

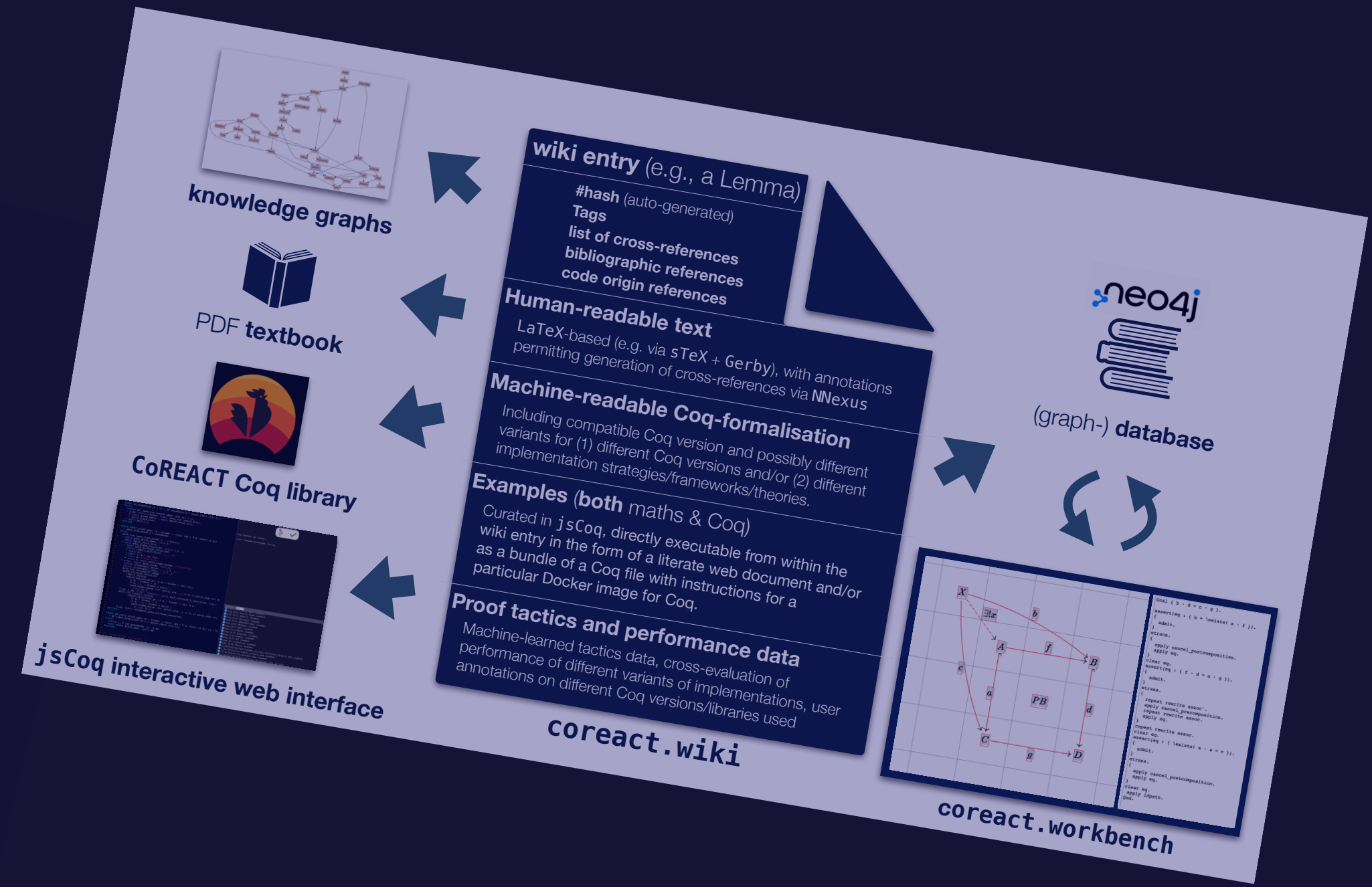
CoREACT

Coq-based Rewriting: towards Executable Applied Category Theory

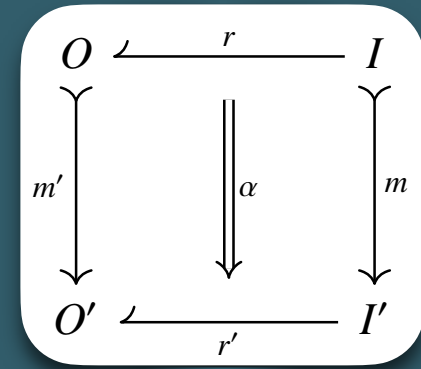
Consortium: IRIF (UP), LIP (ENS-Lyon), LIX (École Polytechnique), Sophia-Antipolis (Inria)



```
83
84
85 (** * Species Sum/Coproduct *)
86
87 Definition spec_sum (X Y : Species) : Species
88   := ((X.1 + Y.1)%type; sum_rect _ X.2 Y.2).
89
90 Lemma sigma_functor_sum (X : Type) (P Q : X -> Type) :
91   ({x : X & P x} + {x : X & Q x}) <~> {x : X & (P x + Q x)%type}.
92 Proof.
93   refine (equiv_adjointify _ _ _).
94   - intros [x w] | [x w]; exists x; [left | right]; apply w.
95   - intros [x [w | w]]; [left | right]; apply (x; w).
96   - intros [x [w | w]]; reflexivity.
97   - intros [[x w] | [x w]]; reflexivity.
98 Defined.
99
100 Definition stuff_spec_sum (P Q : FinSet -> Type) := fun A => (P A + Q A)%type.
101
102 Lemma stuff_spec_sum_correct (P Q : FinSet -> Type) :
103   spec_from_stuff (stuff_spec_sum P Q)
104   =
```



compositional rewriting double categories (crDCs)

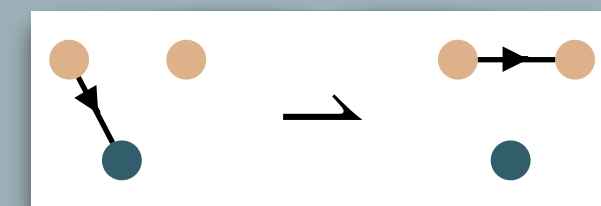
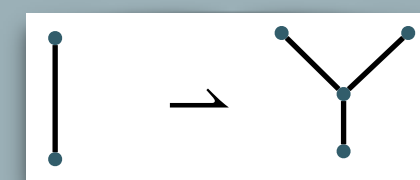
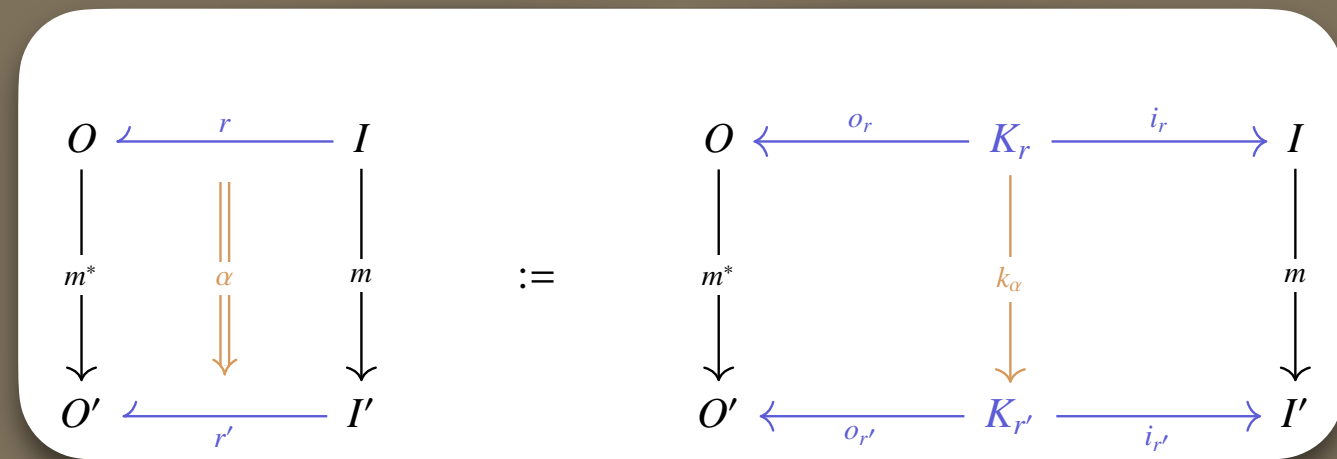


Fundamentals of Compositional Rewriting Theory*

Nicolas Behr^{a,*}, Russell Harmer^b, Jean Krivine^a

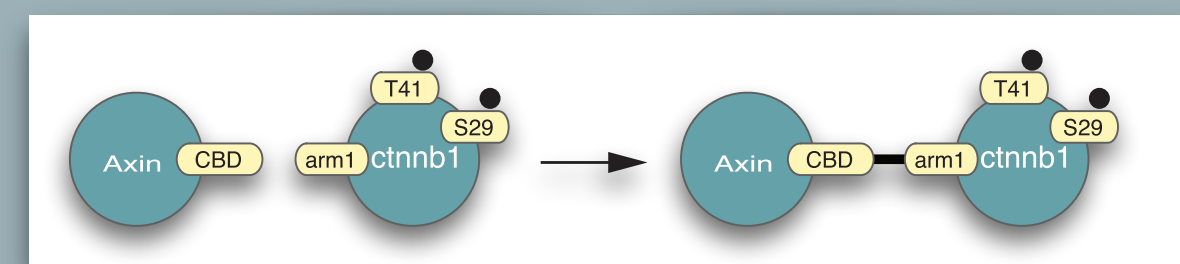
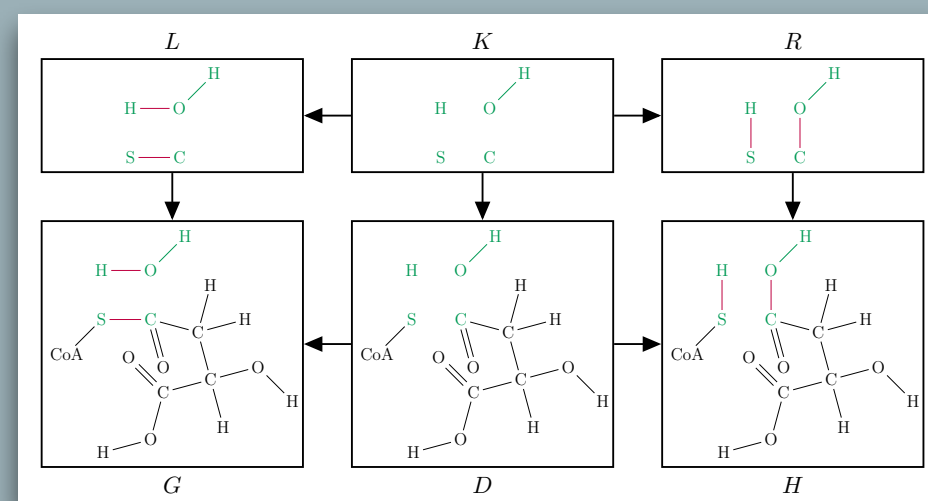
^aUniversité Paris Cité, CNRS, IRIF, 8 Place Aurélie Nemours, Paris Cedex 13, 75205, France

^bUniversité de Lyon, ENS de Lyon, UCBL, CNRS, LIP, 46 allée d'Italie, Lyon Cedex 07, 69364, France



Explicit rewriting semantics (DPO, SqPO, ...)

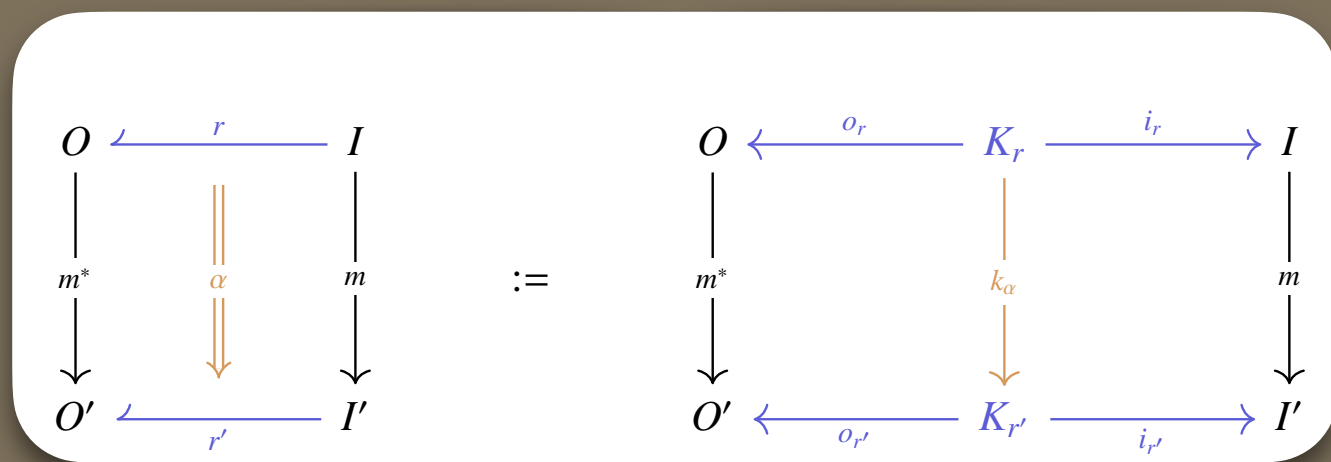
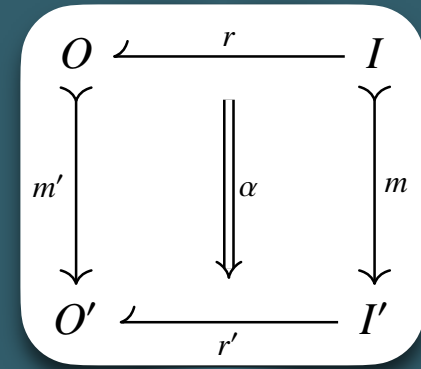
organic chemistry



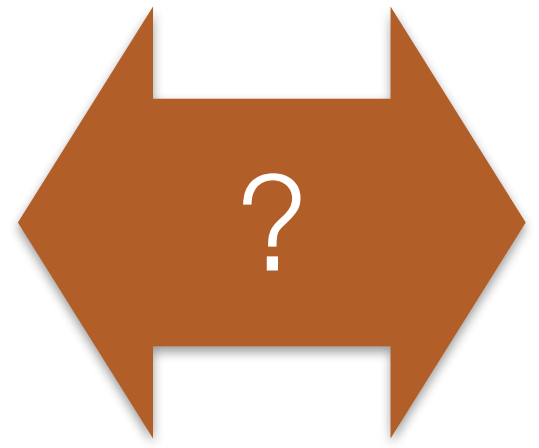
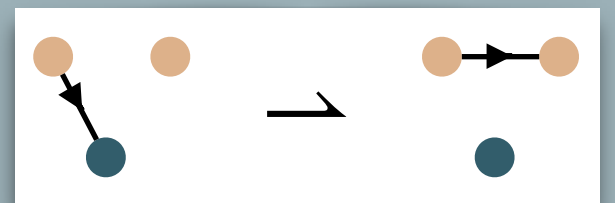
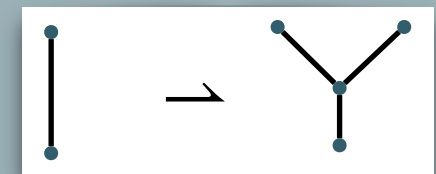
biochemistry

Instantiations of rewriting semantics in theory and applications

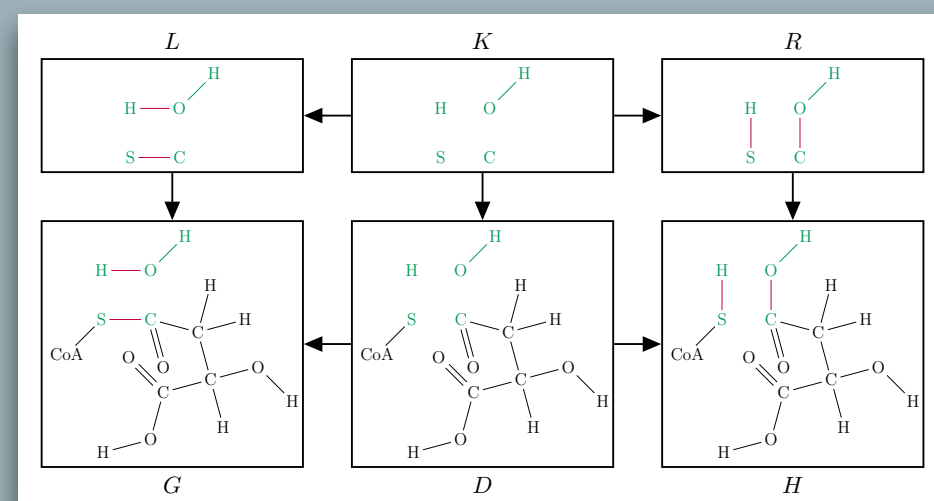
compositional rewriting double categories (crDCs)



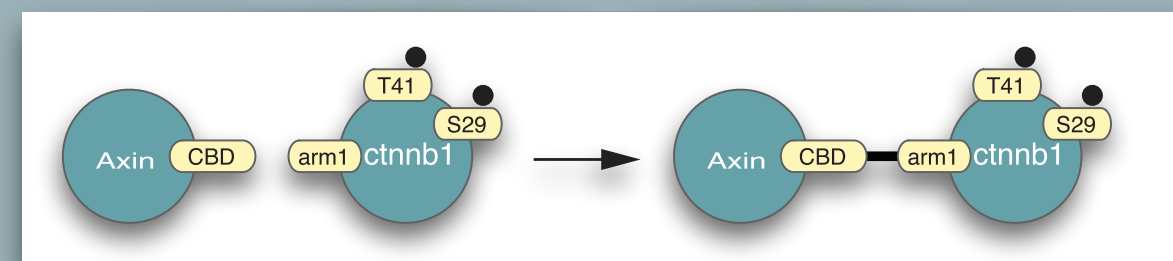
Explicit rewriting semantics (DPO, SqPO, ...)



organic chemistry



biochemistry



Instantiations of rewriting semantics in theory and applications

Fundamentals of Compositional Rewriting Theory*

Nicolas Behr^{a,*}, Russell Harmer^b, Jean Krivine^a

^aUniversité Paris Cité, CNRS, IRIF, 8 Place Aurélie Nemours, Paris Cedex 13, 75205, France

^bUniversité de Lyon, ENS de Lyon, UCBL, CNRS, LIP, 46 allée d'Italie, Lyon Cedex 07, 69364, France

nLab Home Page All Pages

species

Contents

- [1. Idea](#)
- [2. Definition](#)
 - [1-categorical](#)
 - [2-categorical](#)
 - [\(∞, 1\)-categorical](#)
 - [Operations on species](#)
 - [Sum](#)
 - [Cauchy product](#)
 - [Hadamard product](#)
 - [Dirichlet product](#)
 - [Composition product](#)
- [3. In Homotopy Type Theory](#)
 - [Operations on species](#)
 - [Coproduct](#)
 - [Hadamard product](#)

