. Critical Facts of λ
3. Chase with UC-Rules

$$x : \text{🌲} \rightarrow \left(\exists y.\ y : \text{🏍} \wedge x \in y \right) \vee x : \text{🔧} \quad (1)$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \wedge x \ni z \quad (2)$$

Would not work with $x \in y \rightarrow y \ni x$

$f_y.c : \text{🏍}$ $\quad c \in f_y.c$ $\quad \star, c : \text{🏍}$ $\quad \star, c : \text{🌲}$ $\quad \star, c : \text{🔧}$

$\star \in \star$, $\star \in c$, $c \in \star$, $c \in c$ $\quad \star \ni \star$, $\star \ni c$, $c \ni \star$, $c \ni c$

$d_y : \text{🏍}$ $\quad \star, c \in d_y$ $\quad d_z : \text{🌲}$ $\quad \star, c \ni d_z$ $\quad d_z \in d_y$ $\quad d_y \ni d_z$

3. Unique Constant (UC) Unblockability

UC-Overapproximation
before a trigger λ
1. Birth Facts of λ
2. Critical Facts of λ
3. Chase with UC-Rules

Lemma.
If a trigger is uc-unblockable, it is g-unblk.
Every possible fact that may occur in any CT where the trigger has not been applied can be embedded into the overapprox.

$f_y.c :$  $\quad c \in f_y.c \quad \star, c :$  $\quad \star, c :$  $\quad \star, c :$ 

$\star \in \star$, $\star \in c$, $c \in \star$, $c \in c$ $\quad \star \ni \star$, $\star \ni c$, $c \ni \star$, $c \ni c$

$d_y :$  $\quad \star, c \in d_y \quad d_z :$  $\quad \star, c \ni d_z \quad d_z \in d_y \quad d_y \ni d_z$

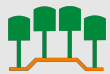
3.' Unblockability Propagates

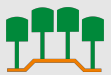
We only want to check uc-unblockability for finitely many triggers.

3.' Unblockability Propagates

We only want to check uc-unblockability for finitely many triggers.

Reversible Constant Mappings [GC23] preserve uc-unblk.

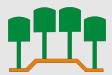
$$x : \text{  } \rightarrow \left(\exists y.\ y : \text{  } \wedge x \in y \right) \vee x : \text{  } \quad (1)$$

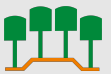
$$x : \text{  } \rightarrow \exists z.\ z : \text{  } \wedge x \ni z \quad (2)$$

3.' Unblockability Propagates

We only want to check uc-unblockability for finitely many triggers.

Reversible Constant Mappings [GC23] preserve uc-unblk.

$$x : \text{  } \rightarrow \Big(\exists y.\ y : \text{  } \wedge x \in y\Big) \vee x : \text{  } \quad (1)$$

$$x : \text{  } \rightarrow \exists z.\ z : \text{  } \wedge x \ni z \quad (2)$$

$\lambda' = \langle (2), [x/f_y.f_z.f_y.c] \rangle$ results from $\lambda = \langle (2), [x/f_y.c] \rangle$ with the reversible constant mapping $g : c \mapsto f_z.f_y.c$.

3.' Unblockability Propagates

We only want to check uc-unblockability for finitely many triggers.

Reversible Constant Mappings [GC23] preserve uc-unblk.

$$x : \text{🌲} \rightarrow \left(\exists y.\ y : \text{🏍} \wedge x \in y\right) \vee x : \text{🔧} \quad (1)$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \wedge x \ni z \quad (2)$$

$\lambda' = \langle (2), [x/f_y.f_z.f_y.c] \rangle$ results from $\lambda = \langle (2), [x/f_y.c] \rangle$ with the reversible constant mapping $g : c \mapsto f_z.f_y.c$. The UC-overapproximation of λ' can be embedded into the uc-overapproximation of λ by mapping all "superterms" of $f_z.f_y.c$ by the inverse of g and all other terms to $\star$.

3.' Unblockability Propagates

We only want to check uc-unblockability for finitely many triggers.

Reversible Constant Mappings [GC23] preserve uc-unblk.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y \right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

$$f_y.c : \text{bike} \quad c \in f_y.c \quad \star, c : \text{bike} \quad \star, c : \text{tree} \quad \star, c : \text{wrench}$$

$$\star \in \star \quad \star \in c \quad c \in \star \quad c \in c \quad \star \ni \star \quad \star \ni c \quad c \ni \star \quad c \ni c$$

$$d_y : \text{bike} \quad \star, c \in d_y \quad d_z : \text{tree} \quad \star, c \ni d_z \quad d_z \in d_y \quad d_y \ni d_z$$

3.' Unblockability Propagates

We only want to check uc-unblockability for finitely many triggers.

Reversible Constant Mappings [GC23] preserve uc-unblk.

$$x : \text{🌲} \rightarrow \left(\exists y.\ y : \text{🏍} \wedge x \in y\right) \vee x : \text{🔧} \quad (1)$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \wedge x \ni z \quad (2)$$

$\lambda' = \langle (2), [x/f_y.f_z.f_y.c] \rangle$ results from $\lambda = \langle (2), [x/f_y.c] \rangle$ with the reversible constant mapping $g : c \mapsto f_z.f_y.c$. The UC-overapproximation of λ' can be embedded into the uc-overapproximation of λ by mapping all "superterms" of $f_z.f_y.c$ by the inverse of g and all other terms to $\star$.
In fact, we can show that this is always possible!

3.' Unblockability Propagates

We only want to check uc-unblockability for finitely many triggers.

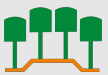
Reversible Constant Mappings [GC23] preserve uc-unblk.

Lemma.
If a trigger $\langle \rho, \sigma \rangle$ is uc-unblk., then $\langle \rho, g \circ \sigma \rangle$ is uc-unblk. for every rev. constant mapping g.

$\lambda' = \langle (2), [x/f_y.f_z.f_y.c] \rangle$ results from $\lambda = \langle (2), [x/f_y.c] \rangle$ with the reversible constant mapping $g : c \mapsto f_z.f_y.c$. The UC-overapproximation of λ' can be embedded into the uc-overapproximation of λ by mapping all "superterms" of $f_z.f_y.c$ by the inverse of g and all other terms to $\star$.
In fact, we can show that this is always possible!

4. Cyclicity Prefixes

Finite Trigger Sequence extendable into an (infinite) Cyclicity Sequence

$$x : \text{  } \rightarrow \left(\exists y.\ y : \text{  } \wedge x \in y \right) \vee x : \text{  } \quad (1)$$

$$x : \text{  } \rightarrow \exists z.\ z : \text{  } \wedge x \ni z \quad (2)$$

4. Cyclicity Prefixes

Finite Trigger Sequence extendable into an (infinite) Cyclicity Sequence

1. Start: Rule DB Trg

$$x : \text{🌲} \rightarrow \left(\exists y.\ y : \text{🏍} \wedge x \in y\right) \vee x : \text{🔧} \quad (1)$$

$$x : \text{🏍} \rightarrow \exists z.\ z : \text{🌲} \wedge x \ni z \quad (2)$$

Consider the rule DB for (1): $c_x : \text{🌲}$

Prefix: $\langle (1), [x/c_x] \rangle$

4. Cyclicity Prefixes

Finite Trigger Sequence extendable into an (infinite) Cyclicity Sequence

1. Start: Rule DB Trg
2. DB Trg is g-unblk.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Consider the rule DB for (1): $c_x : \text{tree}$

Prefix: $\langle (1), [x/c_x] \rangle$

4. Cyclicity Prefixes

Finite Trigger Sequence extendable into an (infinite) Cyclicity Sequence

1. Start: Rule DB Trg
2. DB Trg is g-unblk.
3. Ldd, UC-unblk trgs

$$x : \text{trees} \rightarrow \left(\exists y.\ y : \text{motorcycle} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{motorcycle} \rightarrow \exists z.\ z : \text{trees} \wedge x \ni z \quad (2)$$

