

DCR Execution Equivalence

Søren Debois & Axel Christfort

February 6, 2026

Abstract

We present an Isabelle formalization of the basics of DCR-graphs [1] before defining *Execution Equivalent* markings. We then prove that execution equivalent markings are perfectly interchangeable during process execution, yielding significant state-space reduction for execution-based model-checking of DCR graphs.

Contents

1 DCR processes	1
1.1 Execution semantics	2
1.2 Execution Equivalence	3

```
theory DCRExecutionEquivalence
  imports Main
begin
```

1 DCR processes

Although we use the term "process", the present theory formalises DCR graphs as defined in the original places and other papers.

type-synonym *event* = *nat*

The static structure. This encompasses the relations, the set of event *dom* of the process, and the labelling function *lab*. We do not explicitly enforce that relations and marking are confined to this set, except in definitions of enabledness and execution below.

record *rels* =

```
cond :: event rel
pend :: event rel
incl :: event rel
excl :: event rel
```

mist :: event rel

dom :: event set

The dynamic structure, called the marking

record *marking* =

Ex :: event set

In :: event set

Re :: event set

It will be convenient to have notation for the events required, excluded, etc. by a given event.

abbreviation *conds* :: rels \Rightarrow event \Rightarrow event set

where

$conds\ T\ e \equiv \{ f . (f, e) \in cond\ T \}$

abbreviation *excls* :: rels \Rightarrow event \Rightarrow event set

where

$excls\ T\ e \equiv \{ x . (e, x) \in excl\ T \wedge (e, x) \notin incl\ T \}$

abbreviation *incls* :: rels \Rightarrow event \Rightarrow event set

where

$incls\ T\ e \equiv \{ x . (e, x) \in incl\ T \}$

abbreviation *resps* :: rels \Rightarrow event \Rightarrow event set

where

$resps\ T\ e \equiv \{ f . (e, f) \in pend\ T \}$

abbreviation *mists* :: rels \Rightarrow event \Rightarrow event set

where

$mists\ T\ e \equiv \{ f . (f, e) \in mist\ T \}$

Similarly, it is convenient to be able to identify directly the currently excluded events.

1.1 Execution semantics

definition *enabled* :: rels \Rightarrow marking \Rightarrow event \Rightarrow bool

where

$enabled\ T\ M\ e \equiv$

$e \in In\ M \wedge$

$(conds\ T\ e \cap In\ M) - Ex\ M = \{\}$ \wedge

$(mists\ T\ e \cap In\ M) - (dom\ T - Re\ M) = \{\}$

definition *execute* :: rels \Rightarrow marking \Rightarrow nat \Rightarrow marking

where

$$\begin{aligned}
\text{execute } T M e &\equiv \langle \! \langle \\
& \quad Ex = Ex M \cup \{ e \}, \\
& \quad In = (In M - \text{excls } T e) \cup \text{incls } T e, \\
& \quad Re = (Re M - \{ e \}) \cup \text{resps } T e \\
& \rangle \! \rangle
\end{aligned}$$

1.2 Execution Equivalence

definition *accepting* :: marking \Rightarrow bool **where**
accepting $M = (Re M \cap In M = \{\})$

fun *acceptingrun* :: rels \Rightarrow marking \Rightarrow event list \Rightarrow bool **where**
acceptingrun $T M [] = \text{accepting } M$
 $| \text{acceptingrun } T M (e\#t) = (\text{enabled } T M e \wedge \text{acceptingrun } T (\text{execute } T M e) t)$

definition *all-conds* :: rels \Rightarrow nat set **where**
all-conds $T = \{ \text{fst } rel \mid rel . rel \in \text{cond } T \}$

definition *execution-equivalent* :: rels \Rightarrow marking \Rightarrow marking \Rightarrow bool **where**
execution-equivalent $T M1 M2 = ($
 $(In M1 = In M2) \wedge$
 $(Re M1 = Re M2) \wedge$
 $((Ex M1 \cap \text{all-conds } T) = (Ex M2 \cap \text{all-conds } T))$
 $)$

lemma *conds-subset-eq-all-conds*: $\text{conds } T e \subseteq \text{all-conds } T$
 $\langle \text{proof} \rangle$

lemma *ex-equiv-over-cond*: $(Ex M1 \cap \text{all-conds } T) = (Ex M2 \cap \text{all-conds } T) \implies$
 $(Ex M1 \cap \text{conds } T e) = (Ex M2 \cap \text{conds } T e)$
 $\langle \text{proof} \rangle$

lemma *enabled-ex-equiv*:
assumes *execution-equivalent* $T M1 M2$ *enabled* $T M1 e$
shows *enabled* $T M2 e$
 $\langle \text{proof} \rangle$

lemma *execute-ex-equiv*:
assumes *execution-equivalent* $T M1 M2$ *execute* $T M1 e = M3$ *execute* $T M2 e = M4$
shows *execution-equivalent* $T M3 M4$
 $\langle \text{proof} \rangle$

lemma *accepting-ex-equiv*: *execution-equivalent* $T M1 M2 \implies \text{accepting } M1 \implies$
 $\text{accepting } M2$
 $\langle \text{proof} \rangle$

theorem *acceptingrun-ex-equiv*:
assumes *acceptingrun* $T M1$ *seq execution-equivalent* $T M1 M2$

shows *acceptingrun T M2 seq*
<proof>

end

References

- [1] C. O. Back, T. Slaats, T. T. Hildebrandt, and M. Marquard. Discover: accurate and efficient discovery of declarative process models. *International Journal on Software Tools for Technology Transfer*, pages 1–25, 2021.