<https://ncatlab.org>

CoREACT

Coq-based Rewriting: towards Executable Applied Category Theory

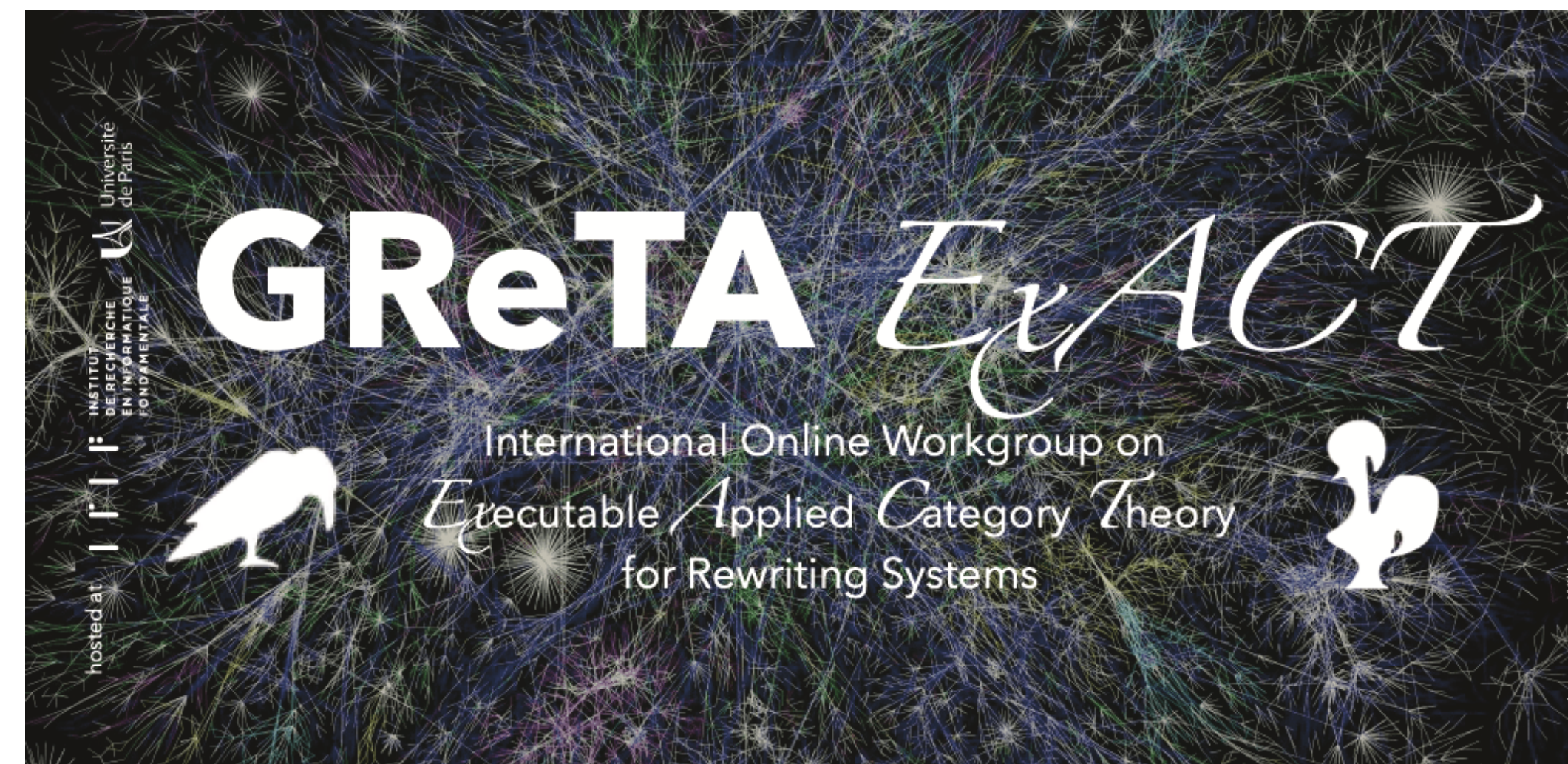
Consortium: IRIF (UP), LIP (ENS-Lyon), LIX (École Polytechnique), Sophia-Antipolis (Inria)

Partner	Last name	First name	Position
Université Paris Cité	BEHR	Nicolas	CNRS CR
	GALLEGO	Emilio	Inria SRP
	GHEERBRANT	Amélie	MdC
	HERBELIN	Hugo	Inria DR
	MELLIÈS	Paul-André	CNRS DR
	ROGOVA	Alexandra	PhD st.
	PhD student	(to recruit)	
ENS-Lyon	HARMER	Russell	CNRS CR
	HIRSCHOWITZ	Tom	CNRS DR
	POUS	Damien	CNRS DR
	PostDoc	(to recruit)	
École Polytechnique	MIMRAM	Samuel	PR
	WERNER	Benjamin	PR
	ZEILBERGER	Noam	MdC
	PostDoc	(to recruit)	
Inria Sophia-Antipolis	BERTOT	Yves	Inria DR
	COHEN	Cyril	Inria CR
	TASSI	Enrico	Inria CR
	PostDoc	(to recruit)	
Cambridge University	LAFONT	Ambroise	PostDoc

Project type / duration / start date **PRC / 48 months / March 2023**

Budget **€424335**

Scientific coordinator **Nicolas BEHR**



coreact.wiki

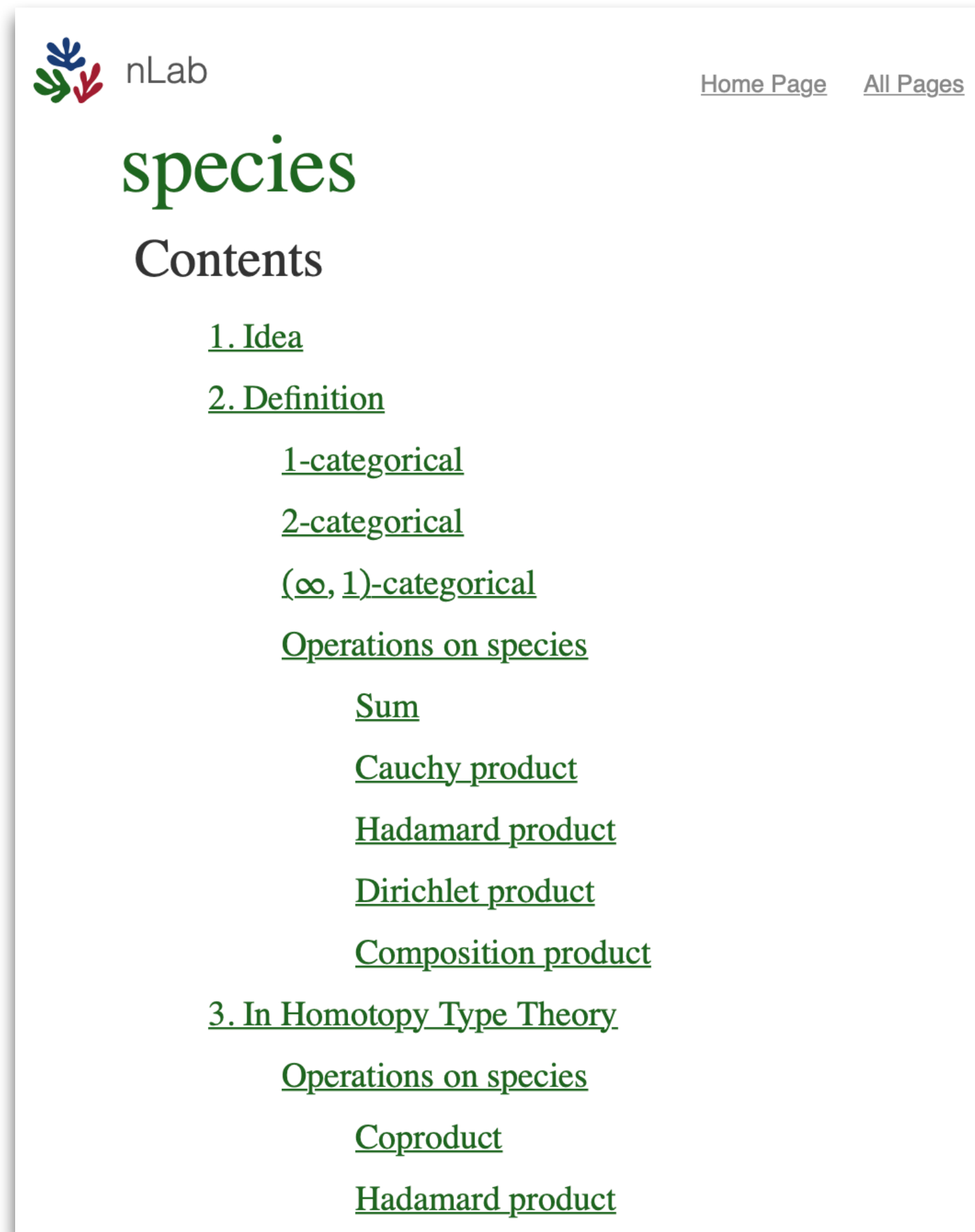
Main objectives of the CoREACT/GReTA ExACT initiative

- Development of a methodology for **diagrammatic reasoning in Coq**
- **Formalization** (in Coq) and **certification** of a representative collection of axioms and theorems for **compositional categorical rewriting theory**
- Development of a **Coq-enabled interactive database and wiki system**
- Development of a CoREACT wiki-based “**proof-by-pointing**” **engine**
- Executable **reference prototype algorithms** from categorical structures in Coq (via the use of SMT solvers/theorem provers such as Z3)

Main objectives of the CoREACT/GReTA ExACT initiative

- Development of a methodology for **diagrammatic reasoning in Coq**
- **Formalization** (in Coq) and **certification** of a representative collection of axioms and theorems for **compositional categorical rewriting theory**
- Development of a **Coq-enabled interactive database and wiki system**
- Development of a CoREACT wiki-based “**proof-by-pointing**” **engine**
- Executable **reference prototype algorithms** from categorical structures in Coq (via the use of SMT solvers/theorem provers such as Z3)

A (very non-exhaustive!) view on **wiki systems in mathematics/ (A)CT**



nLab

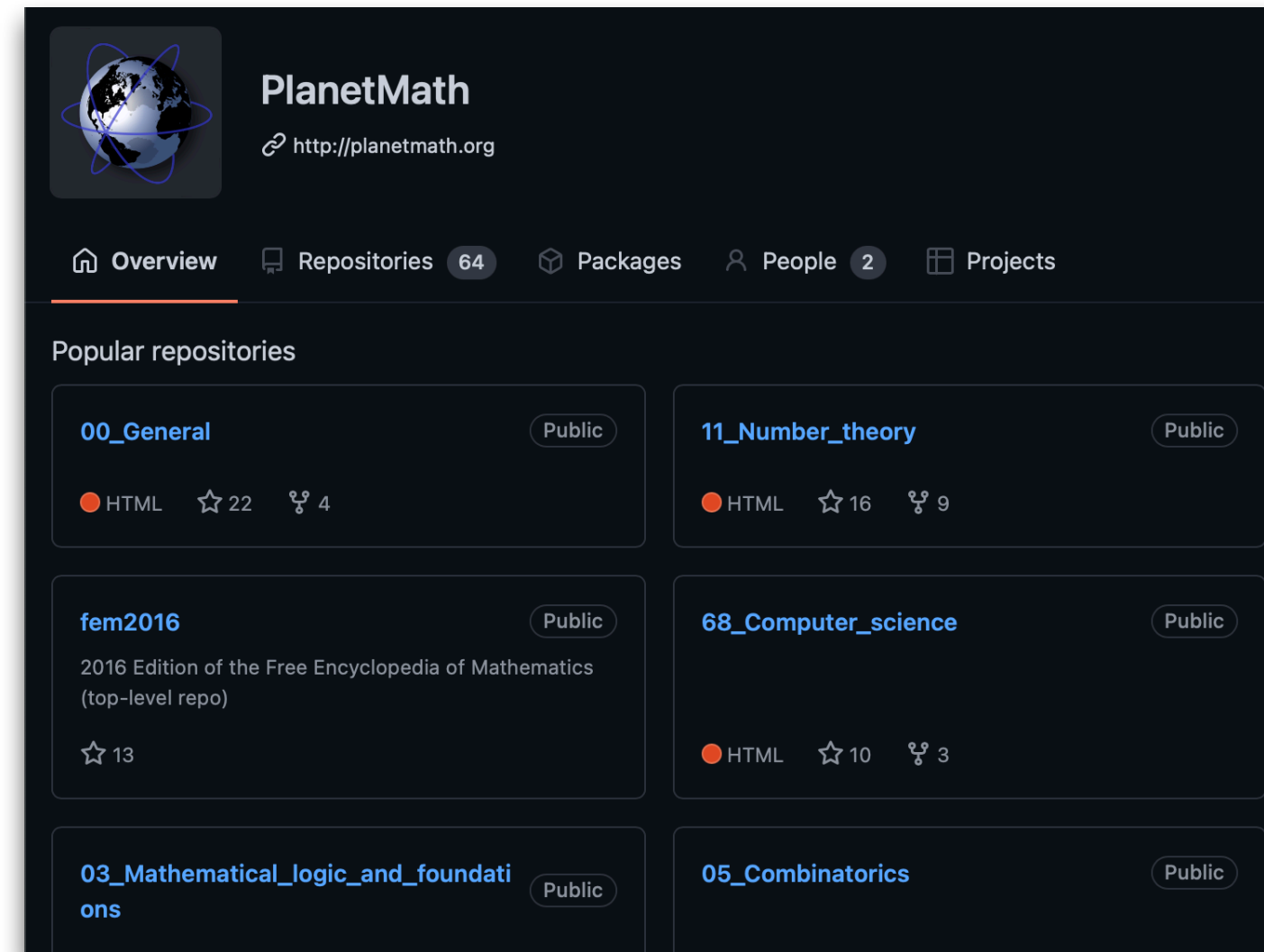
Home Page All Pages

species

Contents

- [1. Idea](#)
- [2. Definition](#)
 - [1-categorical](#)
 - [2-categorical](#)
 - [\$\(\infty, 1\)\$ -categorical](#)
 - [Operations on species](#)
 - [Sum](#)
 - [Cauchy product](#)
 - [Hadamard product](#)
 - [Dirichlet product](#)
 - [Composition product](#)
- [3. In Homotopy Type Theory](#)
 - [Operations on species](#)
 - [Coproduct](#)
 - [Hadamard product](#)

<https://ncatlab.org>



PlanetMath

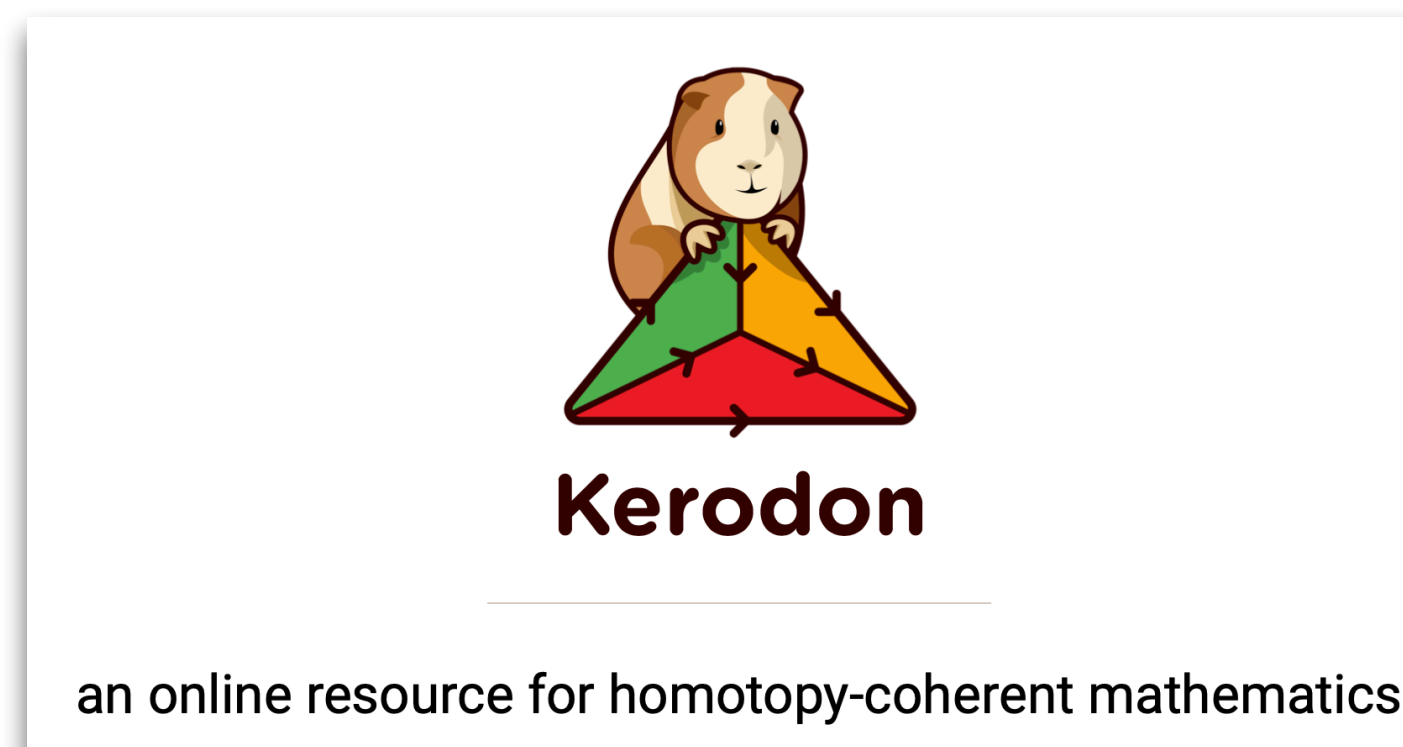
<http://planetmath.org>

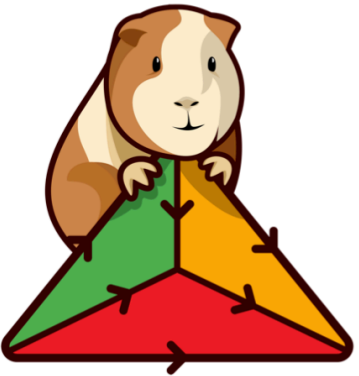
Overview Repositories 64 Packages People 2 Projects

Popular repositories

- [00_General](#) Public HTML ☆ 22 🗨 4
- [11_Number_theory](#) Public HTML ☆ 16 🗨 9
- [fem2016](#) Public 2016 Edition of the Free Encyclopedia of Mathematics (top-level repo) ☆ 13
- [68_Computer_science](#) Public HTML ☆ 10 🗨 3
- [03_Mathematical_logic_and_foundations](#) Public
- [05_Combinatorics](#) Public