Consider the rule DB for (1): $c_x : \text{trees}$

Prefix: $\langle (1), [x/c_x] \rangle$ $\langle (2), [x/f_y.c_x] \rangle$

4. Cyclicity Prefixes

Finite Trigger Sequence extendable into an (infinite) Cyclicity Sequence

1. Start: Rule DB Trg
2. DB Trg is g-unblk.
3. Ldd, UC-unblk trgs
4. Repeat with cycl. t.

$$x : \text{trees} \rightarrow \left(\exists y.\ y : \text{motorcycle} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{motorcycle} \rightarrow \exists z.\ z : \text{trees} \wedge x \ni z \quad (2)$$

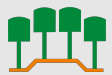
Consider the rule DB for (1): $c_x : \text{trees}$

Prefix: $\langle (1), [x/c_x] \rangle$ $\langle (2), [x/f_y.c_x] \rangle$ $\langle (1), [x/f_z.f_y.c_x] \rangle$

4. Cyclicity Prefixes

Finite Trigger Sequence extendable into an (infinite) Cyclicity Sequence

1. Start: Rule DB Trg
2. DB Trg is g-unblk.
3. Ldd, UC-unblk trgs
4. Repeat with cycl. t.
5. Induced CM is rev.

$$x : \text{  } \rightarrow \left(\exists y.\ y : \text{  } \wedge\ x \in y\right) \vee x : \text{  } \quad (1)$$

$$x : \text{  } \rightarrow \exists z.\ z : \text{  } \wedge\ x \ni z \quad (2)$$

Consider the rule DB for (1): $c_x : \text{  }$

Prefix: $\langle (1), [x/c_x] \rangle$ $\langle (2), [x/f_y.c_x] \rangle$ $\langle (1), [x/f_z.f_y.c_x] \rangle$

with constant mapping $g : c_x \mapsto f_z.f_y.c_x$ (which is reversible).

4. Cyclicity Prefixes

Finite Trigger Sequence extendable into an (infinite) Cyclicity Sequence

1. Start: Rule DB Trg
2. DB Trg is g-unblk.
3. Ldd, UC-unblk trgs
4. Repeat with cycl. t.
5. Induced CM is rev.

$$x : \text{trees} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y \right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{trees} \wedge x \ni z \quad (2)$$

Consider the rule DB for (1): $c_x : \text{trees}$

Rule Database

$c_x : \text{trees}$

4. Cyclicity Prefixes

Finite Trigger Sequence extendable into an (infinite) Cyclicity Sequence

1. Start: Rule DB Trg
2. DB Trg is g-unblk.
3. Ldd, UC-unblk trgs
4. Repeat with cycl. t.
5. Induced CM is rev.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Consider the rule DB for (1): $c_x : \text{tree}$

Rule Database

$c_x : \text{tree}$

$f_y.c : \text{bike}$

$c \in f_y.c$

4. Cyclicity Prefixes

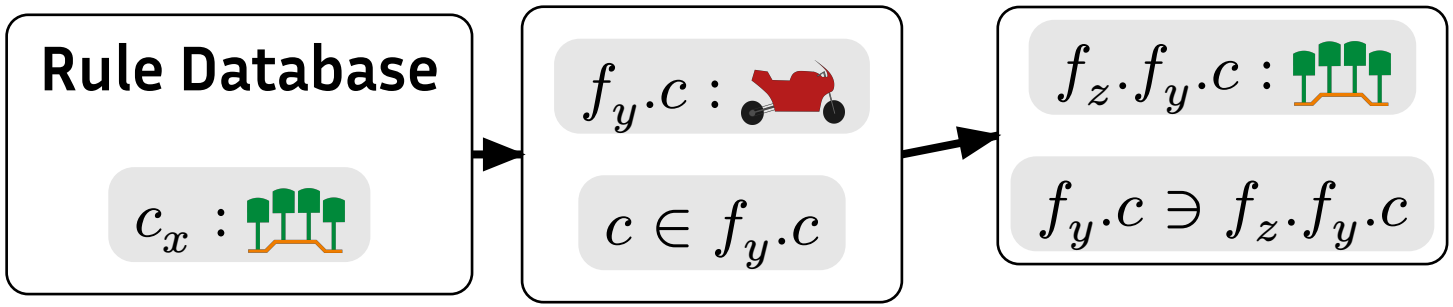
Finite Trigger Sequence extendable into an (infinite) Cyclicity Sequence

1. Start: Rule DB Trg
2. DB Trg is g-unblk.
3. Ldd, UC-unblk trgs
4. Repeat with cycl. t.
5. Induced CM is rev.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Consider the rule DB for (1): $c_x : \text{tree}$



4. Cyclicity Prefixes

Finite Trigger Sequence extendable into an (infinite) Cyclicity Sequence

1. Start: Rule DB Trg
2. DB Trg is g-unblk.
3. Ldd, UC-unblk trgs
4. Repeat with cycl. t.
5. Induced CM is rev.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Consider the rule DB for (1): $c_x : \text{tree}$

Rule Database

$c_x : \text{tree}$

$f_y.c : \text{bike}$

$c \in f_y.c$

$f_z.f_y.c : \text{tree}$

$f_y.c \ni f_z.f_y.c$

$f_y.f_z.f_y.c : \text{bike}$

$f_z.f_y.c \in f_y.f_z.f_y.c$

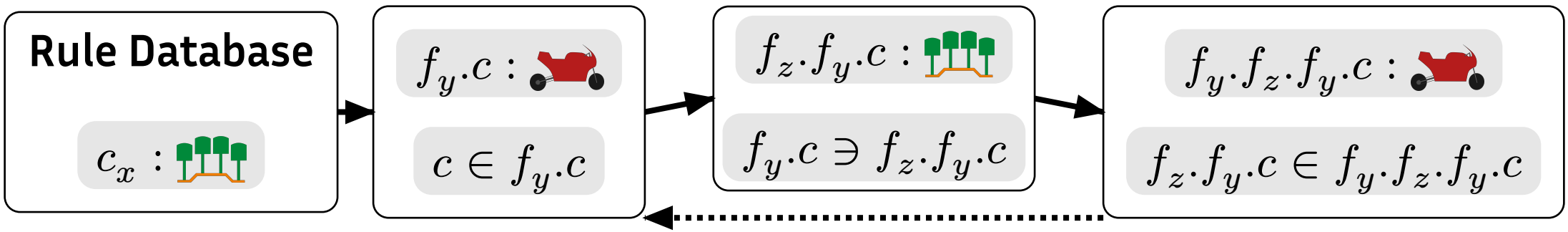
4. Cyclicity Prefixes

Finite Trigger Sequence extendable into an (infinite) Cyclicity Sequence

1. Start: Rule DB Trg
2. DB Trg is g-unblk.
3. Ldd, UC-unblk trgs
4. Repeat with cycl. t.
5. Induced CM is rev.

Theorem.
Every cyclicity prefix can be extended into a cyclicity sequence implying never-term.

Consider the rule DB for (1): c_x : 



5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

$$x : \text{  } \rightarrow \Big(\exists y.\ y : \text{  } \wedge x \in y \Big) \vee x : \text{  } \quad (1)$$

$$x : \text{  } \rightarrow \exists z.\ z : \text{  } \wedge x \ni z \quad (2)$$

5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

Each rule ρ and HC

$$x : \text{tree} \rightarrow \Big(\exists y.\ y : \text{bike} \wedge x \in y \Big) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Consider the rule DB for (1): $c_x : \text{tree}$

Rule Database

$c_x : \text{tree}$

5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

Each rule ρ and HC

1. Apply Rule-DB Trg

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y \right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Consider the rule DB for (1): $c_x : \text{tree}$

Rule Database

$c_x : \text{tree}$

$f_y.c : \text{bike}$

$c \in f_y.c$

5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

Each rule ρ and HC
1. Apply Rule-DB Trg
2. Ldd, UC-unblk trgs w/o cycl. until cycl.

$$x : \text{tree} \rightarrow \left(\exists y.\ y : \text{bike} \wedge x \in y\right) \vee x : \text{wrench} \quad (1)$$

$$x : \text{bike} \rightarrow \exists z.\ z : \text{tree} \wedge x \ni z \quad (2)$$

Consider the rule DB for (1): $c_x : \text{tree}$

Rule Database

$c_x : \text{tree}$

$f_y.c : \text{bike}$

$c \in f_y.c$

$f_z.f_y.c : \text{tree}$

$f_y.c \ni f_z.f_y.c$

5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

Each rule ρ and HC
1. Apply Rule-DB Trg
2. Ldd, UC-unblk trgs w/o cycl. until cycl.
3. For ρ: Subs is inj.

$$x : \blacktriangle \rightarrow \left(\exists y.\ y : \blacktriangleright \wedge x \in y\right) \vee x : \blacktriangleleft \quad (1)$$

$$x : \blacktriangleright \rightarrow \exists z.\ z : \blacktriangle \wedge x \ni z \quad (2)$$

Consider the rule DB for (1): $c_x : \blacktriangle$

Rule Database

$c_x : \blacktriangle$

$f_y.c : \blacktriangleright$

$c \in f_y.c$

$f_z.f_y.c : \blacktriangle$

$f_y.c \ni f_z.f_y.c$

$f_y.f_z.f_y.c : \blacktriangleright$

$f_z.f_y.c \in f_y.f_z.f_y.c$

5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

Each rule ρ and HC
1. Apply Rule-DB Trg
2. Ldd, UC-unblk trgs w/o cycl. until cycl.
3. For ρ: Subs is inj.

$$x : \blacktriangle \rightarrow \Big(\exists y.\ y : \blacktriangleright\!\!\blacktriangleleft \wedge\ x \in y\Big) \vee x : \text{🔧} \qquad (1)$$

$$x : \blacktriangleright\!\!\blacktriangleleft \rightarrow \exists z.\ z : \blacktriangle \wedge\ x \ni z \qquad (2)$$

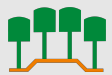
Consider the rule DB for (1): $c_x : \blacktriangle$

Extracted triggers: $\langle (1), [x/c_x] \rangle$ $\langle (2), [x/f_y.c_x] \rangle$ $\langle (1), [x/f_z.f_y.c_x] \rangle$

5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

Each rule ρ and HC
1. Apply Rule-DB Trg
2. Ldd, UC-unblk trgs w/o cycl. until cycl.
3. For ρ: Subs is inj.

$$x : \text{  } \rightarrow \Big(\exists y.\ y : \text{  } \wedge\ x \in y\Big) \vee x : \text{  } \quad (1)$$

$$x : \text{  } \rightarrow \exists z.\ z : \text{  } \wedge\ x \ni z \quad (2)$$

Consider the rule DB for (1): c_x : 

Extracted triggers: $\langle (1), [x/c_x] \rangle$ $\langle (2), [x/f_y.c_x] \rangle$ $\langle (1), [x/f_z.f_y.c_x] \rangle$

We just saw that this is a Cyclicity Prefix!

5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

Each rule ρ and HC
1. Apply Rule-DB Trg
2. Ldd, UC-unblk trgs w/o cycl. until cycl.
3. For ρ: Subs is inj.

Theorem.
If a rule set is RPC, it never-terminates.

Extracted triggers: $\langle (1), [x/c_x] \rangle$ $\langle (2), [x/f_y.c_x] \rangle$ $\langle (1), [x/f_z.f_y.c_x] \rangle$

We just saw that this is a Cyclicity Prefix!

5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

Cyclicity Prefix:

1. Start: Rule DB Trg
2. DB Trg is g-unblk.
3. Ldd, UC-unblk trgs
4. Repeat with cycl. t.
5. Induced CM is rev.

Theorem.
If a rule set is RPC, it never-terminates.

Extracted triggers: $\langle (1), [x/c_x] \rangle$ $\langle (2), [x/f_y.c_x] \rangle$ $\langle (1), [x/f_z.f_y.c_x] \rangle$

We just saw that this is a Cyclicity Prefix!

5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

Cyclicity Prefix:

1. Start: Rule DB Trg
2. DB Trg is g-unblk.
3. Ldd, UC-unblk trgs
4. Repeat with cycl. t.
5. Induced CM is rev.

Theorem.
If a rule set is RPC, it never-terminates.
We (almost) get a cyclicity prefix by construction.

Extracted triggers: $\langle (1), [x/c_x] \rangle$ $\langle (2), [x/f_y.c_x] \rangle$ $\langle (1), [x/f_z.f_y.c_x] \rangle$

We just saw that this is a Cyclicity Prefix!

5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

Cyclicity Prefix:

1. Start: Rule DB Trg
2. **DB Trg is g-unblk.**
3. Ldd, UC-unblk trgs
4. Repeat with cycl. t.
5. **Induced CM is rev.**

Theorem.
If a rule set is RPC, it never-terminates.
*We (almost) get a cyclicity prefix by construction. Points **2** and **5** need more love!*

Extracted triggers: $\langle (1), [x/c_x] \rangle$ $\langle (2), [x/f_y.c_x] \rangle$ $\langle (1), [x/f_z.f_y.c_x] \rangle$

We just saw that this is a Cyclicity Prefix!

5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

Cyclicity Prefix:

1. Start: Rule DB Trg
2. **DB Trg is g-unblk.**
3. Ldd, UC-unblk trgs
4. Repeat with cycl. t.
5. **Induced CM is rev.**

Theorem.
If a rule set is RPC, it never-terminates.
*We (almost) get a cyclicity prefix by construction. Points **2** and **5** need more love!*
***2** follows from uc-unblk of last trigger.*

Extracted triggers: $\langle (1), [x/c_x] \rangle$ $\langle (2), [x/f_y.c_x] \rangle$ $\langle (1), [x/f_z.f_y.c_x] \rangle$

We just saw that this is a Cyclicity Prefix!

5. Restricted Prefix Cyclicity (RPC)

RPC is a computational procedure for (possibly) finding a Cyclicity Prefix:

Cyclicity Prefix:

1. Start: Rule DB Trg
2. **DB Trg is g-unblk.**
3. Ldd, UC-unblk trgs
4. Repeat with cycl. t.
5. **Induced CM is rev.**

Theorem.
If a rule set is RPC, it never-terminates.
5 *follows from injectivity of substitutions in trgs with* ρ *and the observation that the first functional term is a subterm of every other.*

Extracted triggers: $\langle (1), [x/c_x] \rangle$ $\langle (2), [x/f_y.c_x] \rangle$ $\langle (1), [x/f_z.f_y.c_x] \rangle$

We just saw that this is a Cyclicity Prefix!

Summary & Outlook

- Define RPC (and DRPC) as cyclicity notions for the restricted chase in a framework-like manner with additional improvements over RMFC.

Summary & Outlook

- Define RPC (and DRPC) as cyclicity notions for the restricted chase in a framework-like manner with additional improvements over RMFC.
- Empirically verify the generality of the notions.

Summary & Outlook

- Define RPC (and DRPC) as cyclicity notions for the restricted chase in a framework-like manner with additional improvements over RMFC.
- Empirically verify the generality of the notions.
- Use information from checks to automatically explain and fix potential modelling mistakes that lead to non-termination.

Summary & Outlook

- Define RPC (and DRPC) as cyclicity notions for the restricted chase in a framework-like manner with additional improvements over RMFC.
- Empirically verify the generality of the notions.
- Use information from checks to automatically explain and fix potential modelling mistakes that lead to non-termination.
- The framework can be extended even further.