<https://planetmath.org>

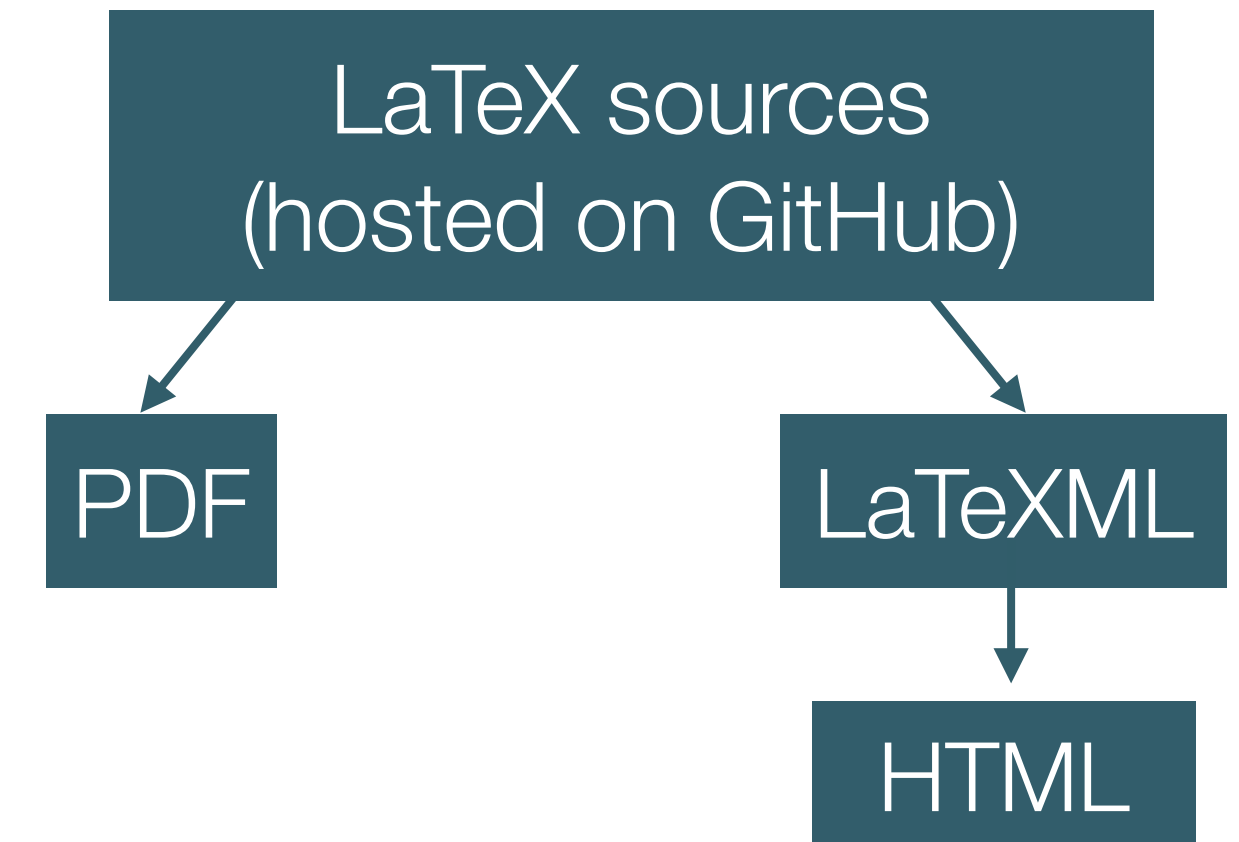




Kerodon

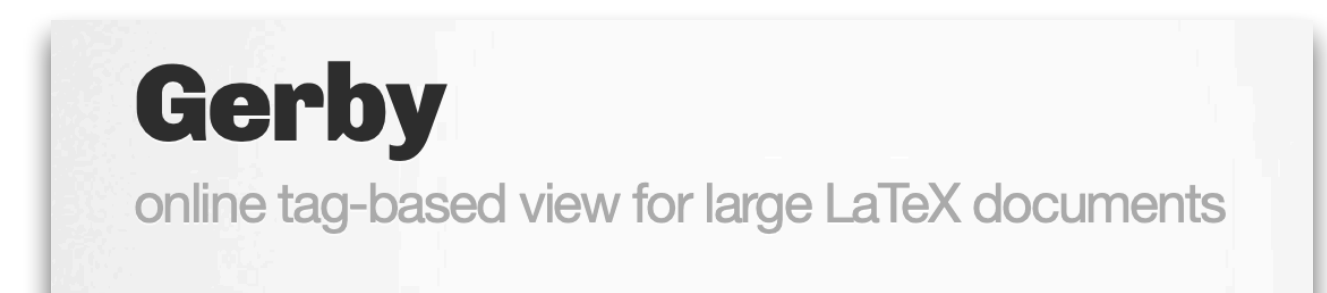
an online resource for homotopy-coherent mathematics

<https://kerodon.net>



(semi-) automatic cross-linking
provided via NNexus system

- J. Lurie's **online textbook** on categorical homotopy theory
- technology based upon **online tags view** via the Gerby system

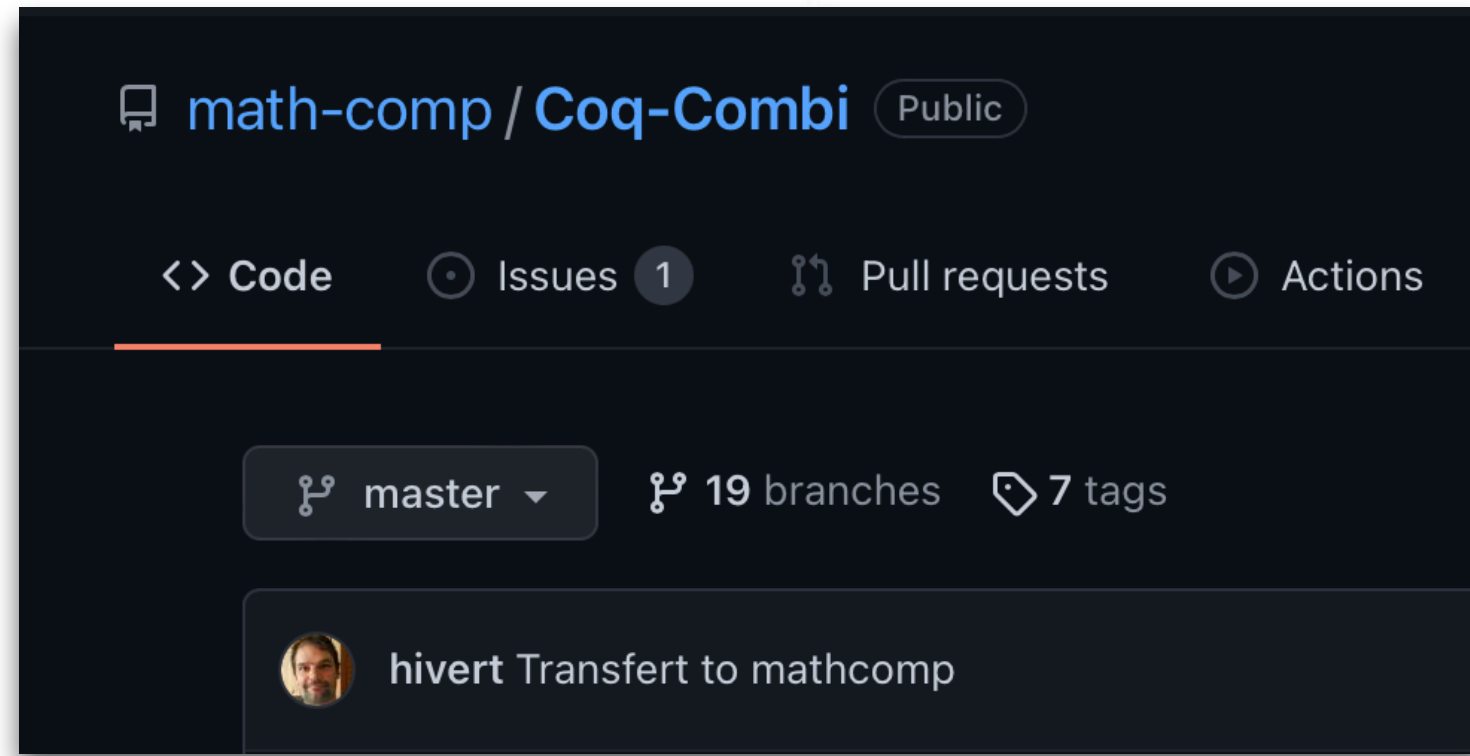


Gerby
online tag-based view for large LaTeX documents

<https://gerby-project.github.io>

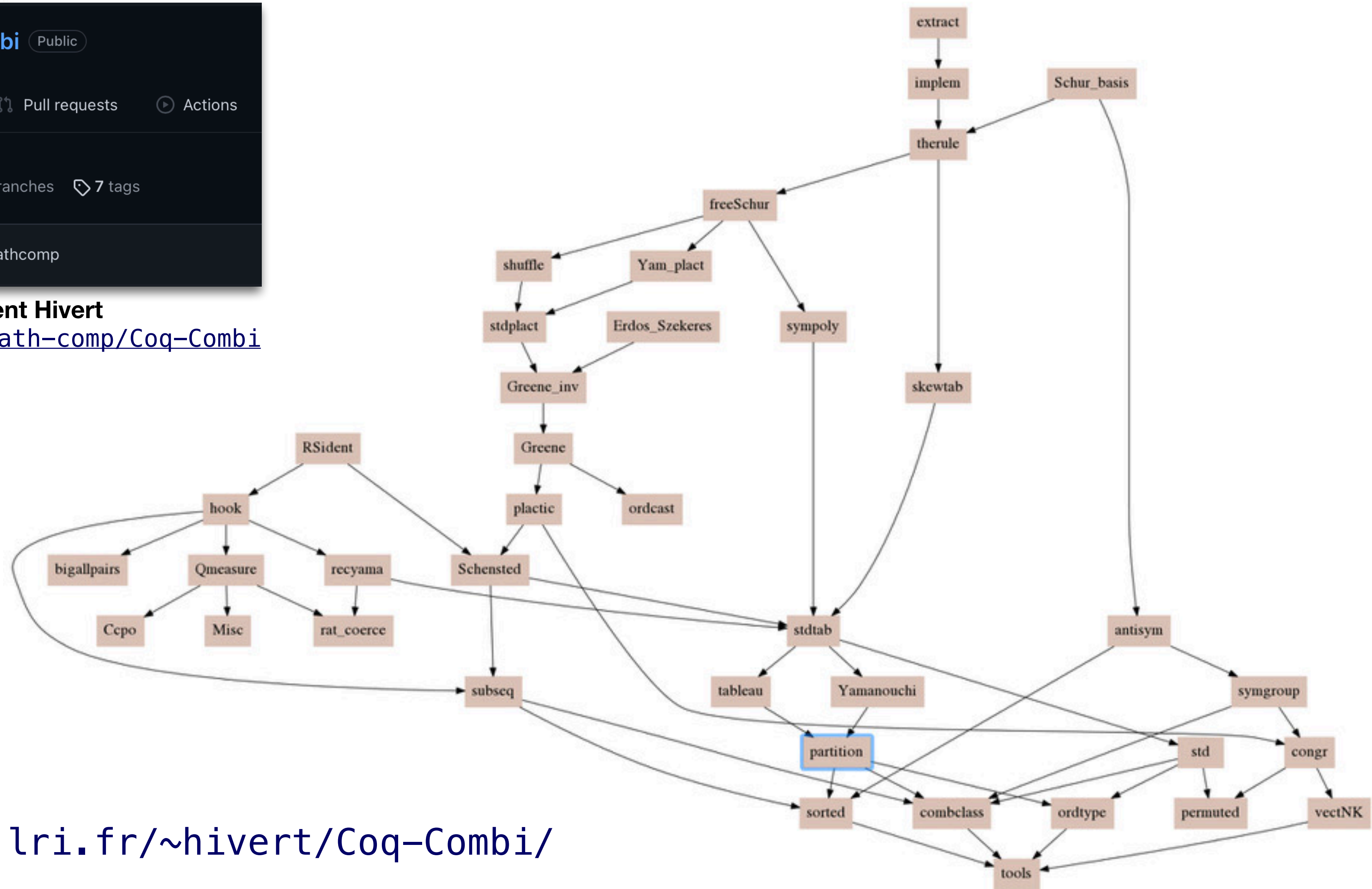
Coq-combi

Algebraic Combinatorics in Coq/SSReflect Documentation



Author: **Florent Hivert**

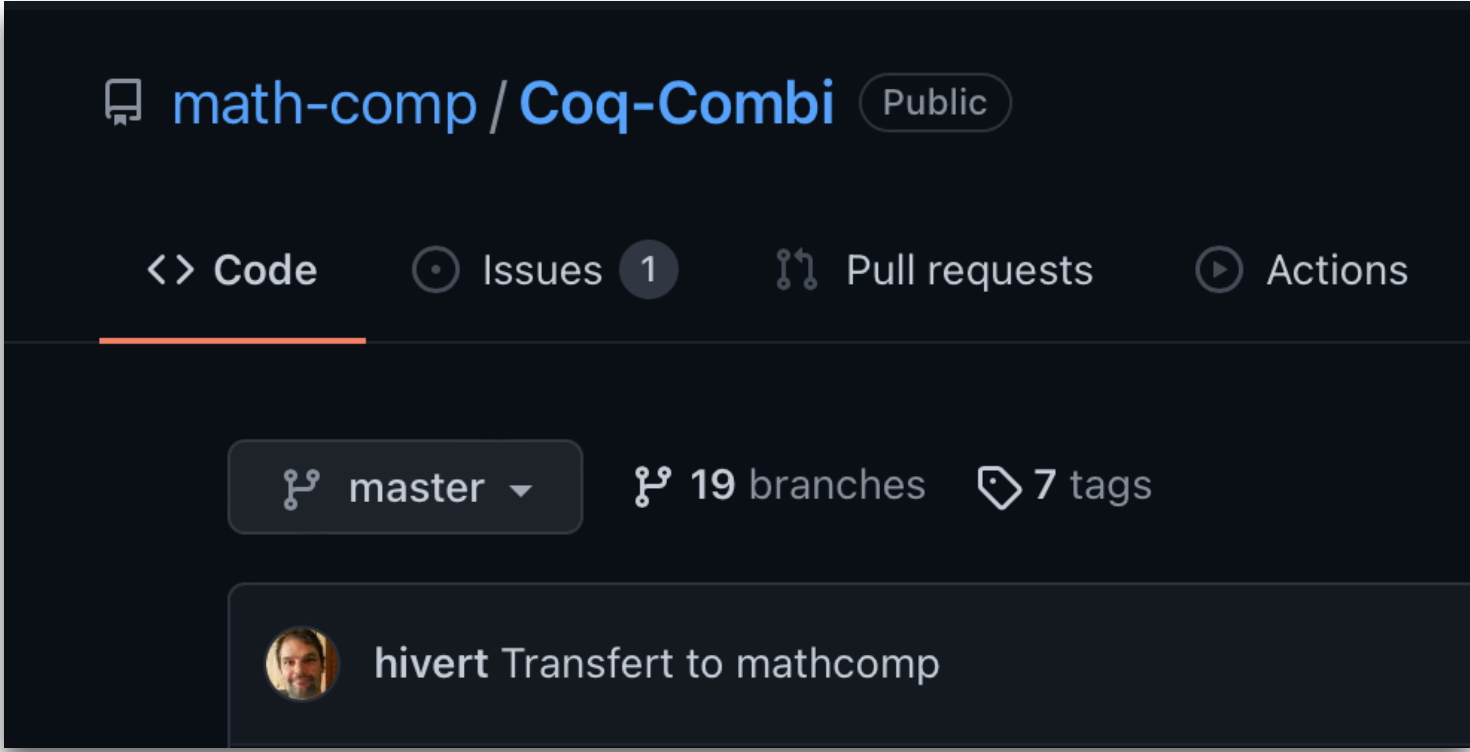
<https://github.com/math-comp/Coq-Combi>



<https://www.lri.fr/~hivert/Coq-Combi/>

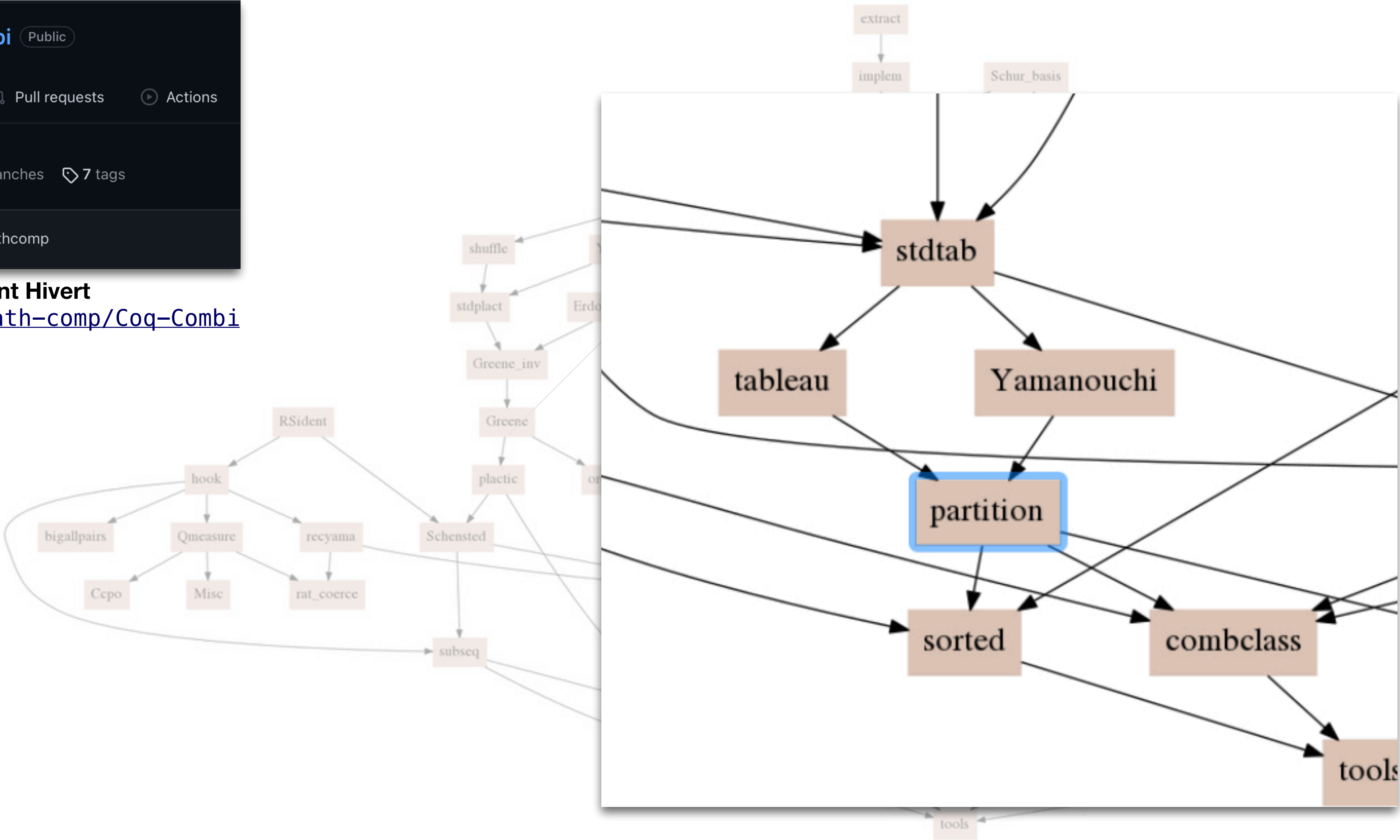
Coq-combi

Algebraic Combinatorics in Coq/SSReflect Documentation



Author: **Florent Hivert**

<https://github.com/math-comp/Coq-Combi>



Coq-combi

Combi.Combi.partition: Integer Partitions

- Shapes and Integer Partitions
- Shapes
 - A finite type `finType` for coordinate of boxes inside a shape
 - Rewriting bigops running along the boxes of a shape
 - Adding a box to a shape
- Integer Partitions
 - Definitions and basic properties
 - Corners, adding and removing corners
 - Conjugate of a partition
 - Partial sum of partitions
- Inclusion of Partitions and Skew Partitions
- Sigma Types for Partitions
- Counting functions
- TODO: Generalize and move in `finOrdType`

Coq-combi

Shapes and Integer Partitions

Partitions (and more generally shapes) are stored by terms of type `seq (seq nat)`. We define the following predicates and operations on `seq (seq nat)`: (r, c) is in `sh` if $r < sh[i]$

- `is_in_shape sh r c` == the box with coordinate (r, c) belongs to the shape `sh`, that is: $c < sh[r]$.
- `is_box_in_shape (r, c)` == uncurried version: same as `is_in_shape sh r c`.
- `box_in sh` == a sigma type for boxes in `sh` : $\{ b \mid is_box_in_shape\ sh\ b \}$ is canonically a `subFinType`.
- `enum_box_in sh` == a full duplicate free list of the boxes in `sh`.