Summary & Outlook

- Define RPC (and DRPC) as cyclicity notions for the restricted chase in a framework-like manner with additional improvements over RMFC.
- Empirically verify the generality of the notions.
- Use information from checks to automatically explain and fix potential modelling mistakes that lead to non-termination.
- The framework can be extended even further.
- It would be interesting to find a proper counterexample for RMFC. (We have something in mind but it's hard to verify.)

Summary & Outlook

- Define RPC (and DRPC) as cyclicity notions for the restricted chase in a framework-like manner with additional improvements over RMFC.
- Empirically verify the generality of the notions.
- Use information from checks to automatically explain and fix potential modelling mistakes that lead to non-termination.
- The framework can be extended even further.
- It would be interesting to find a proper counterexample for RMFC. (We have something in mind but it's hard to verify.)

These slides are built using https://typst.app with the Polylux and CeTZ packages; give it a try!

Summary & Outlook

- Define RPC (and DRPC) as cyclicity notions for the restricted chase in a framework-like manner with additional improvements over RMFC.
- Empirically verify the generality of the notions.
- Use information from checks to automatically explain and fix potential modelling mistakes that lead to non-termination.
- The framework can be extended even further.
- It would be interesting to find a proper counterexample for RMFC. (We have something in mind but it's hard to verify.)

These slides are built using https://typst.app with the Polylux and CeTZ packages; give it a try! 😎

(If you are fine with not having colored emojis in PDF exports yet. 😉*)*

References

[GM14] T. Gogacz, and J. Marcinkowski, "All-instances termination of chase is undecidable," in *Proc. 41st Int. Colloq. Automata, Languages, Program. (ICALP 2014), Denmark, Part Ii* in Lecture Notes in Computer Science, vol. 8573, 2014, pp. 293–304.

[Bou+16] P. Bourhis, M. Manna, M. Morak, and A. Pieris, "Guarded-based disjunctive tuple-generating dependencies," *ACM Trans. Database Syst. (Tods)*, vol. 41, no. 4, pp. 1–45, 2016.

[CDK17] D. Carral, I. Dragoste, and M. Krötzsch, "Restricted chase (non)termination for existential rules with disjunctions," in *Proc. 26th Int. Joint Conf. Artif. Intell. (IJCAI 2017), Aust.*, 2017, pp. 922–928.

[GO18] G. Grahne, and A. Onet, “Anatomy of the chase,” *Fundamenta Informaticae*, vol. 157, no. 3, pp. 221–270, 2018.

[GC23] L. Gerlach, and D. Carral, “General acyclicity and cyclicity notions for the disjunctive skolem chase,” *Proc. AAAI Conf. Artif. Intell.*, vol. 37, no. 5, pp. 6372–6379, Jun. 2023, doi: 10.1609/aaai.v37i5.25784. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/25784

[BV81] C. Beeri, and M. Y. Vardi, “The implication problem for data dependencies,” in *Proc. 8th Int. Colloq. Automata, Languages Program. (ICALP 1981), Isr.* in Lecture Notes in Computer Science, vol. 115, 1981, pp. 73–85.

3.' Unblockability Propagates

3.' Unblockability Propagates

If a trigger is unblockable, then so are *similar repeated* triggers.

3.' Unblockability Propagates

If a trigger is unblockable, then so are *similar repeated* triggers.

A *constant mapping* g replaces constants with terms in expressions.

3.' Unblockability Propagates

If a trigger is unblockable, then so are *similar repeated* triggers.

A *constant mapping* g replaces constants with terms in expressions.

Consider a term set T closed under subterms; g is *reversible* for T if

3.' Unblockability Propagates

If a trigger is unblockable, then so are *similar repeated* triggers.

A *constant mapping* g replaces constants with terms in expressions.

Consider a term set T closed under subterms; g is *reversible* for T if

1. g is defined for every constant in T,

3.' Unblockability Propagates

If a trigger is unblockable, then so are *similar repeated* triggers.

A *constant mapping* g replaces constants with terms in expressions.

Consider a term set T closed under subterms; g is *reversible* for T if

1. g is defined for every constant in T,
2. $g(s) \neq g(t)$ for every $s, t \in T$ with $s \neq t$, and

3.' Unblockability Propagates

If a trigger is unblockable, then so are *similar repeated* triggers.

A *constant mapping* g replaces constants with terms in expressions.

Consider a term set T closed under subterms; g is *reversible* for T if

1. g is defined for every constant in T,
2. $g(s) \neq g(t)$ for every $s, t \in T$ with $s \neq t$, and
3. for every constant $c \in T$, every subterm s of $g(c)$, and every functional term $u \in T$; we have $g(u) \neq s$.

3.' Unblockability Propagates

If a trigger is unblockable, then so are *similar repeated* triggers.

A *constant mapping* g replaces constants with terms in expressions.

Consider a term set T closed under subterms; g is *reversible* for T if

1. g is defined for every constant in T,
2. $g(s) \neq g(t)$ for every $s, t \in T$ with $s \neq t$, and
3. for every constant $c \in T$, every subterm s of $g(c)$, and every functional term $u \in T$; we have $g(u) \neq s$.

If $\langle \rho, \sigma \rangle$ is uc-unblk and g rev for its skeleton, then $\langle \rho, g \circ \sigma \rangle$ is uc-unblk.

Why condition 2 for reversibility?

Rev. Cond. 2: We have $g(s) \neq g(t)$ for every $s, t \in T$ with $s \neq t$.

Why condition 2 for reversibility?

Rev. Cond. 2: We have $g(s) \neq g(t)$ for every $s, t \in T$ with $s \neq t$.

$$x \xrightarrow{\rho} y \rightarrow \exists u.\ x \xrightarrow{\gamma} u \wedge y \xrightarrow{\beta} u \tag{1}$$

$$x \xrightarrow{\gamma} y \rightarrow \exists v.\ y \xrightarrow{\alpha} v \tag{2}$$

$$x \xrightarrow{\gamma} y \wedge x \xrightarrow{\beta} y \rightarrow y \xrightarrow{\alpha} x \tag{3}$$

$$x \xrightarrow{\alpha} y \rightarrow y \xrightarrow{\rho} y \tag{4}$$

Why condition 2 for reversibility?

Rev. Cond. 2: We have $g(s) \neq g(t)$ for every $s, t \in T$ with $s \neq t$.

$$x \xrightarrow{R} y \rightarrow \exists u.\ x \xrightarrow{G} u \wedge y \xrightarrow{B} u \tag{1}$$

$$x \xrightarrow{G} y \rightarrow \exists v.\ y \xrightarrow{O} v \tag{2}$$

$$x \xrightarrow{G} y \wedge x \xrightarrow{B} y \rightarrow y \xrightarrow{O} x \tag{3}$$

$$x \xrightarrow{O} y \rightarrow y \xrightarrow{R} y \tag{4}$$

The trigger $\langle (2), [x/c, y/f_1(c,d)] \rangle$ is uc-unblockable.

Why condition 2 for reversibility?

Rev. Cond. 2: We have $g(s) \neq g(t)$ for every $s, t \in T$ with $s \neq t$.

$$x \xrightarrow{R} y \rightarrow \exists u.\ x \xrightarrow{G} u \wedge y \xrightarrow{B} u \qquad (1)$$

$$x \xrightarrow{G} y \rightarrow \exists v.\ y \xrightarrow{O} v \qquad (2)$$

$$x \xrightarrow{G} y \wedge x \xrightarrow{B} y \rightarrow y \xrightarrow{O} x \qquad (3)$$

$$x \xrightarrow{O} y \rightarrow y \xrightarrow{R} y \qquad (4)$$

The trigger $\langle (2), [x/c, y/f_1(c,d)] \rangle$ is uc-unblockable.
The trigger $\langle (2), [x/f_2(f_1(c,d)), y/f_1(f_2(\ldots), f_2(\ldots))] \rangle$ is not.

Why condition 2 for reversibility?

Rev. Cond. 2: We have $g(s) \neq g(t)$ for every $s, t \in T$ with $s \neq t$.

$$x \xrightarrow{R} y \rightarrow \exists u.\ x \xrightarrow{G} u \wedge y \xrightarrow{B} u \quad (1)$$

$$x \xrightarrow{G} y \rightarrow \exists v.\ y \xrightarrow{O} v \quad (2)$$

$$x \xrightarrow{G} y \wedge x \xrightarrow{B} y \rightarrow y \xrightarrow{O} x \quad (3)$$

$$x \xrightarrow{O} y \rightarrow y \xrightarrow{R} y \quad (4)$$

The trigger $\langle (2), [x/c, y/f_1(c,d)] \rangle$ is uc-unblockable.
The trigger $\langle (2), [x/f_2(f_1(c,d)), y/f_1(f_2(\ldots), f_2(\ldots))] \rangle$ is not.
Note that the second results from the first with the constant mapping that maps both c and d to $f_2(f_1(c,d))$ and thus violates cond. 2. ⚡

Why condition 3 for reversibility?

Rev. Cond. 3: For every constant $c \in T$, every subterm s of $g(c)$, and every functional term $u \in T$; we have $g(u) \neq s$.

Why condition 3 for reversibility?

Rev. Cond. 3: For every constant $c \in T$, every subterm s of $g(c)$, and every functional term $u \in T$; we have $g(u) \neq s$.

$$x : A \to \exists u.\ x \to u \qquad x \to y \to y : T \vee \exists z.\ R(x,y,z) \qquad (\dagger)$$

$$x : B \to \exists v.\ x \to v \qquad x \to y \to x : T$$

$$x : C \to \exists w.\ x \to w$$

Why condition 3 for reversibility?

Rev. Cond. 3: For every constant $c \in T$, every subterm s of $g(c)$, and every functional term $u \in T$; we have $g(u) \neq s$.

$$x : A \to \exists u.\ x \mathrel{\color{red}{\longrightarrow}} u \qquad x \mathrel{\color{red}{\longrightarrow}} y \to y : T \lor \exists z.\ R(x,y,z) \qquad (\dagger)$$

$$x : B \to \exists v.\ x \mathrel{\color{green}{\longrightarrow}} v \qquad x \mathrel{\color{green}{\longrightarrow}} y \to x : T$$

$$x : C \to \exists w.\ x \mathrel{\color{blue}{\longrightarrow}} w$$

The trigger $\langle (\dagger), [x/c, y/f_u(d)] \rangle$ is uc-unblockable.

Why condition 3 for reversibility?

Rev. Cond. 3: For every constant $c \in T$, every subterm s of $g(c)$, and every functional term $u \in T$; we have $g(u) \neq s$.

$$x : A \rightarrow \exists u.\ x \xrightarrow{} u \qquad x \xrightarrow{} y \rightarrow y : T \vee \exists z.\ R(x,y,z) \qquad (\dagger)$$

$$x : B \rightarrow \exists v.\ x \xrightarrow{} v \qquad x \xrightarrow{} y \rightarrow x : T$$

$$x : C \rightarrow \exists w.\ x \xrightarrow{} w$$

The trigger $\langle (\dagger), [x/c, y/f_u(d)] \rangle$ is uc-unblockable. Mapping c to $f_w(f_v(f_u(d)))$ and d to itself via g fulfills conditions 1 and 2 of reversibility but not 3 since $g(f_u(d)) = f_u(d)$ is a subterm of $g(c)$.

Why condition 3 for reversibility?

Rev. Cond. 3: For every constant $c \in T$, every subterm s of $g(c)$, and every functional term $u \in T$; we have $g(u) \neq s$.

$$x : A \to \exists u.\ x \xrightarrow{} u \qquad x \xrightarrow{} y \to y : T \vee \exists z.\ R(x,y,z) \qquad (\dagger)$$

$$x : B \to \exists v.\ x \xrightarrow{} v \qquad x \xrightarrow{} y \to x : T$$

$$x : C \to \exists w.\ x \xrightarrow{} w$$

The trigger $\langle(\dagger), [x/c, y/f_u(d)]\rangle$ is uc-unblockable. Mapping c to $f_w(f_v(f_u(d)))$ and d to itself via g fulfills conditions 1 and 2 of reversibility but not 3 since $g(f_u(d)) = f_u(d)$ is a subterm of $g(c)$. Indeed, $\langle(\dagger), [x/f_w(f_v(f_u(d))), y/f_u(d)]\rangle$ is not uc-unblockable! ⚡

Problematic RMFC Example

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u \quad (1)$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v \quad (2)$$

$$\not\square\ y : \bigcirc \wedge y \rightarrow w \rightarrow w : \bigcirc \quad (3)$$

Problematic RMFC Example

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \xrightarrow{r} u \wedge y \xrightarrow{r} u \quad (1)$$

$$x : \square \wedge x \xrightarrow{r} z \rightarrow \exists v.\ x \xrightarrow{g} v \wedge z \xrightarrow{b} v \quad (2)$$

$$\not\square\ y : \bigcirc \wedge y \xrightarrow{g} w \rightarrow w : \bigcirc \quad (3)$$

$$y \xrightarrow{r} z \wedge z \xrightarrow{b} w \wedge x \xrightarrow{g} w \rightarrow y \xrightarrow{g} y \wedge z \xrightarrow{b} y \wedge y : \square \quad (4)$$

Problematic RMFC Example

$$x : \square \land y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \land y \rightarrow u \quad (1)$$

$$x : \square \land x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \land z \rightarrow v \quad (2)$$

$$\not\square\ y : \bigcirc \land y \rightarrow w \rightarrow w : \bigcirc \quad (3)$$

$$y \rightarrow z \land z \rightarrow w \land x \rightarrow w \rightarrow y \rightarrow y \land z \rightarrow y \land y : \square \quad (4)$$

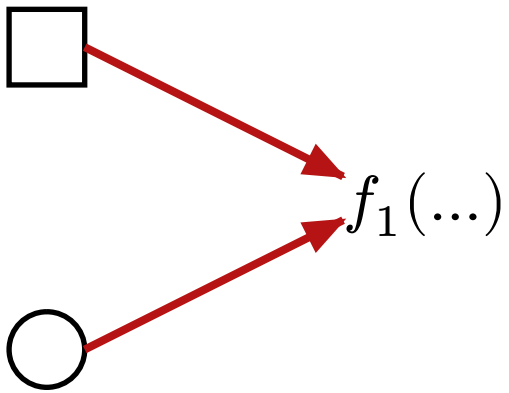
Problematic RMFC Example

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u \quad (1)$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v \quad (2)$$

$$\not\square\ y : \bigcirc \wedge y \rightarrow w \rightarrow w : \bigcirc \quad (3)$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square \quad (4)$$