Integer Partitions:

- `is_part sh` == `sh` is a partition
- `rem_trail0 sh` == remove the trailing zeroes of a shape
- `is_add_corner sh i` == i is the row of an addable corner of `sh`
- `is_rem_corner sh i` == i is the row of a removable corner of `sh`
- `incr_nth sh i` == the shape obtained by adding a box at the end of the i -th row. This gives a partition if i is an addable corner of `sh` (Lemma `is_part_incr_nth`)
- `decr_nth sh i` == the shape obtained by removing a box at the end of the i -th row. This gives a partition if i is an removable corner of `sh`

Sigma Types for Partitions

Section `PartCombClass`.

```
Structure intpart : Type := IntPart {pval :=> seq nat; _ : is_part pval}.
Canonical intpart_subType := Eval hnf in [subType for pval].
Definition intpart_eqMixin := Eval hnf in [eqMixin of intpart by <:].
Canonical intpart_eqType := Eval hnf in EqType intpart intpart_eqMixin.
Definition intpart_choiceMixin := Eval hnf in [choiceMixin of intpart by <:].
Canonical intpart_choiceType := Eval hnf in ChoiceType intpart intpart_choiceMixin.
Definition intpart_countMixin := Eval hnf in [countMixin of intpart by <:].
Canonical intpart_countType := Eval hnf in CountType intpart intpart_countMixin.

Lemma intpartP (p : intpart) : is_part p.

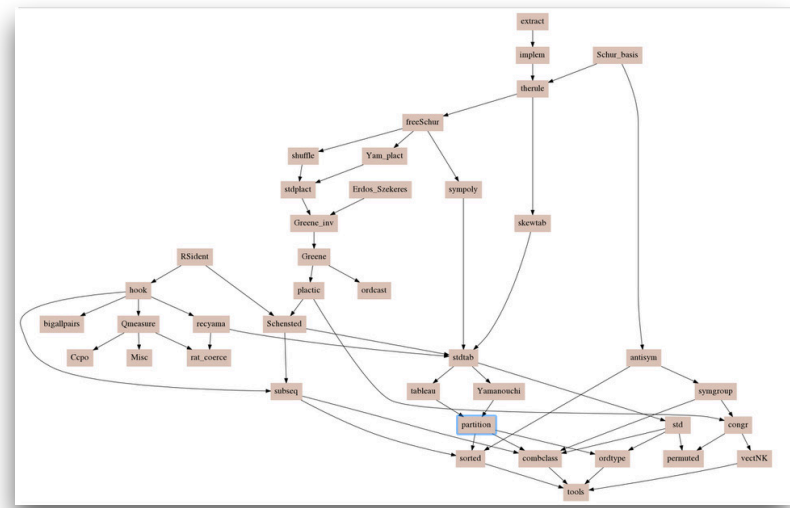
Hint Resolve intpartP.

Canonical conj_intpart p := IntPart (is_part_conj (intpartP p)).

Lemma conj_intpartK : involutive conj_intpart.

Lemma intpart_sum_inj (s t : intpart) :
  (∀ k, part_sum s k = part_sum t k) → s = t.

Fixpoint enum_partnsk sm sz mx : (seq (seq nat)) :=
  if sz is sz.+1 then
    flatten [seq [seq i :: p | p <- enum_partnsk (sm - i) sz i] | i <- iota 1 (minn sm mx)]
  else if sm is sm.+1 then [::] else [:: [::]].
Definition enum_partns sm sz := enum_partnsk sm sz sm.
Definition enum_partn sm := flatten [seq enum_partns sm sz | sz <- iota 0 sm.+1 ].
```



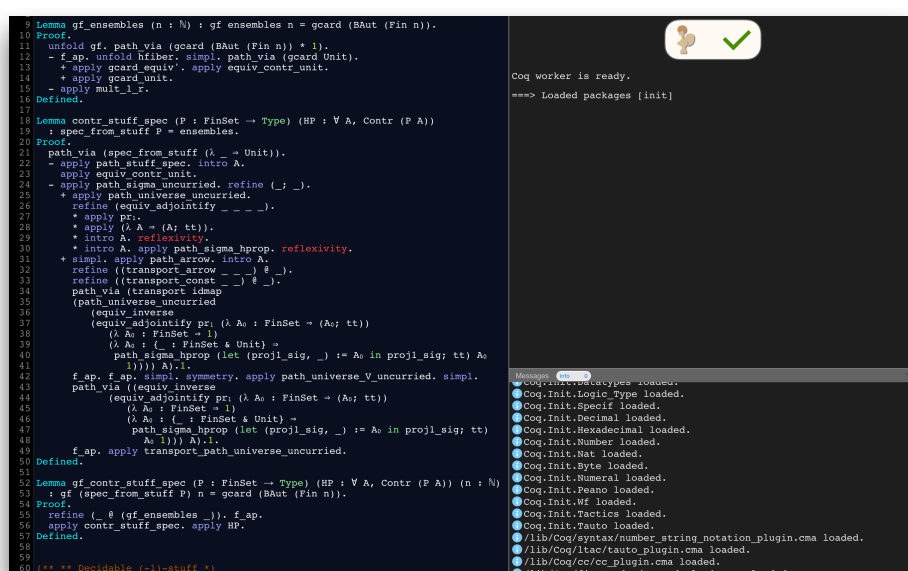
knowledge graphs



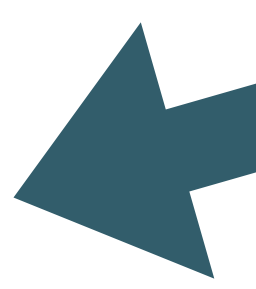
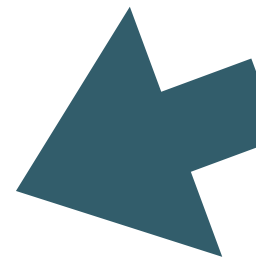
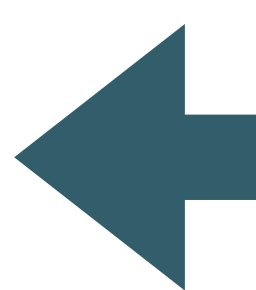
PDF textbook



CoREACT Coq library



jsCoq interactive web interface



wiki entry (e.g., a Lemma)

#hash (auto-generated)
Tags
list of cross-references
bibliographic references
code origin references

Human-readable text

LaTeX-based (e.g. via sTeX + Gerby), with annotations permitting generation of cross-references via NNexus

Machine-readable Coq-formalisation

Including compatible Coq version and possibly different variants for (1) different Coq versions and/or (2) different implementation strategies/frameworks/theories.

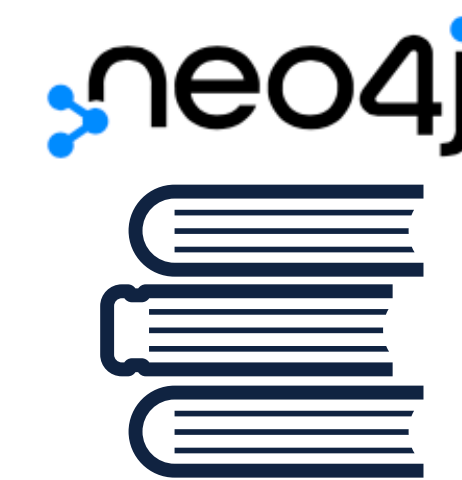
Examples (both maths & Coq)

Curated in jsCoq, directly executable from within the wiki entry in the form of a literate web document and/or as a bundle of a Coq file with instructions for a particular Docker image for Coq.

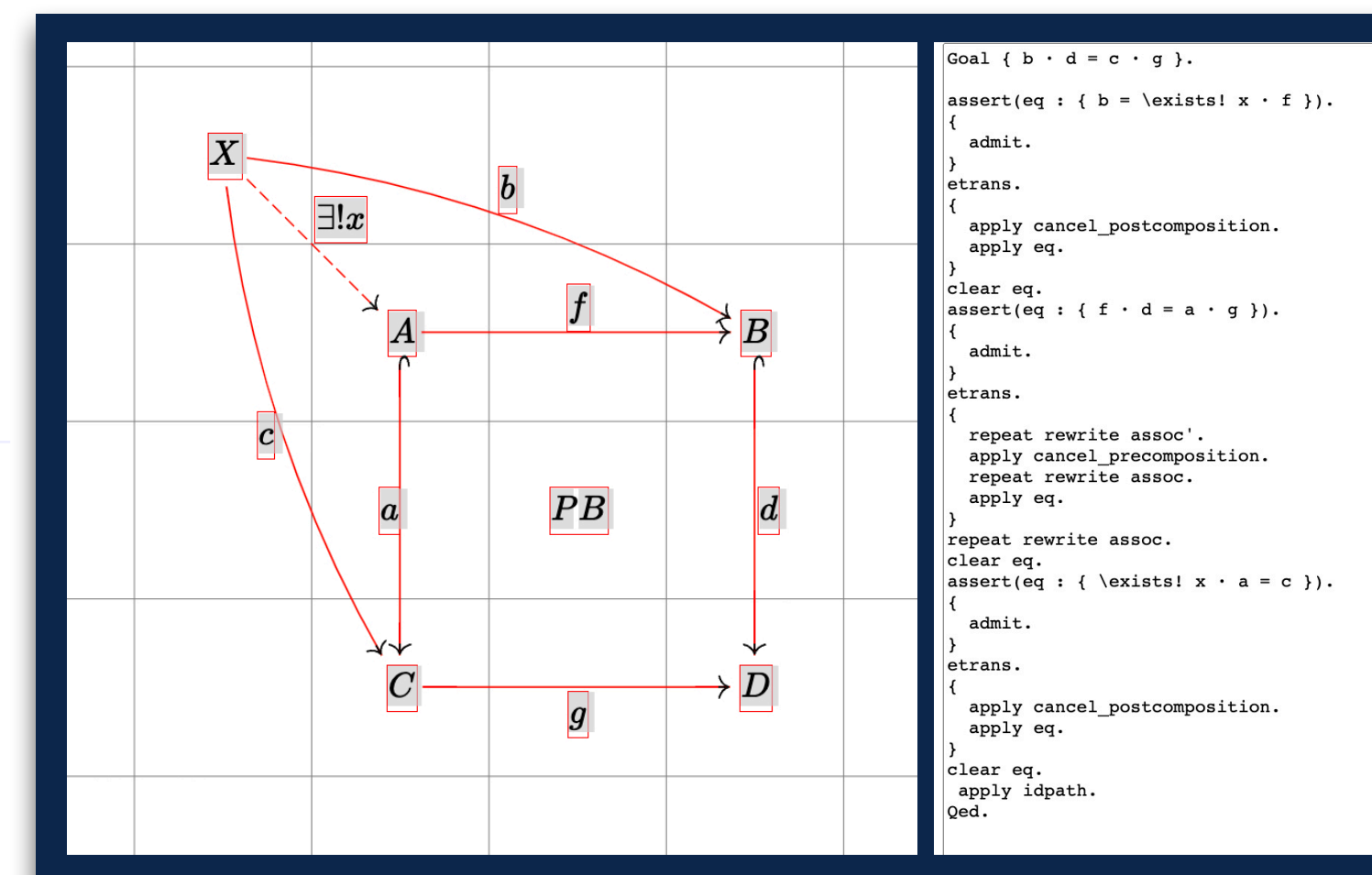
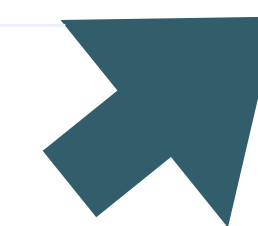
Proof tactics and performance data

Machine-learned tactics data, cross-evaluation of performance of different variants of implementations, user annotations on different Coq versions/libraries used

coreact.wiki



(graph-) database



coreact.workbench

Welcome to the jsCoq Interactive Online System!

Welcome to the jsCoq technology demo! jsCoq is an interactive, web-based environment for the Coq Theorem prover, and is a collaborative development effort. See the [list of contributors](#) below.

jsCoq is open source. If you find any problem or want to make any contribution, you are extremely welcome! We await your feedback at [GitHub](#) and [Zulip](#).

Instructions:

The following document contains embedded Coq code. All the code is editable and can be run directly on the page. Once jsCoq finishes loading, you are free to experiment by stepping through the proof and viewing intermediate proof states on the right panel.

Actions:

Button	Key binding	Action
	Alt + ↑ / ↓ or Alt + N / P	Move through the proof.
	Alt + Enter or Alt + -	Run (or go back) to the current point.
	F8	Toggles the goal panel.

Creating your own proof scripts:

The *scratchpad* offers simple, local storage functionality. It also allows you to share your development with other users in a manner that is similar to Pastebin.

A First Example: The Infinitude of Primes

If you are new to Coq, check out this [introductory tutorial](#) by Mike Nahas. As a more advanced showcase, we display a proof of the infinitude of primes in Coq. The proof relies on the [Mathematical Components](#) library from the MSR/Inria team led by Georges Gonthier, so our first step will be to load it:

```
1 From Coq Require Import ssreflect ssrfun ssrbool.
2 From mathcomp Require Import eqtype ssrnat div prime.
```

Ready to do Proofs!

Once the basic environment has been set up, we can proceed to the proof:

```
3 (* A nice proof of the infinitude of primes, by Georges Gonthier *)
4 Lemma prime_above m : {p | m < p & prime p}.
5 Proof.
```

Goals

```
jsCoq (0.13.3), Coq 8.13.2/81300 (September 2021),
compiled on Sep 21 2021 15:32:50
OCaml 4.12.0, Js_of_ocaml 3.9.0
```

Coq worker is ready.
====> Loaded packages [init]

Messages

```
Coq.Init.Datatypes loaded.
Coq.Init.Logic_Type loaded.
Coq.Init.Specif loaded.
Coq.Init.Decimal loaded.
Coq.Init.Hexadecimal loaded.
Coq.Init.Number loaded.
Coq.Init.Nat loaded.
Coq.Init.Byte loaded.
Coq.Init.Numeral loaded.
Coq.Init.Peano loaded.
```

<https://github.com/jscoq>

```
9 Lemma gf_ensembles (n : N) : gf ensembles n = gcard (BAut (Fin n)).
10 Proof.
11   unfold gf. path_via (gcard (BAut (Fin n)) * 1).
12   - f_ap. unfold hfiber. simpl. path_via (gcard Unit).
13     + apply gcard_equiv'. apply equiv_contr_unit.
14     + apply gcard_unit.
15   - apply mult_1_r.
16 Defined.
17
18 Lemma contr_stuff_spec (P : FinSet → Type) (HP : ∀ A, Contr (P A))
19   : spec_from_stuff P = ensembles.
20 Proof.
21   path_via (spec_from_stuff (λ _ => Unit)).
22   - apply path_stuff_spec. intro A.
23     apply equiv_contr_unit.
24   - apply path_sigma_uncurried. refine (λ _ => _).
25     + apply path_universe_uncurried.
26       refine (equiv_adjointify _ _ _).
27       * apply pr1.
28       * apply (λ A => (A; tt)).
29       * intro A. reflexivity.
30     * intro A. apply path_sigma_hprop. reflexivity.
31   + simpl. apply path_arrow. intro A.
32     refine ((transport_arrow _ _ _) @ _).
33     refine ((transport_const _ _ _) @ _).
34     path_via (transport idmap
35       (path_universe_uncurried
36         (equiv_inverse
37           (equiv_adjointify pr1 (λ A0 : FinSet => (A0; tt))
38             (λ A0 : FinSet => 1)
39             (λ A0 : { _ : FinSet & Unit} =>
40               path_sigma_hprop (let (proj1_sig, _) := A0 in proj1_sig; tt) A0
41               1))) A).1.
42     f_ap. f_ap. simpl. symmetry. apply path_universe_V_uncurried. simpl.
43     path_via ((equiv_inverse
44       (equiv_adjointify pr1 (λ A0 : FinSet => (A0; tt))
45         (λ A0 : FinSet => 1)
46         (λ A0 : { _ : FinSet & Unit} =>
47           path_sigma_hprop (let (proj1_sig, _) := A0 in proj1_sig; tt)
48           A0 1))) A).1.
49     f_ap. apply transport_path_universe_uncurried.
50 Defined.
51
52 Lemma gf_contr_stuff_spec (P : FinSet → Type) (HP : ∀ A, Contr (P A)) (n : N)
53   : gf (spec_from_stuff P) n = gcard (BAut (Fin n)).
54 Proof.
55   refine (λ _ @ (gf_ensembles _)). f_ap.
56   apply contr_stuff_spec. apply HP.
57 Defined.
58
59 (** ** Decidable (-1)-stuff **
60
```

Coq worker is ready.
====> Loaded packages [init]

Messages

```
Coq.Init.Datatypes loaded.
Coq.Init.Logic_Type loaded.
Coq.Init.Specif loaded.
Coq.Init.Decimal loaded.
Coq.Init.Hexadecimal loaded.
Coq.Init.Number loaded.
Coq.Init.Nat loaded.
Coq.Init.Byte loaded.
Coq.Init.Numeral loaded.
Coq.Init.Peano loaded.
Coq.Init.Wf loaded.
Coq.Init.Tactics loaded.
Coq.Init.Tauto loaded.
/lib/Coq/syntax/number_string_notation_plugin.cma loaded.
/lib/Coq/ltac/tauto_plugin.cma loaded.
/lib/Coq/cc/cc_plugin.cma loaded.
/lib/Coq/finstorder/ground_plugin.cma loaded.
```

Core developer team

- Emilio Jesús Gallego Arias , Inria, Université de Paris, IRIF
- Shachar Itzhaky , Technion

Past Contributors

- Benoît Pin, CRI, MINES ParisTech

Coq-community

The screenshot shows the GitHub profile for 'coq-community'. At the top, there's a navigation bar with links for 'Why GitHub?', 'Team', 'Enterprise', 'Explore', 'Marketplace', 'Pricing', 'Search', 'Sign in', and 'Sign up'. The profile header includes the repository name 'coq-community', a description: 'A project for a collaborative, community-driven effort for the long-term maintenance and advertisement of Coq packages.', and the URL 'https://coq-community.org'. Below this, there are tabs for 'Overview', 'Repositories' (58), 'Packages', 'People' (14), and 'Projects'. The main content area is divided into 'Pinned' repositories and a 'People' section. The pinned repositories include 'manifesto', 'hydra-battles', 'awesome-coq', 'vscoq', 'docker-coq', and 'templates'. The 'People' section shows a grid of user avatars. To the right, there are sections for 'Top languages' (Coq, Shell, OCaml, Dockerfile, JavaScript) and 'Most used topics' (coq, docker-coq-action, mathcomp, nix-action, coq-library).

<https://github.com/coq-community>

“A project for a **collaborative, community-driven** effort for the **long-term maintenance** and **advertisement** of Coq packages.”

- **58 repositories**

The screenshot shows the GitHub repository page for 'Mathematical Components'. The navigation bar includes 'Overview', 'Repositories', 'Packages', 'People', and 'Projects'. A search bar and filters for 'Type', 'Language', and 'Sort' are visible. The repository is public and has a description: 'A proof of Abel-Ruffini theorem.' It features tags for 'coq', 'ssreflect', 'galois-theory', 'mathcomp', and 'abel-ruffini'. Below the description, there are statistics: 3 forks, 26 stars, 0 issues, and 0 pull requests, updated 25 days ago. The repository is followed by several other repositories: 'algebra-tactics' (Ring and field tactics for Mathematical Components), 'analysis' (Mathematical Components compliant Analysis Library), 'apery' (A formal proof of the irrationality of zeta(3), the Apéry constant), 'bigenough' (Asymptotic reasoning with bigenough), 'cad' (Formalizing Cylindrical Algebraic Decomposition related theories in mathcomp), and 'Coq-Combi' (Algebraic Combinatorics in Coq). Each repository entry includes its description, tags, and statistics.