$\langle (1), [x/c, y/d] \rangle$

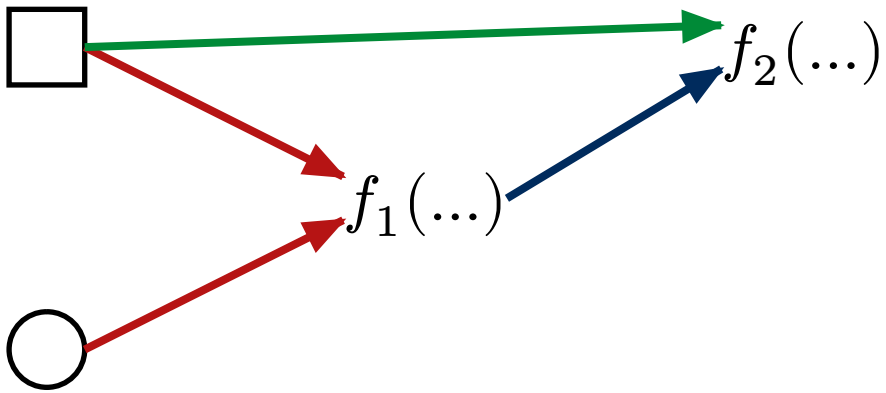
Problematic RMFC Example

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u \qquad (1)$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v \qquad (2)$$

$$\not\square\ y : \bigcirc \wedge y \rightarrow w \rightarrow w : \bigcirc \qquad (3)$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square \qquad (4)$$



$\langle (1), [x/c, y/d] \rangle$
$\langle (2), [x/c, y/f_1(c,d)] \rangle$

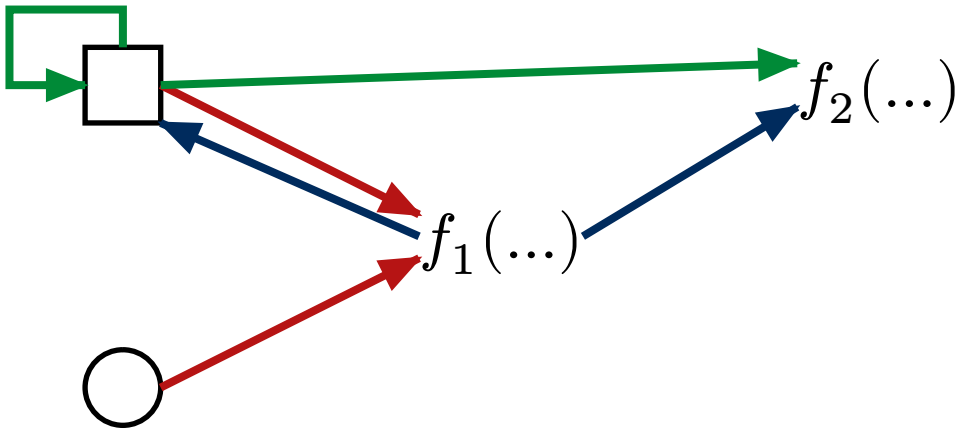
Problematic RMFC Example

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u \quad (1)$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v \quad (2)$$

$$\not\square\ y : \bigcirc \wedge y \rightarrow w \rightarrow w : \bigcirc \quad (3)$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square \quad (4)$$



$\langle (1), [x/c, y/d] \rangle$
$\langle (2), [x/c, y/f_1(c,d)] \rangle$
$\langle (4), [x/c, y/d, z/f_1(\ldots), w/f_2(\ldots)] \rangle$

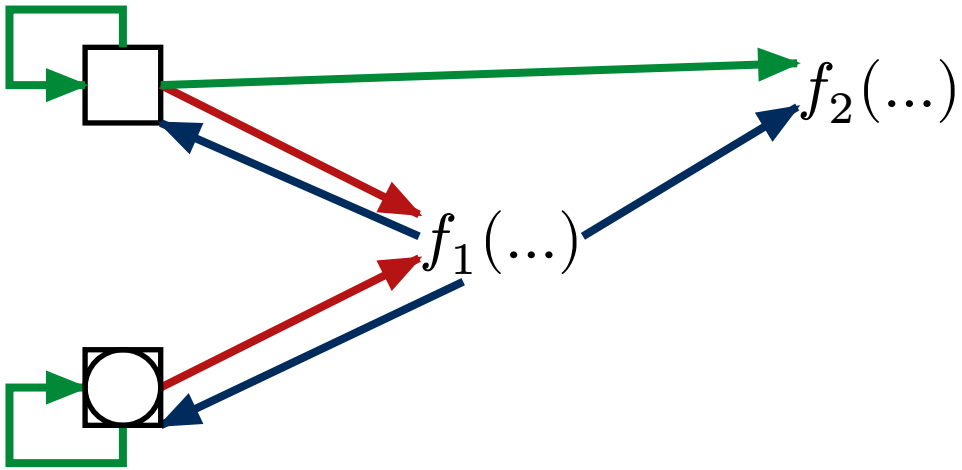
Problematic RMFC Example

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u \quad (1)$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v \quad (2)$$

$$\not\square\ y : \bigcirc \wedge y \rightarrow w \rightarrow w : \bigcirc \quad (3)$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square \quad (4)$$



$\langle (1), [x/c, y/d] \rangle$
$\langle (2), [x/c, y/f_1(c,d)] \rangle$
$\langle (4), [x/c, y/d, z/f_1(\ldots), w/f_2(\ldots)] \rangle$
$\langle (4), [x/d, y/d, z/f_1(\ldots), w/f_2(\ldots)] \rangle$

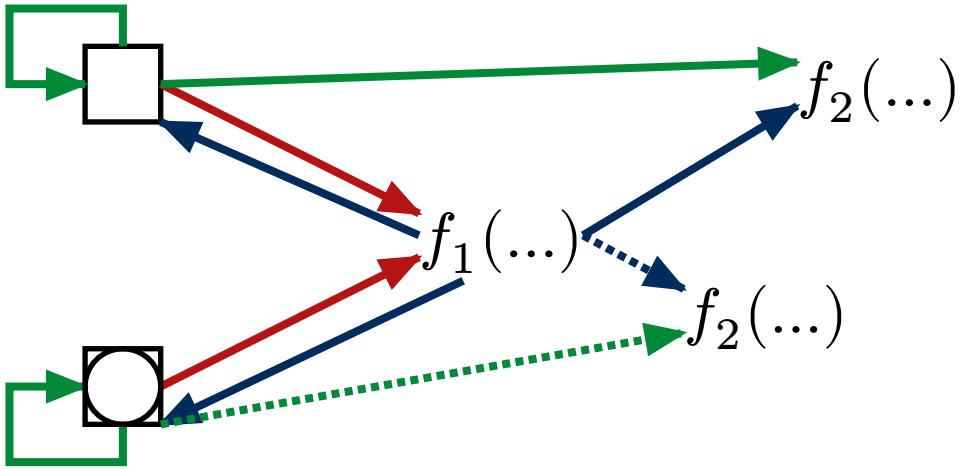
Problematic RMFC Example

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u \qquad (1)$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v \qquad (2)$$

$$\not\square\ y : \bigcirc \wedge y \rightarrow w \rightarrow w : \bigcirc \qquad (3)$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square \qquad (4)$$



$\langle (1), [x/c, y/d] \rangle$
$\langle (2), [x/c, y/f_1(c,d)] \rangle$
$\langle (4), [x/c, y/d, z/f_1(\ldots), w/f_2(\ldots)] \rangle$
$\langle (4), [x/d, y/d, z/f_1(\ldots), w/f_2(\ldots)] \rangle$
$\langle (2), [x/d, y/f_1(c,d)] \rangle$**?**

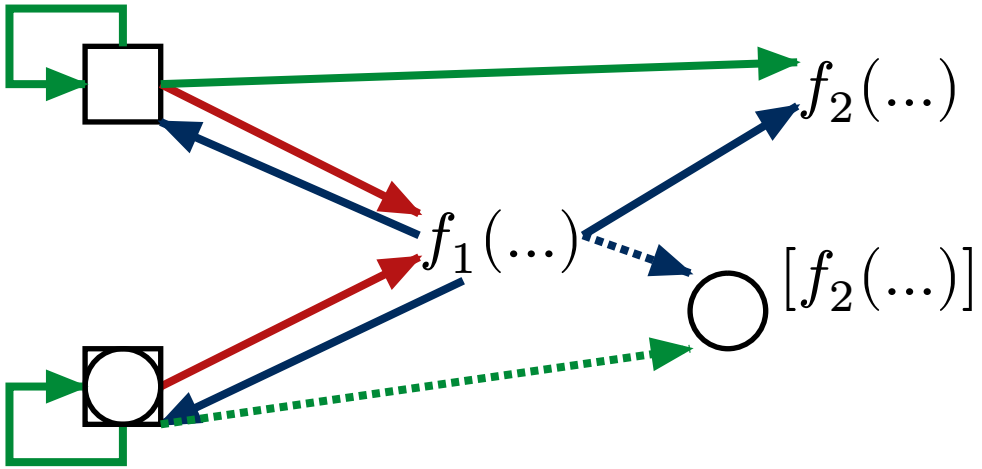
Problematic RMFC Example

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u \qquad (1)$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v \qquad (2)$$

$$\not\square\ y : \bigcirc \wedge y \rightarrow w \rightarrow w : \bigcirc \qquad (3)$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square \qquad (4)$$



$\langle (1), [x/c, y/d] \rangle$
$\langle (2), [x/c, y/f_1(c,d)] \rangle$
$\langle (4), [x/c, y/d, z/f_1(\ldots), w/f_2(\ldots)] \rangle$
$\langle (4), [x/d, y/d, z/f_1(\ldots), w/f_2(\ldots)] \rangle$
$\langle (2), [x/d, y/f_1(c,d)] \rangle$**?**
$\langle (3), [y/d, w/f_2(\ldots)] \rangle$**??**

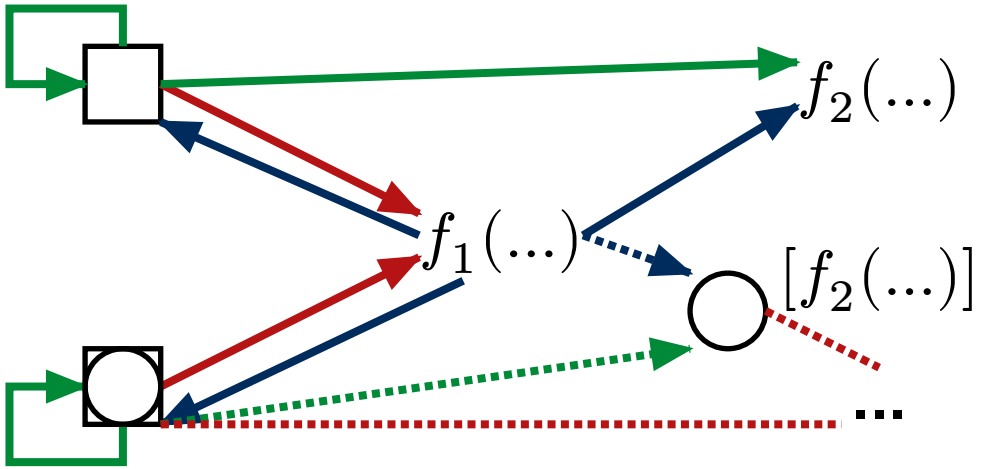
Problematic RMFC Example

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u \qquad (1)$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v \qquad (2)$$

$$\not\square\ y : \bigcirc \wedge y \rightarrow w \rightarrow w : \bigcirc \qquad (3)$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square \qquad (4)$$



$\langle (1), [x/c, y/d] \rangle$
$\langle (2), [x/c, y/f_1(c,d)] \rangle$
$\langle (4), [x/c, y/d, z/f_1(\ldots), w/f_2(\ldots)] \rangle$
$\langle (4), [x/d, y/d, z/f_1(\ldots), w/f_2(\ldots)] \rangle$
$\langle (2), [x/d, y/f_1(c,d)] \rangle$**?**
$\langle (3), [y/d, w/f_2(\ldots)] \rangle$**??**
$\langle (1), [x/d, y/f_2(\ldots)] \rangle$**???**

So does RPC fix our problem? (cont.)

$$x : \square \land y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \land y \rightarrow u$$

$$x : \square \land x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \land z \rightarrow v$$

$$y : \bigcirc \land y \rightarrow w \rightarrow w : \bigcirc$$

$$y \rightarrow z \land z \rightarrow w \land x \rightarrow w \rightarrow y \rightarrow y \land z \rightarrow y \land y : \square$$

So does RPC fix our problem? (cont.)

$$x : \square \land y : \bigcirc \rightarrow \exists u.\ x \xrightarrow{R} u \land y \xrightarrow{R} u$$

$$x : \square \land x \xrightarrow{R} z \rightarrow \exists v.\ x \xrightarrow{G} v \land z \xrightarrow{B} v$$

$$y : \bigcirc \land y \xrightarrow{G} w \rightarrow w : \bigcirc$$

$$y \xrightarrow{R} z \land z \xrightarrow{B} w \land x \xrightarrow{G} w \rightarrow y \xrightarrow{G} y \land z \xrightarrow{B} y \land y : \square$$

RPC ✗;

So does RPC fix our problem? (cont.)

$$x : \square \land y : \bigcirc \rightarrow \exists u.\ x \xrightarrow{r} u \land y \xrightarrow{r} u$$

$$x : \square \land x \xrightarrow{r} z \rightarrow \exists v.\ x \xrightarrow{g} v \land z \xrightarrow{b} v$$

$$y : \bigcirc \land y \xrightarrow{g} w \rightarrow w : \bigcirc$$

$$y \xrightarrow{r} z \land z \xrightarrow{b} w \land x \xrightarrow{g} w \rightarrow y \xrightarrow{g} y \land z \xrightarrow{b} y \land y : \square$$

RPC ☒; RMFC ☑;

So does RPC fix our problem? (cont.)

$$x : \square \land y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \land y \rightarrow u$$

$$x : \square \land x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \land z \rightarrow v$$

$$y : \bigcirc \land y \rightarrow w \rightarrow w : \bigcirc$$

$$y \rightarrow z \land z \rightarrow w \land x \rightarrow w \rightarrow y \rightarrow y \land z \rightarrow y \land y : \square$$

RPC ☒ ; RMFC ☑ ; Cycl. Sequence with $c : \square$, $d : \bigcirc$ ☒ ;

So does RPC fix our problem? (cont.)

$$x : \square \land y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \land y \rightarrow u$$

$$x : \square \land x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \land z \rightarrow v$$

$$y : \bigcirc \land y \rightarrow w \rightarrow w : \bigcirc$$

$$y \rightarrow z \land z \rightarrow w \land x \rightarrow w \rightarrow y \rightarrow y \land z \rightarrow y \land y : \square$$

RPC ✗ ; RMFC ☑ ; Cycl. Sequence with $c : \square$, $d : \bigcirc$ ✗ ; never-term. 🤔

So does RPC fix our problem? (cont.)

$$x : \square \land y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \land y \rightarrow u$$

$$x : \square \land x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \land z \rightarrow v$$

$$y : \bigcirc \land y \rightarrow w \rightarrow w : \bigcirc$$

$$y \rightarrow z \land z \rightarrow w \land x \rightarrow w \rightarrow y \rightarrow y \land z \rightarrow y \land y : \square$$

RPC ✗; RMFC ☑; Cycl. Sequence with $c : \square$, $d : \bigcirc$ ✗; never-term. 🤔

Cycl. Sequence with $d : \square$, $d : \bigcirc$ ☑;

So does RPC fix our problem? (cont.)

$$x : \square \land y : \bigcirc \rightarrow \exists u.\ x \to u \land y \to u$$

$$x : \square \land x \to z \rightarrow \exists v.\ x \to v \land z \to v$$

$$y : \bigcirc \land y \to w \rightarrow w : \bigcirc$$

$$y \to z \land z \to w \land x \to w \rightarrow y \to y \land z \to y \land y : \square$$

RPC ☒; RMFC ☑; Cycl. Sequence with $c : \square$, $d : \bigcirc$ ☒; never-term. 🤔

Cycl. Sequence with $d : \square$, $d : \bigcirc$ ☑; never-term. ☑ 🐕

So does RPC fix our problem? (cont.)

$$x : \square \land y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \land y \rightarrow u$$

$$x : \square \land x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \land z \rightarrow v$$

$$y : \bigcirc \land y \rightarrow w \rightarrow w : \bigcirc$$

$$y \rightarrow z \land z \rightarrow w \land x \rightarrow w \rightarrow y \rightarrow y \land z \rightarrow y \land y : \square$$

RPC ✗ ; RMFC ☑ ; Cycl. Sequence with $c : \square$, $d : \bigcirc$ ✗ ; never-term. 🤔

Cycl. Sequence with $d : \square$, $d : \bigcirc$ ☑ ; never-term. ☑ 🐕

Was RMFC right all along?