Category theory

- **Special types of categories:**
 - adhesive/quasi-adhesive/adhesive HLR/weak adhesive HLR/...
 - quasi-topoi
- **Double categories**
- **Universal constructions:**
 - stable systems of monics, factorisation systems, multi-sums, ...
 - pushouts, pullbacks, final pullback complements, multi-initial pushout complements, final pullback complement augmentations, ...
 - Grothendieck fibrations/multi-opfibrations/residual multi-opfibrations ...
- **Lemmata on special properties of universal constructions:**
 - (De-)composition properties
 - fibrational properties
 - Beck-Chevalley conditions

Diagrammatic reasoning

- **Commutative diagrams**
- **Reasoning moves**
 - from universal properties
 - from diagrammatic lemmata
- **Compositionality of reasoning moves**

Formalisations for `coreact`.workbench

- **Auxiliary tactics** to convert between **drawings** and **Coq expressions**
- From **drawing transformations** to **reasoning moves**
- From **drawing transformations** to **Cypher queries**

Foundations of compositional rewriting theory

- **Compositional rewriting double categories (crDCs)**
- **Concurrency Theorems**
- **Associativity Theorems**
- **Rule Algebra and Stochastic Mechanics**
- **Tracelet Hopf Algebras and Decomposition Spaces**

Collection of rewriting semantics

- **Double Pushout/Sesqui-Pushout/Single-Pushout/AGREE/PBPO+/...**
 - linear/input-linear/output-linear/non-linear/...
- **Theory of constraints and application conditions:**
 - nested application conditions
 - constraint-guaranteeing/-preserving semantics
- **Compositional rewriting for rules with conditions**
 - shift and transport constructions
 - Concurrency and associativity theorems
 - Rule algebras/stochastic mechanics/tracelets/...

Executable Applied Category theory (ExACT)

- **Constructive characterization of categories with adhesivity/quasi-topoi:**
 - Artin gluing/slice/coslice/product/sum/**functor and comma categories**/...
 - **collection** of practically relevant examples (**Graph** as **presheaf topos**, **SimpleGraph** via **Artin gluing**, **HyperGraph** as **comma category**, ...)
- **Translation** from **rewriting semantics** to **SMT solvers/theorem provers**
- **Reference prototype algorithms for concrete rewriting semantics**

GRETA - Graph Transformation Theory and Applications

Université de Paris
INSTITUT DE RECHERCHE EN INFORMATIQUE FONDAMENTALE

GRETA *ExACT*

International Online Workgroup on
Executable Applied Category Theory
for Rewriting Systems

hosted at

The central aims of this workgroup consist in providing an interdisciplinary forum for exploring the diverse aspects of applied category theory relevant in graph transformation systems and their generalizations, in developing a methodology for formalizing diagrammatic proofs as relevant in rewriting theories via proof assistants such as Coq, and in establishing a community-driven wiki system and repository for mathematical knowledge in our research field (akin to a domain-specific Coq-enabled variant of the nLab). A further research question will explore the possibility of deriving reference prototype implementations of concrete rewriting systems (e.g., over multi- or simple directed graphs) directly from the category-theoretical semantics, in the spirit of the translation-based approaches (utilizing theorem provers such as Microsoft Z3).

- To receive regular updates on the GRETA ExACT workgroup sessions, please consider subscribing to our [mailing list](#).
- To suggest speakers and topics for upcoming sessions, and for any other form of feedback and discussions, please consider joining the GRETA ExACT [Mattermost channel](#).

YouTube FR

Search

SIGN IN

Home

Explore

Subscriptions

Library

History

Friday November 12, 2021, 10:00 CET

SMTCoq: the power of SMT solving in Coq

Chantal Keller

PLAY ALL

GRETA-ExACT (Executable Applied Category Theory) online workgroup

4 videos • No views • Last updated on 8 Nov 2021

GRETA Seminar

SUBSCRIBE

1 GRETA-ExACT session #1: "SMTCoq: the power of SMT solving in Coq" GRETA Seminar 55:36

2 GRETA-ExACT session #2: "Hierarchy Builder" GRETA Seminar 1:05:41

3 GRETA-ExACT session #3: "Formalizing Category Theory using Type Theory: A Discussion" GRETA Seminar • Scheduled for 10/12/2021, 15:00 LIVE

4 GRETA seminar #20: "GRETA-ExACT: towards Executable Applied Category Theory" GRETA Seminar 1:46:40

coreact.wiki

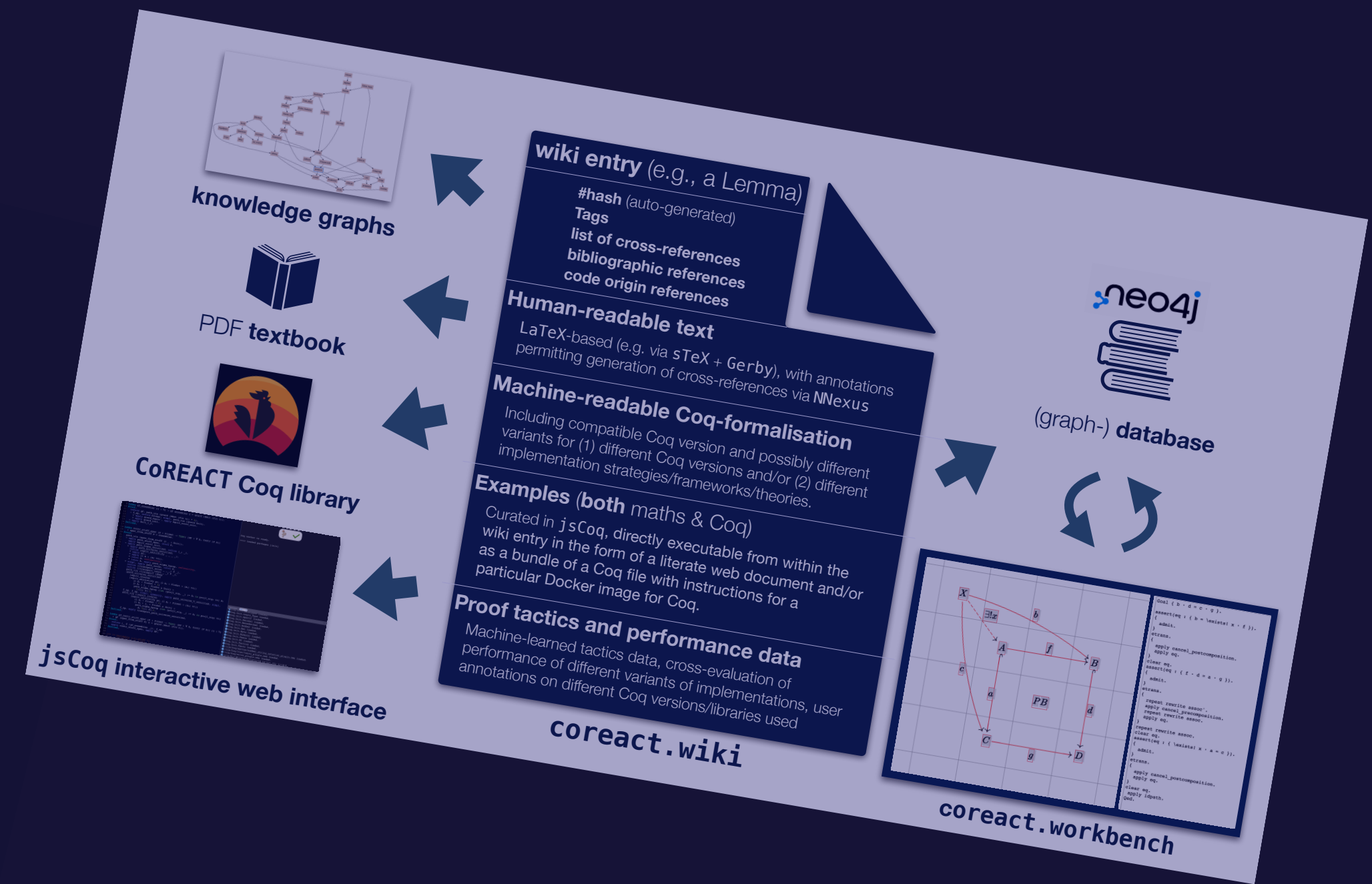
CoREACT

Coq-based Rewriting: towards Executable Applied Category Theory

Consortium: IRIF (UP), LIP (ENS-Lyon), LIX (École Polytechnique), Sophia-Antipolis (Inria)



```
83
84
85 (** * Species Sum/Coproduct *)
86
87 Definition spec_sum (X Y : Species) : Species
88   := ((X.1 + Y.1)%type; sum_rect _ X.2 Y.2).
89
90 Lemma sigma_functor_sum (X : Type) (P Q : X -> Type) :
91   ({x : X & P x} + {x : X & Q x}) <~> {x : X & (P x + Q x)%type}.
92 Proof.
93   refine (equiv_adjointify _ _ _).
94   - intros [x w] | [x w]; exists x; [left | right]; apply w.
95   - intros [x [w | w]]; [left | right]; apply (x; w).
96   - intros [x [w | w]]; reflexivity.
97   - intros [[x w] | [x w]]; reflexivity.
98 Defined.
99
100 Definition stuff_spec_sum (P Q : FinSet -> Type) := fun A => (P A + Q A)%type.
101
102 Lemma stuff_spec_sum_correct (P Q : FinSet -> Type) :
103   spec_from_stuff (stuff_spec_sum P Q)
104   =
```



Mer ci beaucoup !

Appendix: some examples of **diagrammatic proofs** in **compositional rewriting theory** (cf. YouTube video for further details)



Fundamentals of Compositional Rewriting Theory

Nicolas Behr

Topos Institute Colloquium

Appendix: some examples of **diagrammatic proofs** in **compositional rewriting theory** (cf. YouTube video for further details)



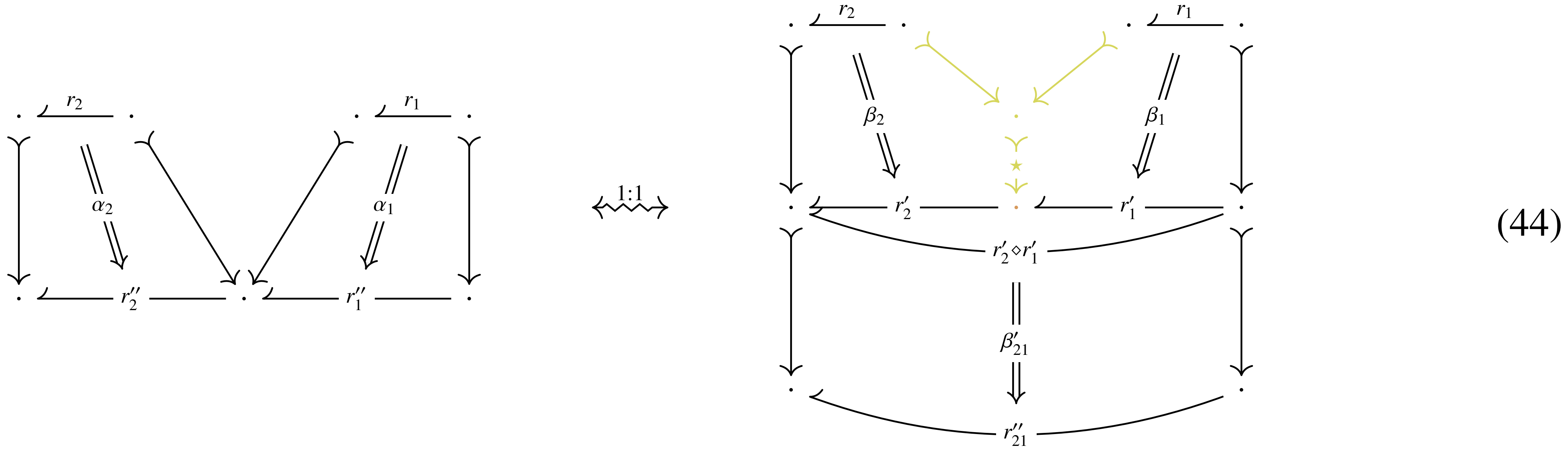
Fundamentals of Compositional Rewriting Theory

Nicolas Behr

Topos Institute Colloquium

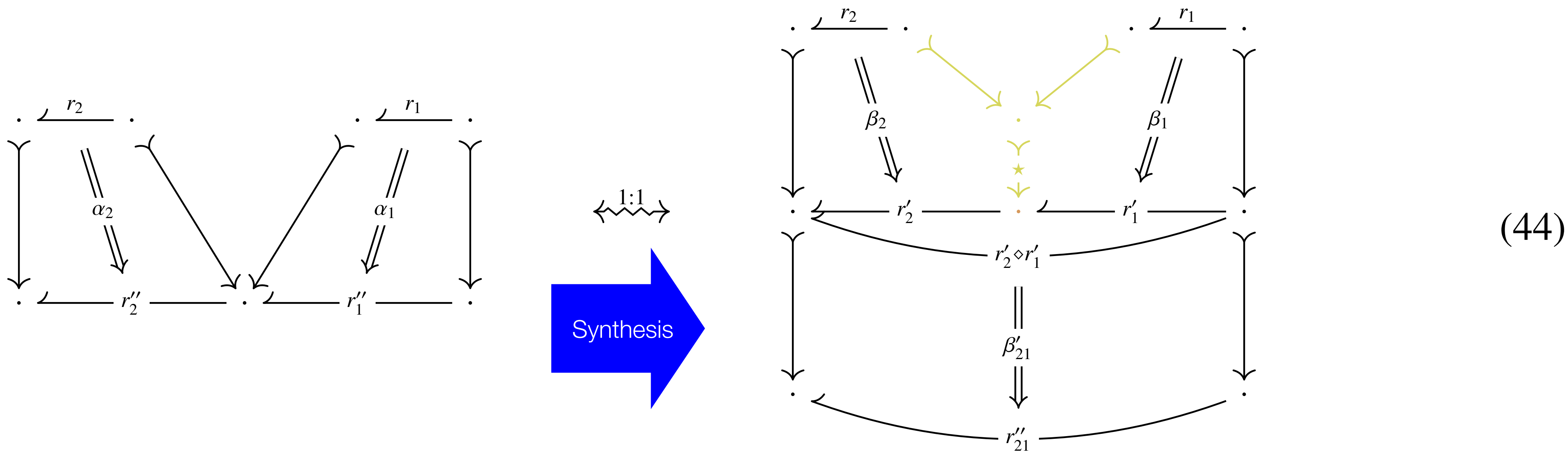
crDCs satisfy a (*universal!*) **Concurrency Theorem**

Theorem 8. *Let \mathbb{D} be a **compositional rewriting double category**. Then the following statements hold (where the morphism marked \star in the diagram on the right is a **residue**, and the cospan into its domain a **multi-sum element**):*



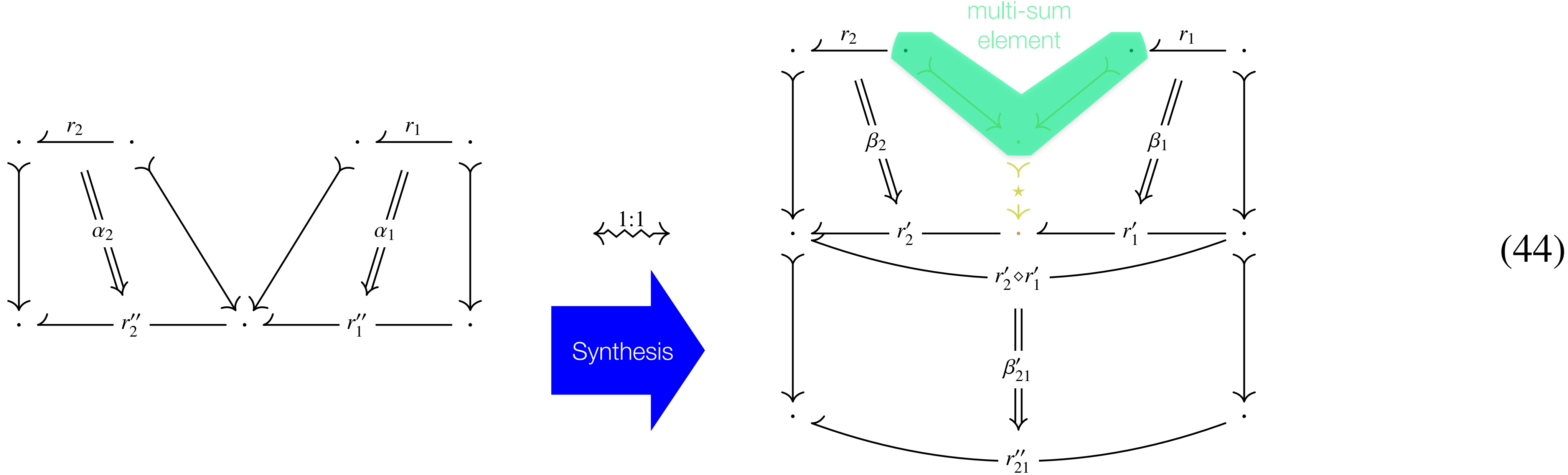
crDCs satisfy a (*universal!*) **Concurrency Theorem**

Theorem 8. Let \mathbb{D} be a *compositional rewriting double category*. Then the following statements hold (where the morphism marked \star in the diagram on the right is a *residue*, and the cospan into its domain a *multi-sum element*):



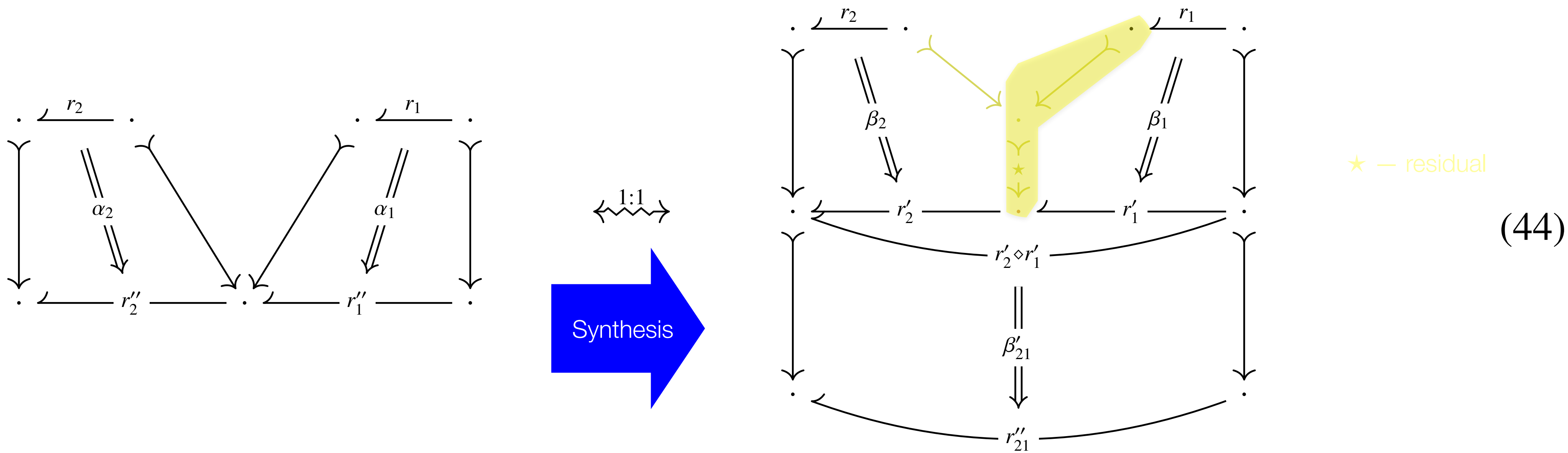
crDCs satisfy a (*universal!*) **Concurrency Theorem**

Theorem 8. Let \mathbb{D} be a *compositional rewriting double category*. Then the following statements hold (where the morphism marked \star in the diagram on the right is a *residue*, and the cospan into its domain a *multi-sum element*):



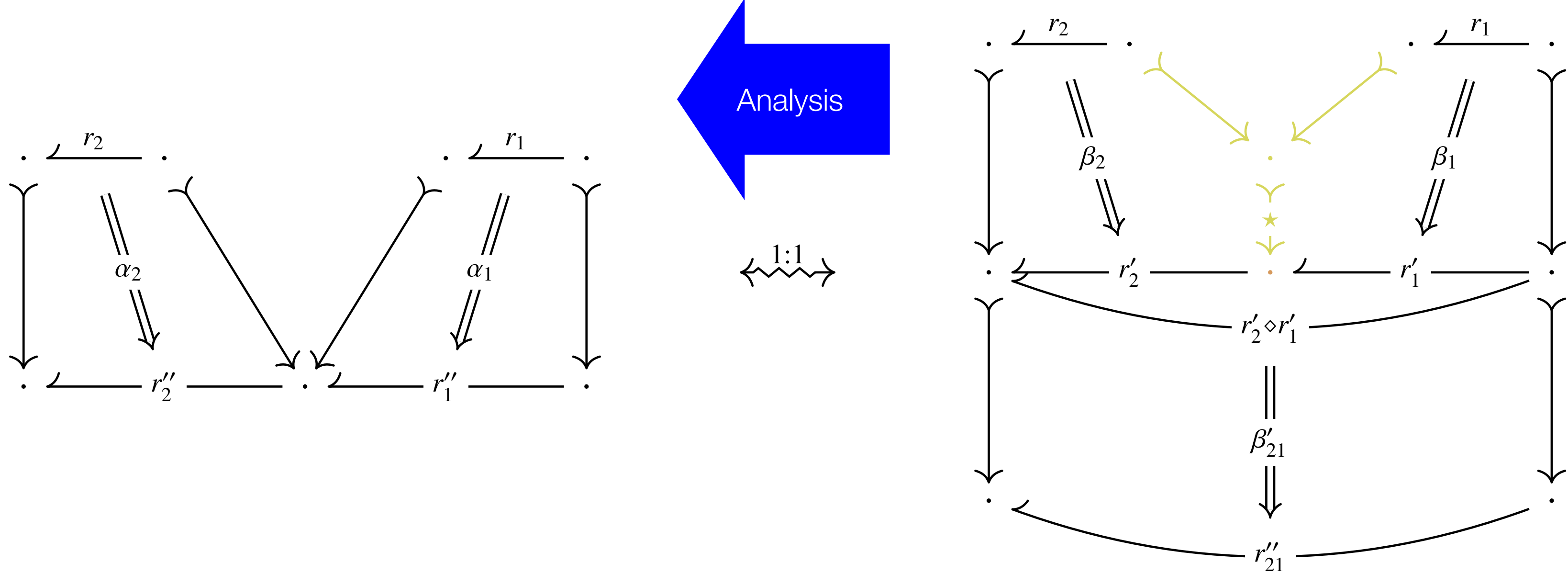
crDCs satisfy a (*universal!*) **Concurrency Theorem**

Theorem 8. Let \mathbb{D} be a *compositional rewriting double category*. Then the following statements hold (where the morphism marked \star in the diagram on the right is a *residue*, and the cospan into its domain a *multi-sum element*):



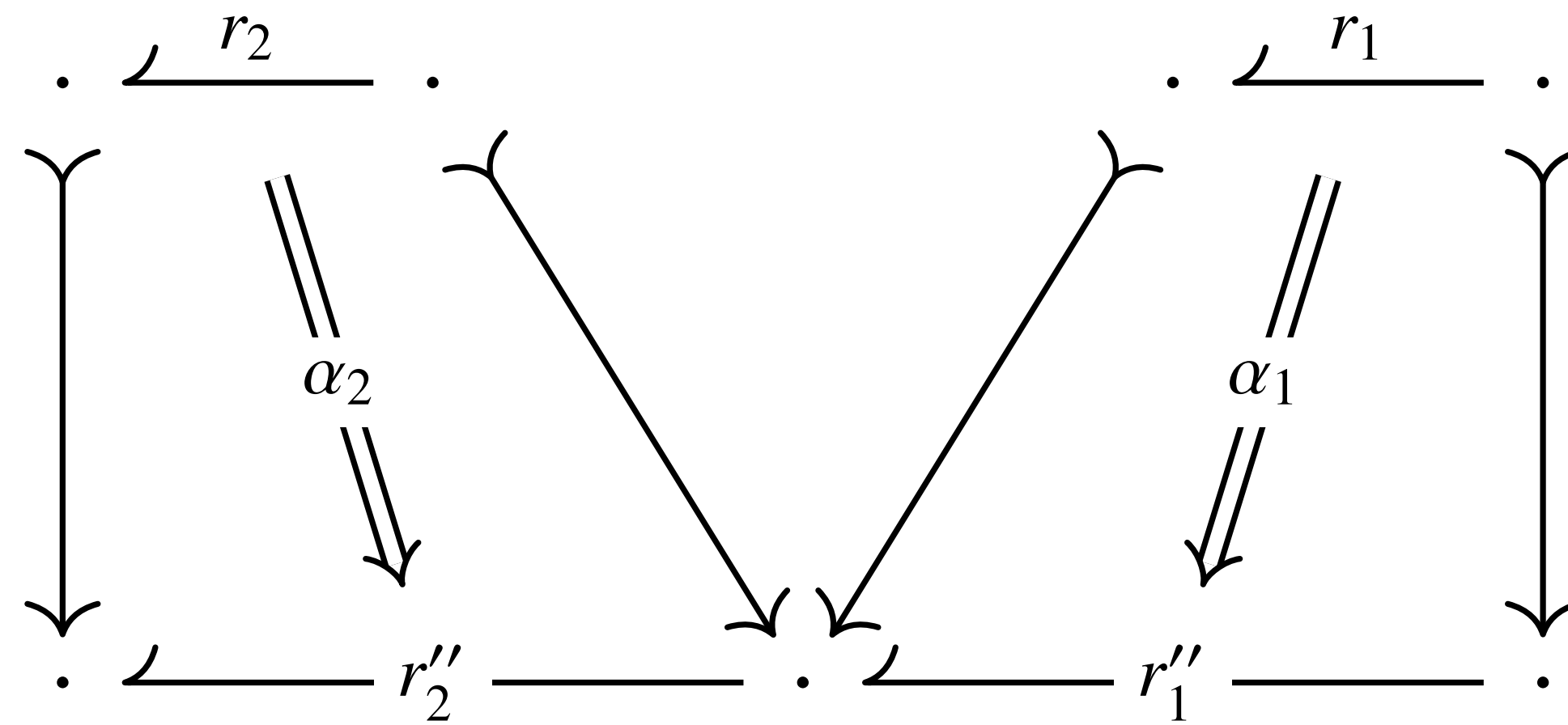
crDCs satisfy a (*universal!*) **Concurrency Theorem**

Theorem 8. Let \mathbb{D} be a *compositional rewriting double category*. Then the following statements hold (where the morphism marked \star in the diagram on the right is a *residue*, and the cospan into its domain a *multi-sum element*):



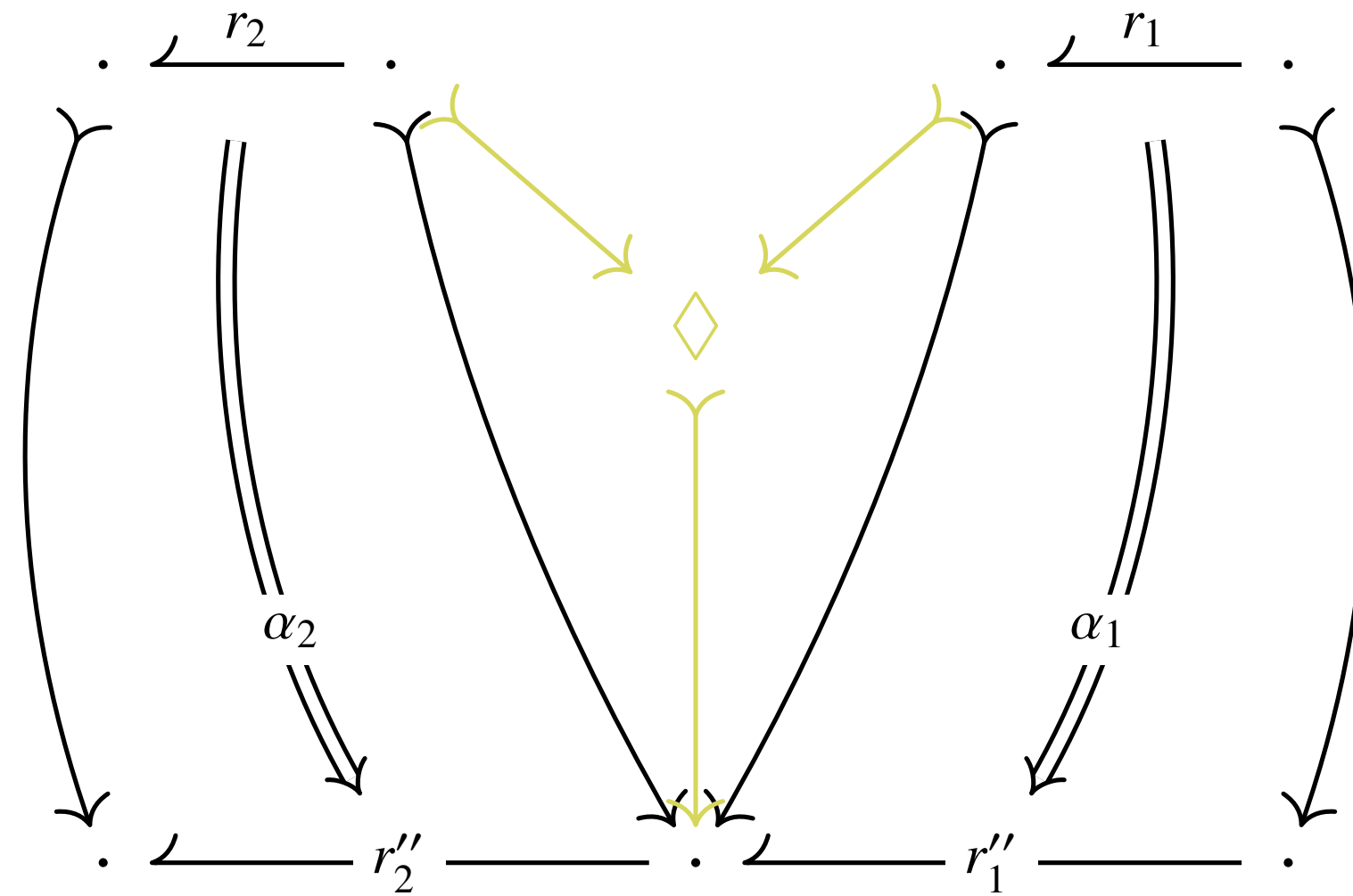
(44)

crDCs satisfy a (*universal!*) **Concurrency Theorem – PROOF**



PROOF. Synthesis part: Construct the diagram in (45) from the premise as follows:

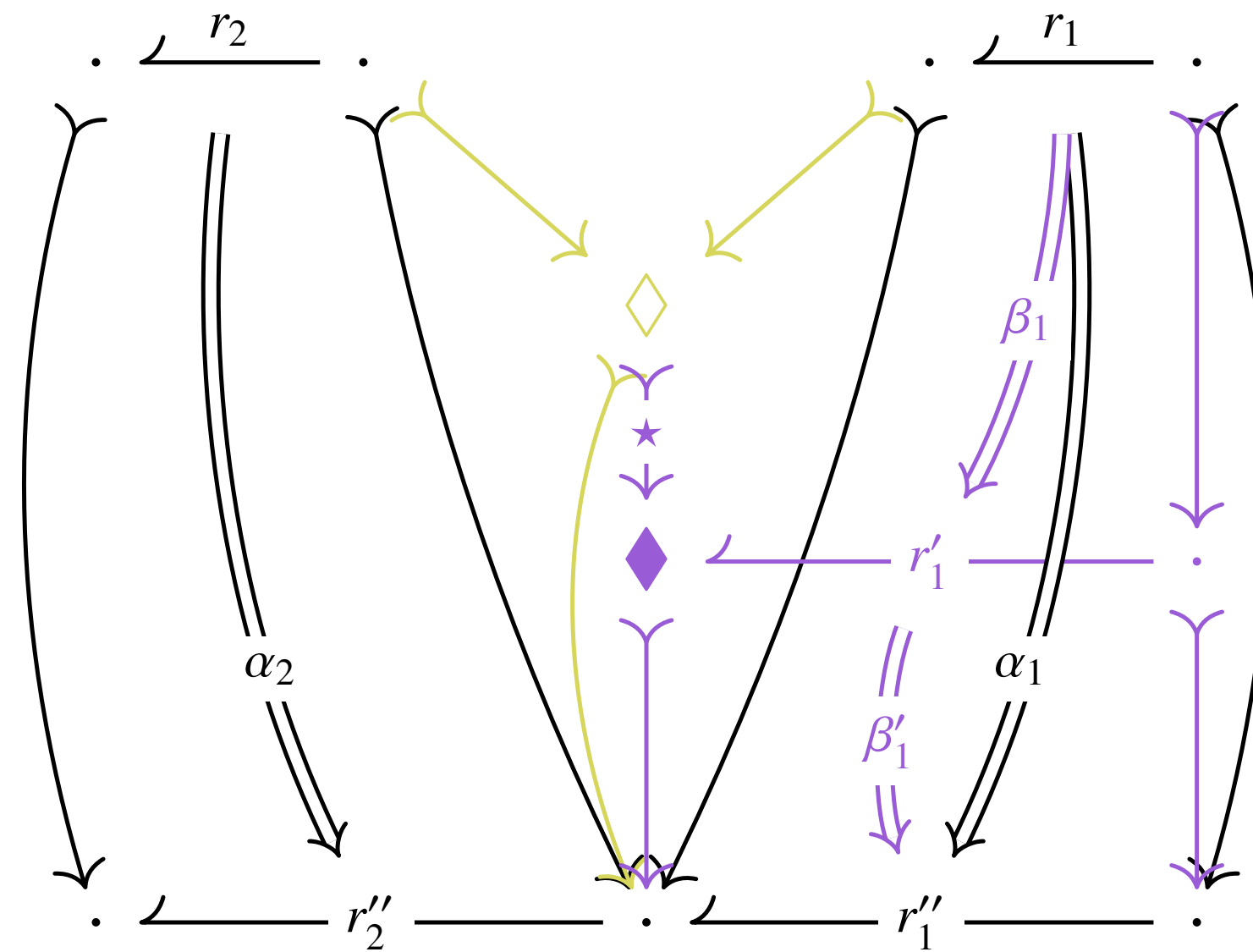
crDCs satisfy a (*universal!*) **Concurrency Theorem** – **PROOF**



PROOF. Synthesis part: Construct the diagram in (45) from the premise as follows:

- Via the **universal property of multi-sums**, there exists a cospan of \mathbb{D}_0 -morphisms into an object \diamond and a mediating \mathcal{M} -morphism $\diamond \rightarrow \cdot$.

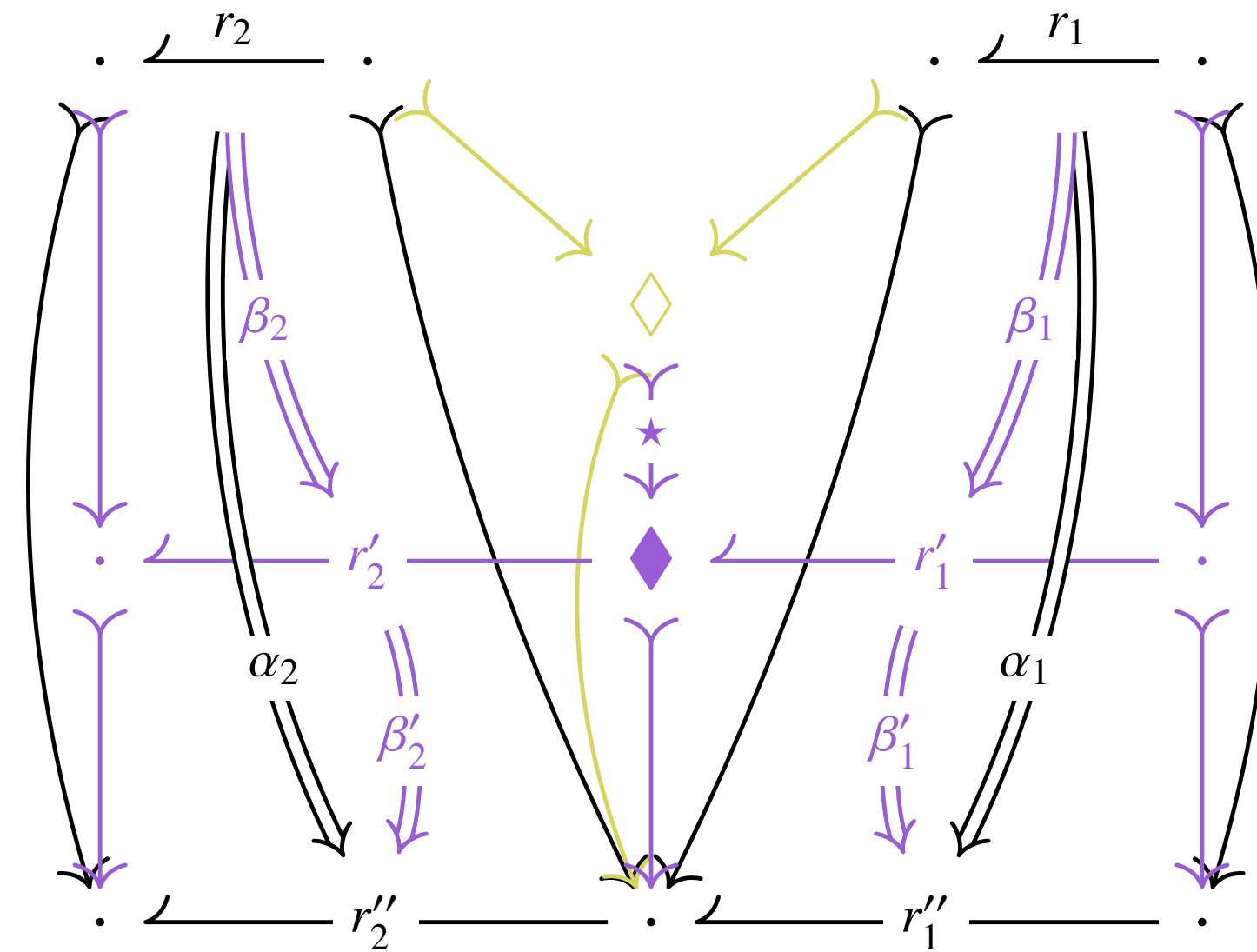
crDCs satisfy a (universal!) **Concurrency Theorem** – **PROOF**



PROOF. Synthesis part: Construct the diagram in (45) from the premise as follows:

- Via the **universal property of multi-sums**, there exists a cospan of \mathbb{D}_0 -morphisms into an object \diamond and a mediating \mathcal{M} -morphism $\diamond \multimap \cdot$.
- Since **the target functor $T : \mathbb{D}_1 \rightarrow \mathbb{D}_0$ is a residual multi-opfibration**, there exists a residue $\diamond \multimap \blacklozenge$ (marked \star) and an \mathbb{D}_0 -morphism $\blacklozenge \multimap \cdot$ such that $\alpha_1 = \beta'_1 \diamond_v \beta_1$.