So does RPC fix our problem? (cont.)

$$x : \square \land y : \bigcirc \rightarrow \exists u.\ x \xrightarrow{R} u \land y \xrightarrow{R} u$$

$$x : \square \land x \xrightarrow{R} z \rightarrow \exists v.\ x \xrightarrow{G} v \land z \xrightarrow{B} v$$

$$y : \bigcirc \land y \xrightarrow{G} w \rightarrow w : \bigcirc$$

$$y \xrightarrow{R} z \land z \xrightarrow{B} w \land x \xrightarrow{G} w \rightarrow y \xrightarrow{G} y \land z \xrightarrow{B} y \land y : \square$$

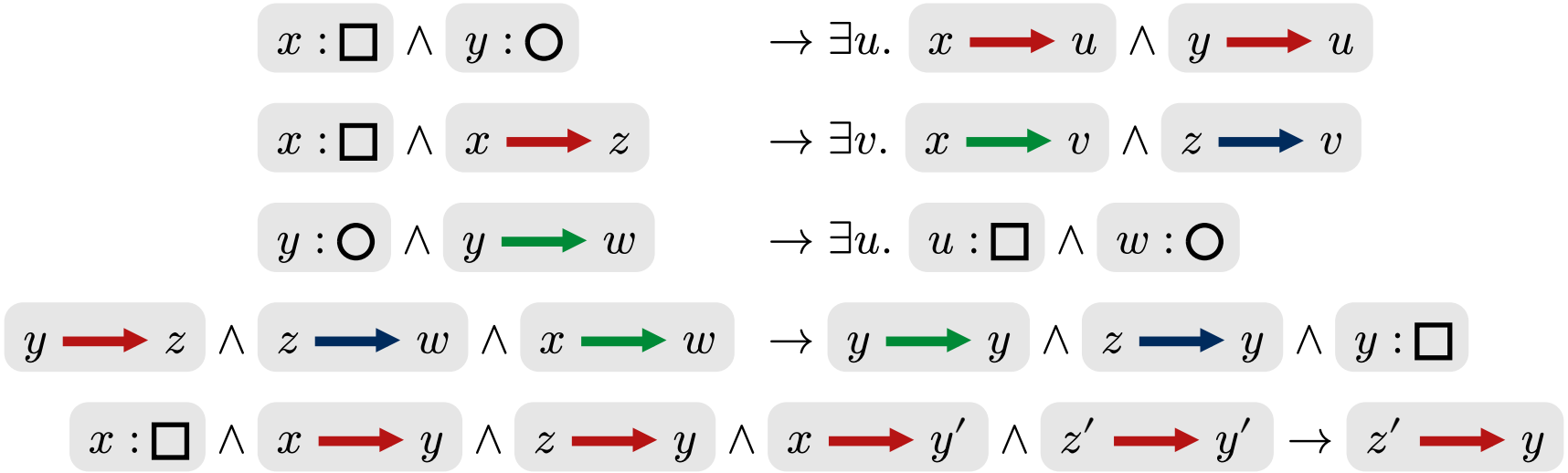
RPC ✗ ; RMFC ☑ ; Cycl. Sequence with $c : \square$, $d : \bigcirc$ ✗ ; never-term. 🤔

Cycl. Sequence with $d : \square$, $d : \bigcirc$ ☑ ; never-term. ☑ 🐕

Was RMFC right all along?

Frankly, we cannot say for sure; what we can say is that its correctness has not been proven.

Potential Counterexample for Never Termination



Potential Counterexample for Never Termination

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v$$

$$y : \bigcirc \wedge y \rightarrow w \rightarrow \exists u.\ u : \square \wedge w : \bigcirc$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square$$

$$x : \square \wedge x \rightarrow y \wedge z \rightarrow y \wedge x \rightarrow y' \wedge z' \rightarrow y' \rightarrow z' \rightarrow y$$

RPC ✗;

Potential Counterexample for Never Termination

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v$$

$$y : \bigcirc \wedge y \rightarrow w \rightarrow \exists u.\ u : \square \wedge w : \bigcirc$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square$$

$$x : \square \wedge x \rightarrow y \wedge z \rightarrow y \wedge x \rightarrow y' \wedge z' \rightarrow y' \rightarrow z' \rightarrow y$$

RPC ☒; RMFC ☑;

Potential Counterexample for Never Termination

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v$$

$$y : \bigcirc \wedge y \rightarrow w \rightarrow \exists u.\ u : \square \wedge w : \bigcirc$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square$$

$$x : \square \wedge x \rightarrow y \wedge z \rightarrow y \wedge x \rightarrow y' \wedge z' \rightarrow y' \rightarrow z' \rightarrow y$$

RPC ☒; RMFC ☑; Cycl. Sequence with $d : \square$, $d : \bigcirc$ ☒;

Potential Counterexample for Never Termination

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v$$

$$y : \bigcirc \wedge y \rightarrow w \rightarrow \exists u.\ u : \square \wedge w : \bigcirc$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square$$

$$x : \square \wedge x \rightarrow y \wedge z \rightarrow y \wedge x \rightarrow y' \wedge z' \rightarrow y' \rightarrow z' \rightarrow y$$

RPC ✗ ; RMFC ☑ ; Cycl. Sequence with $d : \square$, $d : \bigcirc$ ✗ ; never-term. 🤔

Potential Counterexample for Never Termination

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v$$

$$y : \bigcirc \wedge y \rightarrow w \rightarrow \exists u.\ u : \square \wedge w : \bigcirc$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square$$

$$x : \square \wedge x \rightarrow y \wedge z \rightarrow y \wedge x \rightarrow y' \wedge z' \rightarrow y' \rightarrow z' \rightarrow y$$

RPC ✗; RMFC ☑; Cycl. Sequence with $d : \square$, $d : \bigcirc$ ✗; never-term. 🤔

We think that every DB admits a terminating chase tree.

Potential Counterexample for Never Termination

$$x : \square \wedge y : \bigcirc \rightarrow \exists u.\ x \rightarrow u \wedge y \rightarrow u$$

$$x : \square \wedge x \rightarrow z \rightarrow \exists v.\ x \rightarrow v \wedge z \rightarrow v$$

$$y : \bigcirc \wedge y \rightarrow w \rightarrow \exists u.\ u : \square \wedge w : \bigcirc$$

$$y \rightarrow z \wedge z \rightarrow w \wedge x \rightarrow w \rightarrow y \rightarrow y \wedge z \rightarrow y \wedge y : \square$$

$$x : \square \wedge x \rightarrow y \wedge z \rightarrow y \wedge x \rightarrow y' \wedge z' \rightarrow y' \rightarrow z' \rightarrow y$$

RPC ✗; RMFC ☑; Cycl. Sequence with $d : \square$, $d : \bigcirc$ ✗; never-term. 🤔

We think that every DB admits a terminating chase tree. (It could still be that RMFC guarantees "sometimes non-termination" for exotic DBs.)