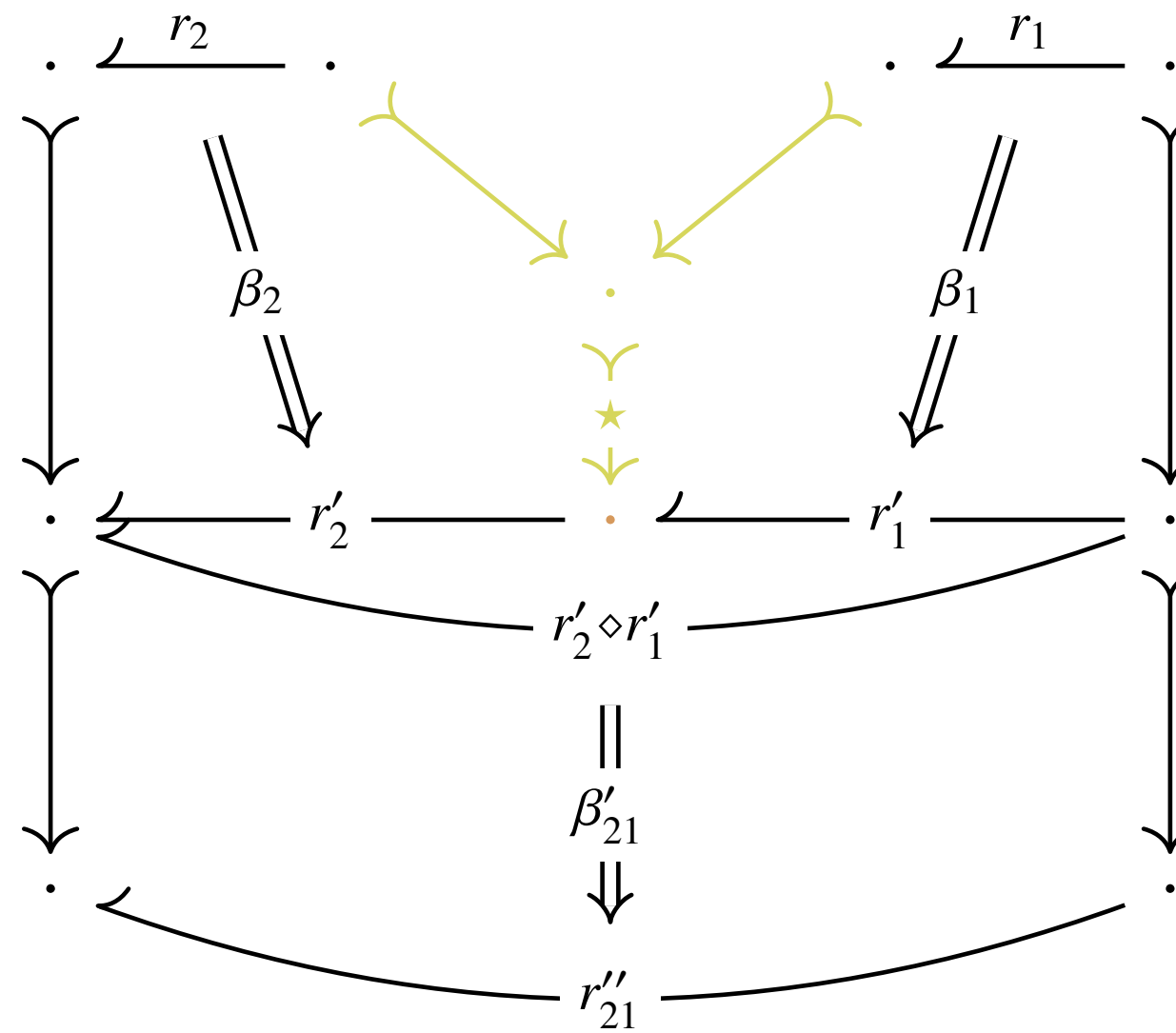
crDCs satisfy a (*universal!*) **Concurrency Theorem** – **PROOF**



PROOF. Synthesis part: Construct the diagram in (45) from the premise as follows:

- Via the **universal property of multi-sums**, there exists a cospan of \mathbb{D}_0 -morphisms into an object \diamond and a mediating \mathcal{M} -morphism $\diamond \rightarrow \cdot$.
- Since the target functor $T : \mathbb{D}_1 \rightarrow \mathbb{D}_0$ is a **residual multi-opfibration**, there exists a residue $\diamond \rightarrow \blacklozenge$ (marked \star) and an \mathbb{D}_0 -morphism $\blacklozenge \rightarrow \cdot$ such that $\alpha_1 = \beta'_1 \diamond_v \beta_1$.
- Since the source functor $S : \mathbb{D}_1 \rightarrow \mathbb{D}_0$ is a **multi-opfibration**, there exist direct derivations β_2 and β'_2 such that $\alpha_2 = \beta'_2 \diamond_v \beta_2$. Thus the claim follows by letting $\beta_{21} := \beta'_2 \diamond_h \beta'_1$.

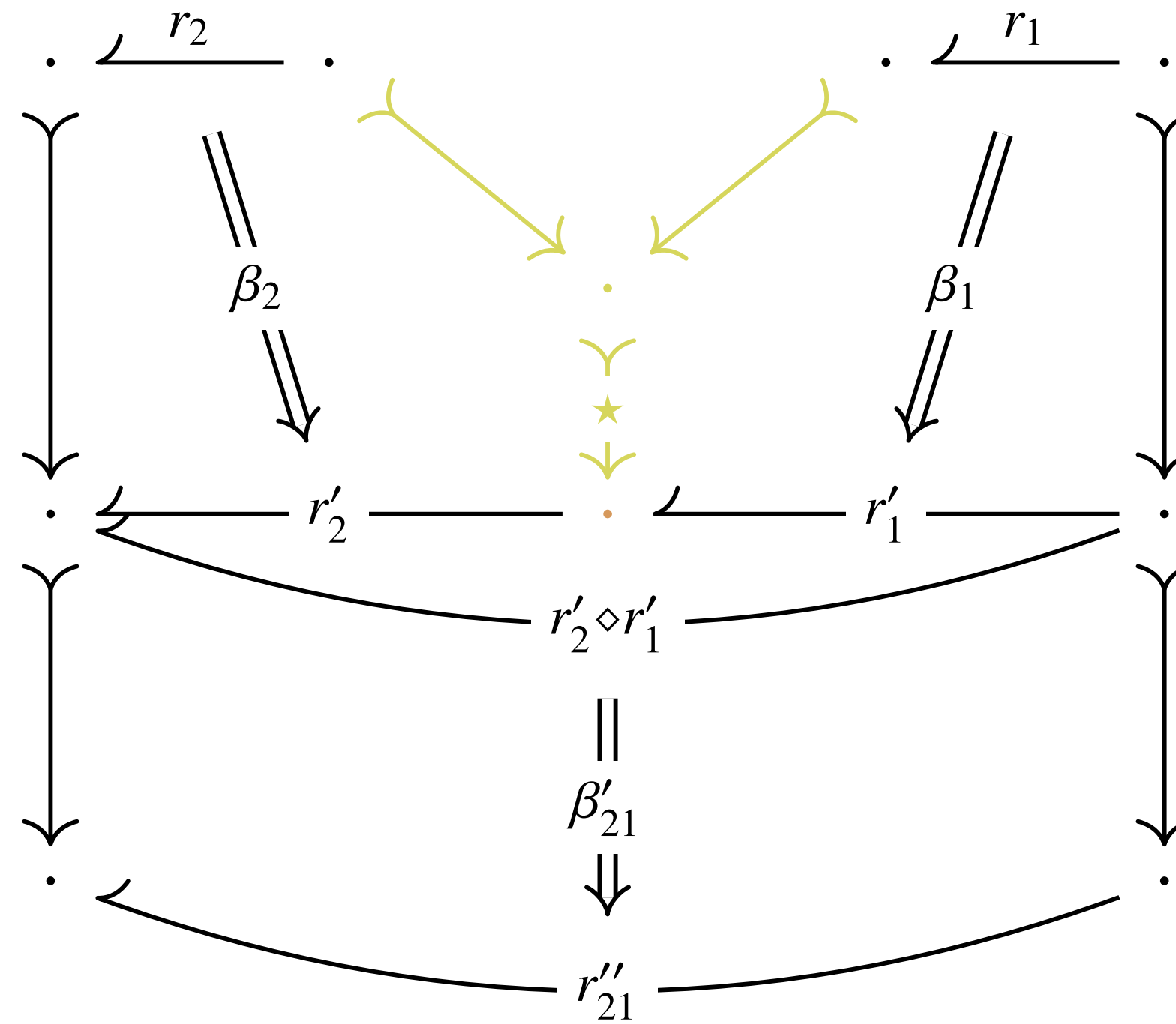
crDCs satisfy a (*universal!*) **Concurrency Theorem** – **PROOF**



PROOF. Synthesis part: Construct the diagram in (45) from the premise as follows:

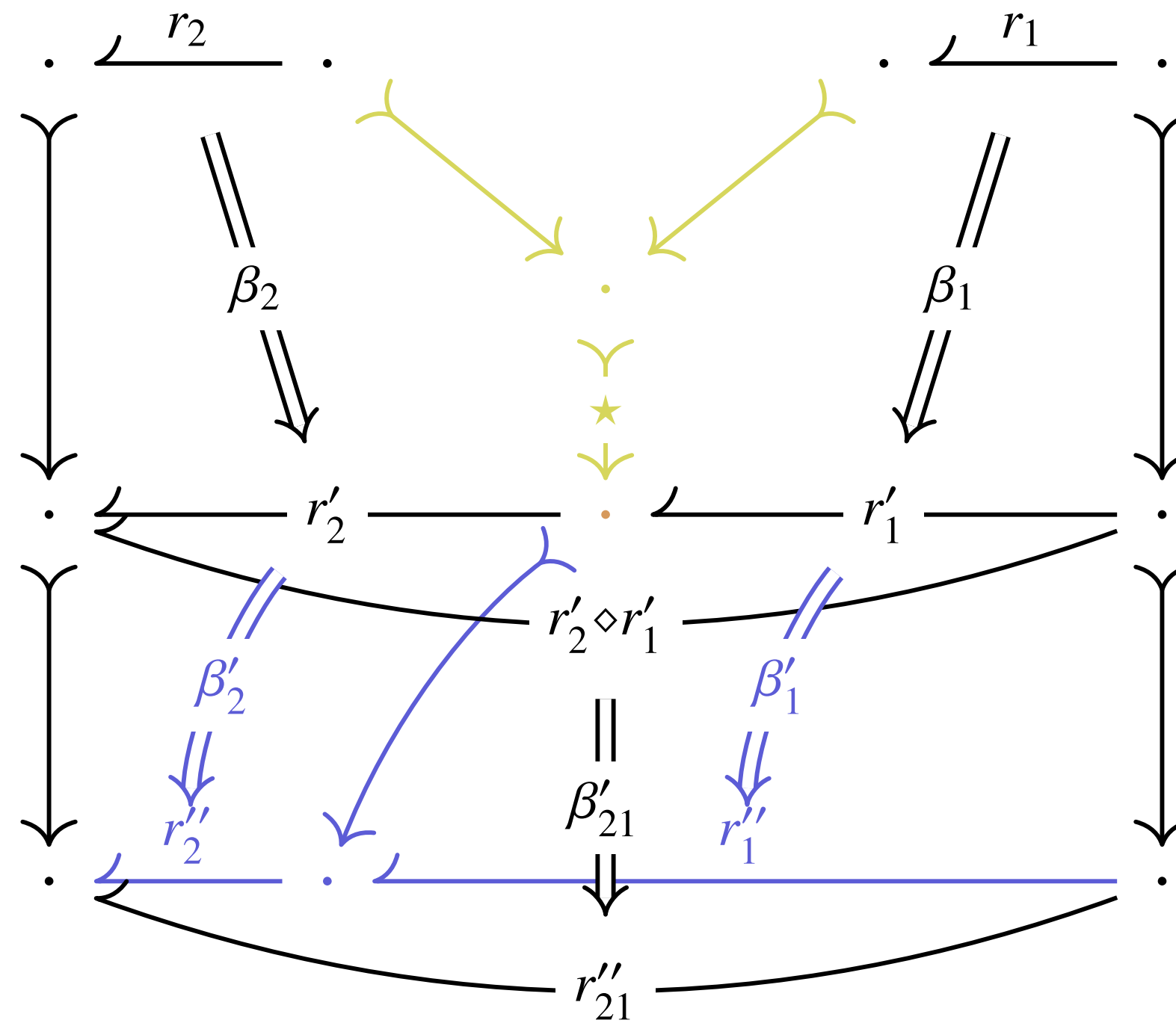
- Via the **universal property of multi-sums**, there exists a cospan of \mathbb{D}_0 -morphisms into an object \diamond and a mediating \mathcal{M} -morphism $\diamond \rightarrow \cdot$.
- Since **the target functor $T : \mathbb{D}_1 \rightarrow \mathbb{D}_0$ is a residual multi-opfibration**, there exists a residue $\diamond \rightarrow \blacklozenge$ (marked \star) and an \mathbb{D}_0 -morphism $\blacklozenge \rightarrow \cdot$ such that $\alpha_1 = \beta'_1 \diamond_v \beta_1$.
- Since **the source functor $S : \mathbb{D}_1 \rightarrow \mathbb{D}_0$ is a multi-opfibration**, there exist direct derivations β_2 and β'_2 such that $\alpha_2 = \beta'_2 \diamond_v \beta_2$. Thus the claim follows by letting $\beta_{21} := \beta'_2 \diamond_h \beta'_1$.

crDCs satisfy a (*universal!*) **Concurrency Theorem – PROOF**



Analysis part: Construct the diagram in (46) as follows:

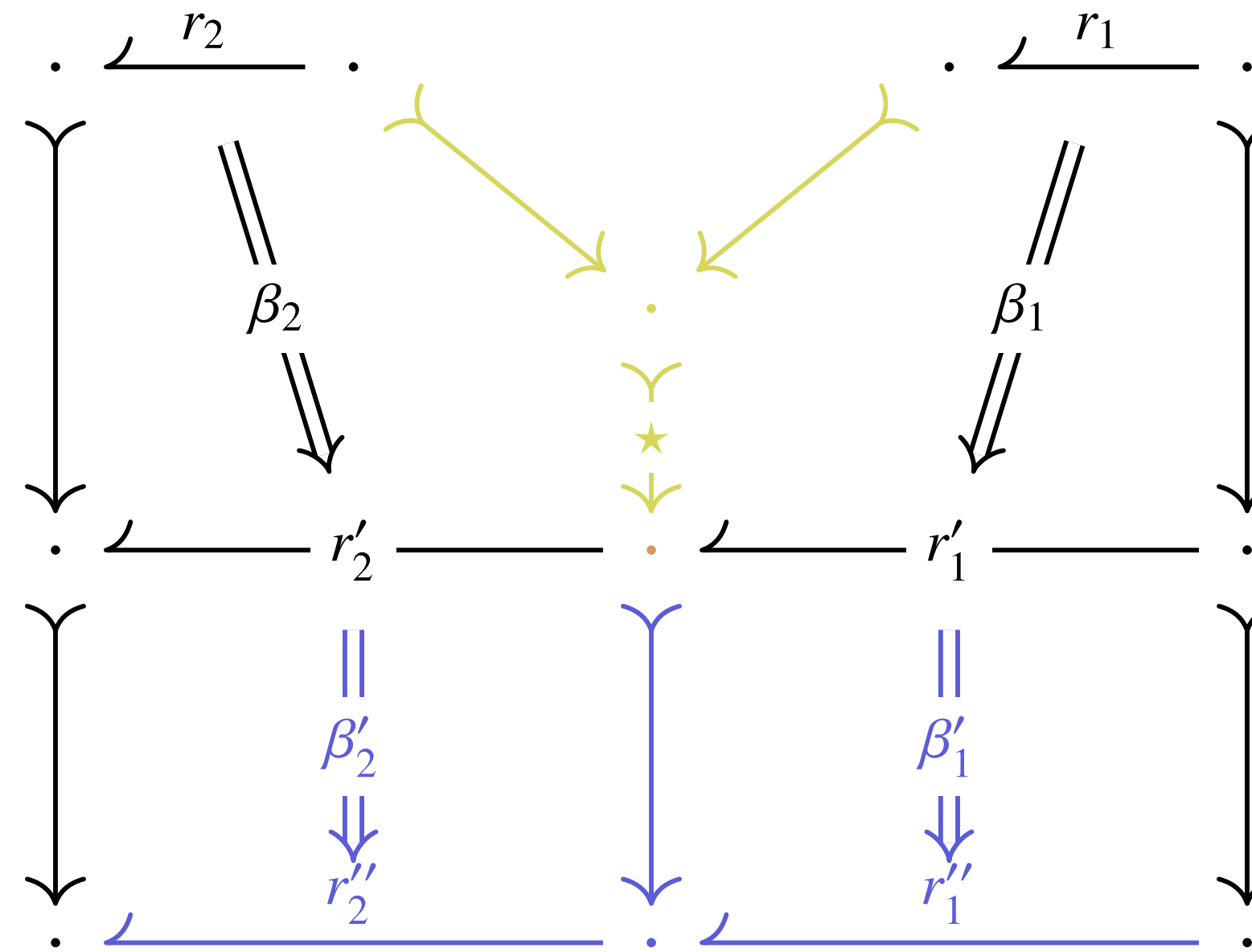
crDCs satisfy a (*universal!*) **Concurrency Theorem** – **PROOF**



Analysis part: Construct the diagram in (46) as follows:

- By the **horizontal decomposition property** of squares in \mathbb{D} , there exist squares β'_2 and β'_1 such that $\beta_{21} = \beta'_2 \diamond_h \beta'_1$.

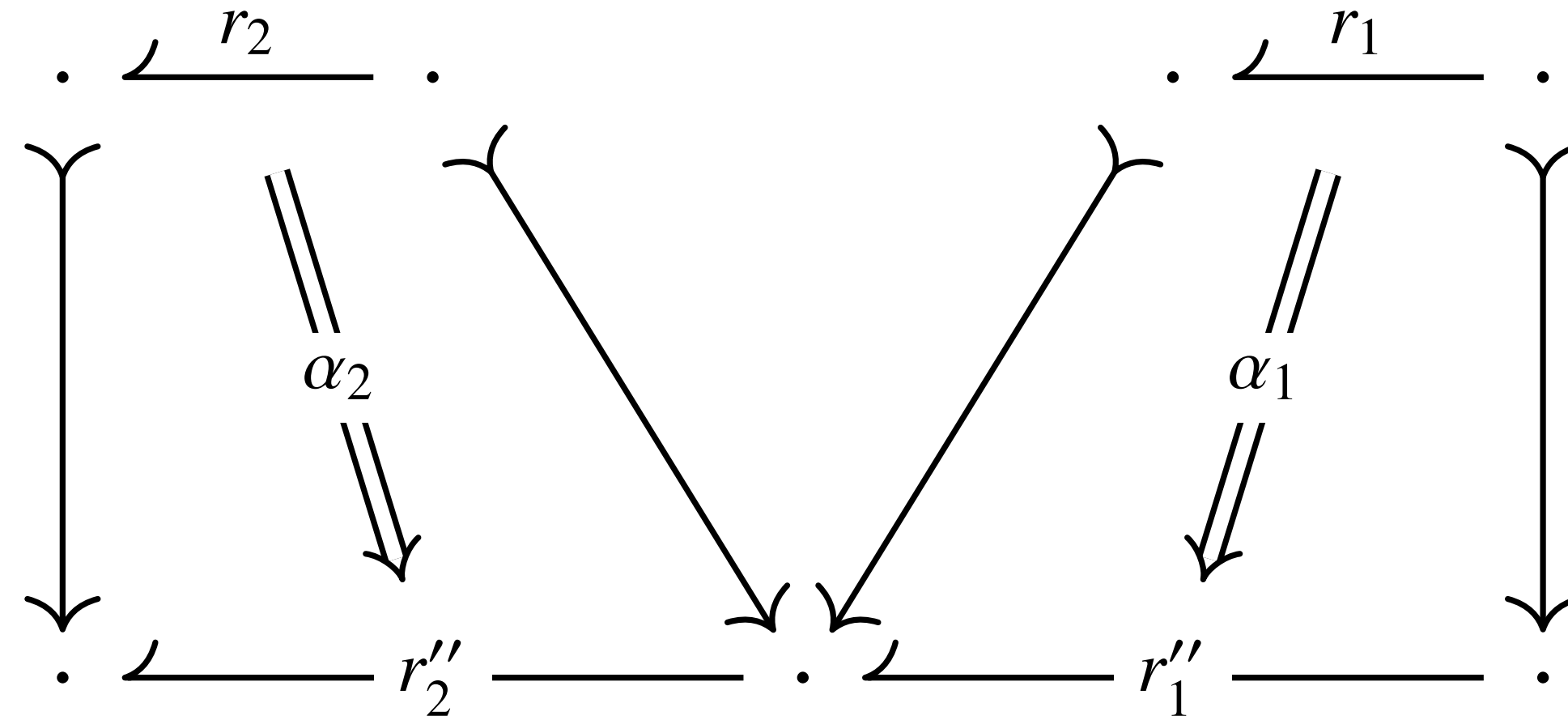
crDCs satisfy a (*universal!*) **Concurrency Theorem** – **PROOF**



Analysis part: Construct the diagram in (46) as follows:

- By the **horizontal decomposition property** of squares in \mathbb{D} , there exist squares β'_2 and β'_1 such that $\beta_{21} = \beta'_2 \diamond_h \beta'_1$.
- The claim follows by letting $\alpha_i := \beta'_i \diamond_v \beta_i$ for $i = 1, 2$.

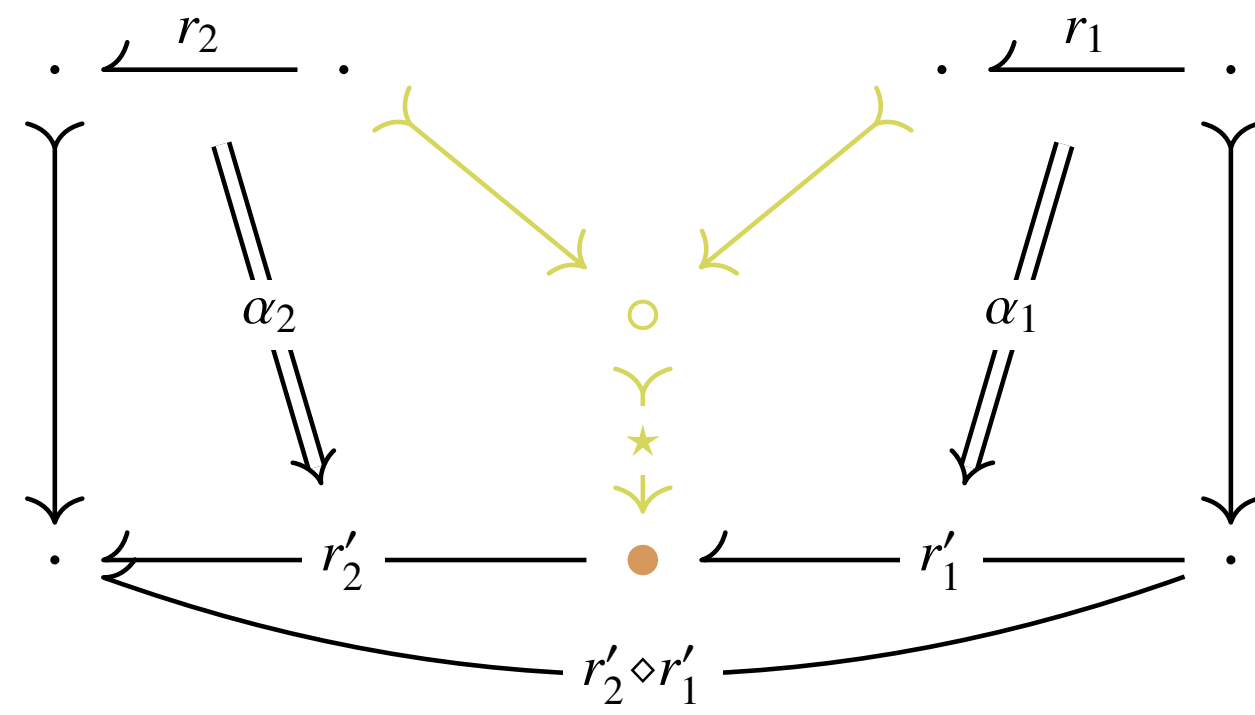
crDCs satisfy a (*universal!*) **Concurrency Theorem** – **PROOF**



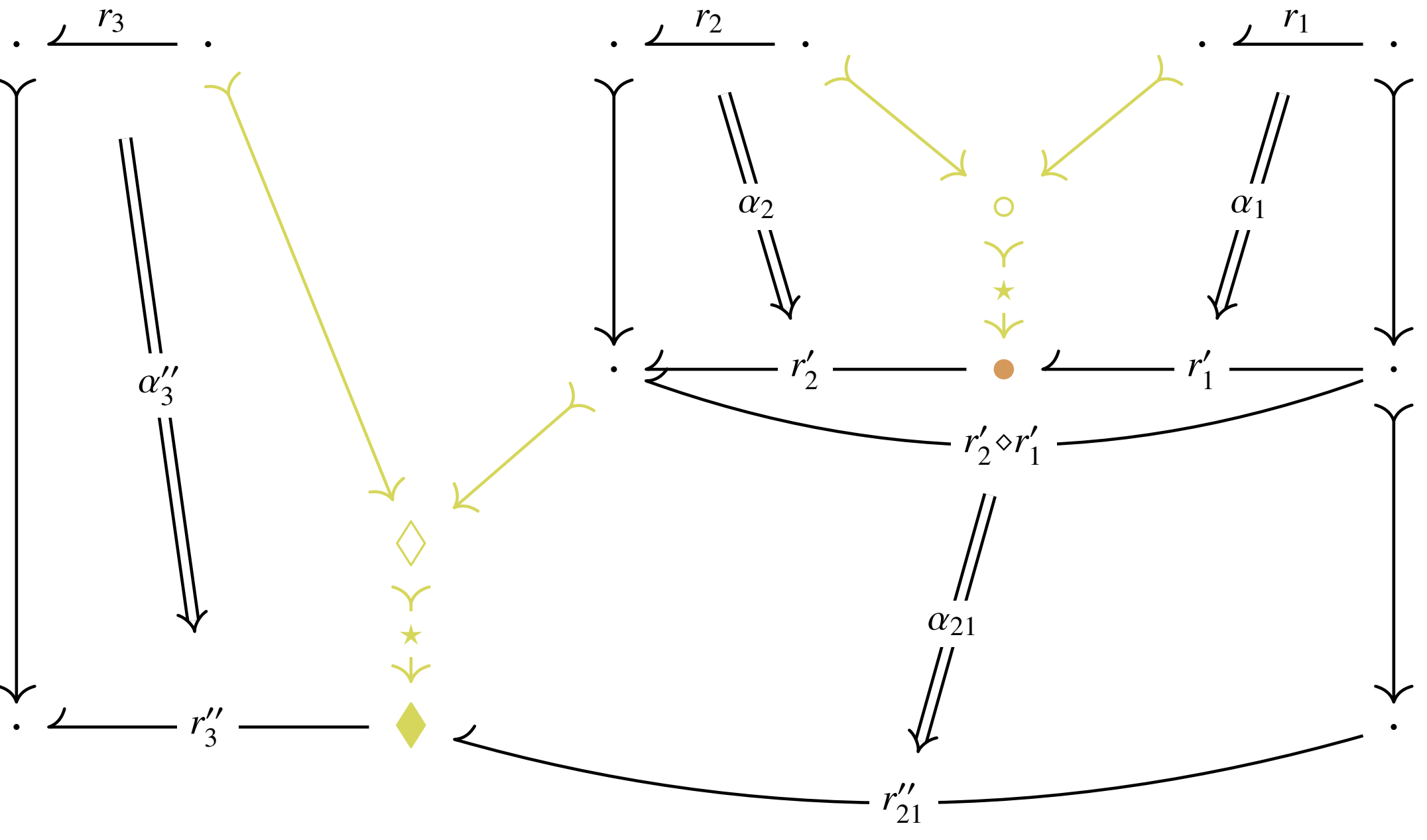
Analysis part: Construct the diagram in (46) as follows:

- By the **horizontal decomposition property** of squares in \mathbb{D} , there exist squares β'_2 and β'_1 such that $\beta_{21} = \beta'_2 \diamond_h \beta'_1$.
- The claim follows by letting $\alpha_i := \beta'_i \diamond_v \beta_i$ for $i = 1, 2$.

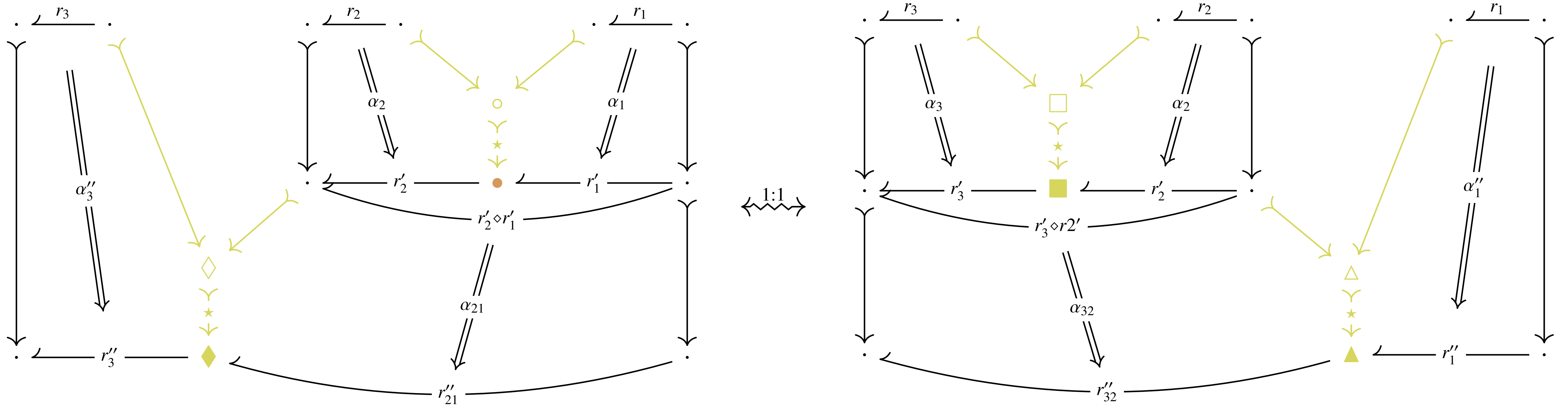
crDCs satisfy a (*universal!*) **Associativity Theorem** (= Thm. 9 in FCRT)



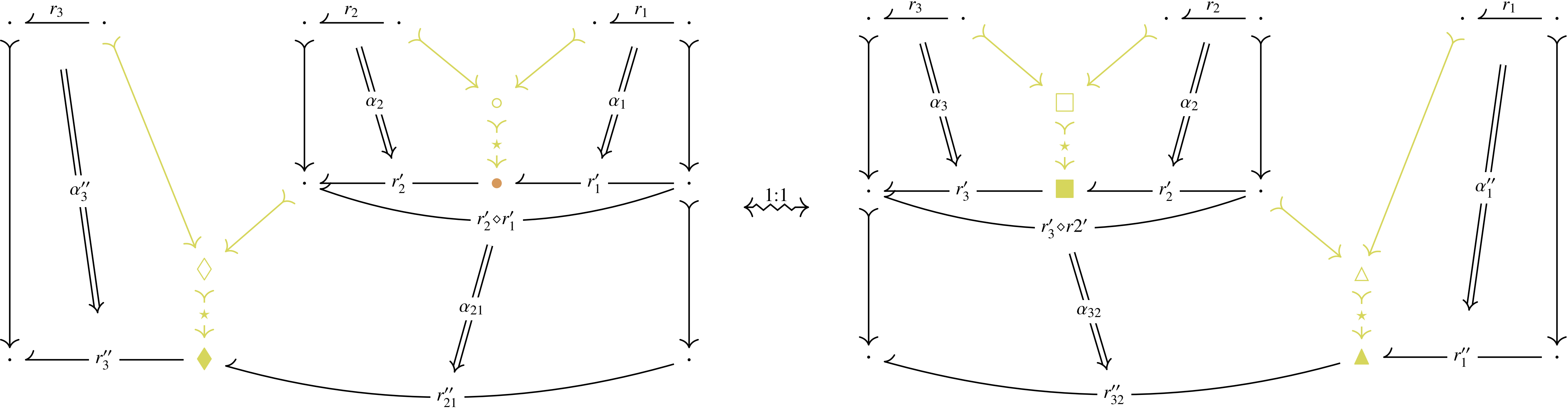
crDCs satisfy a (*universal!*) **Associativity Theorem** (= Thm. 9 in FCRT)



crDCs satisfy a (*universal!*) **Associativity Theorem** (= Thm. 9 in FCRT)



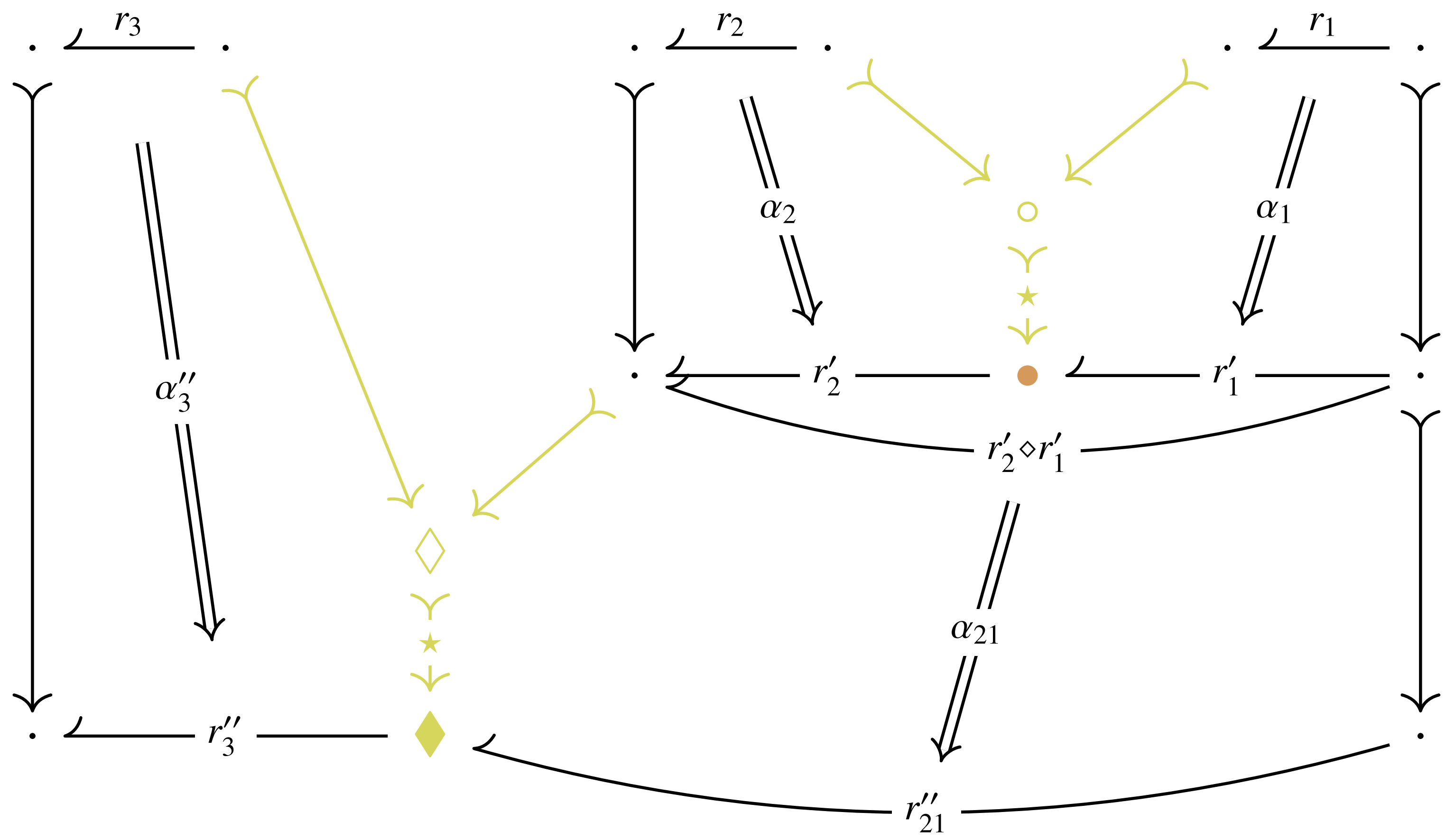
crDCs satisfy a (*universal!*) **Associativity Theorem** (= Thm. 9 in FCRT)



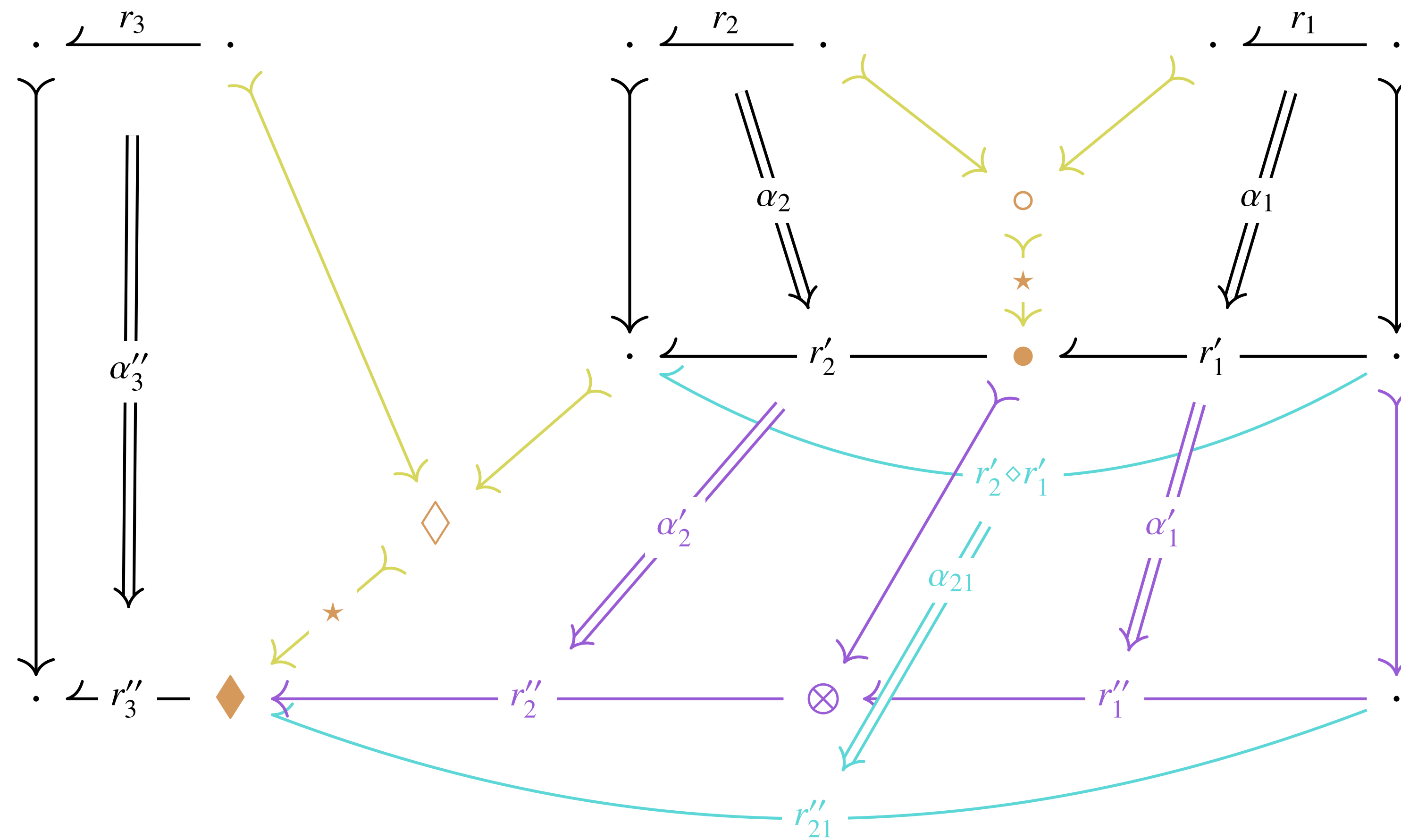
Moreover, the equivalence is such that in addition

$$r''_3 \diamond_h r''_{21} \cong r''_{32} \diamond_h r''_1 . \tag{49}$$

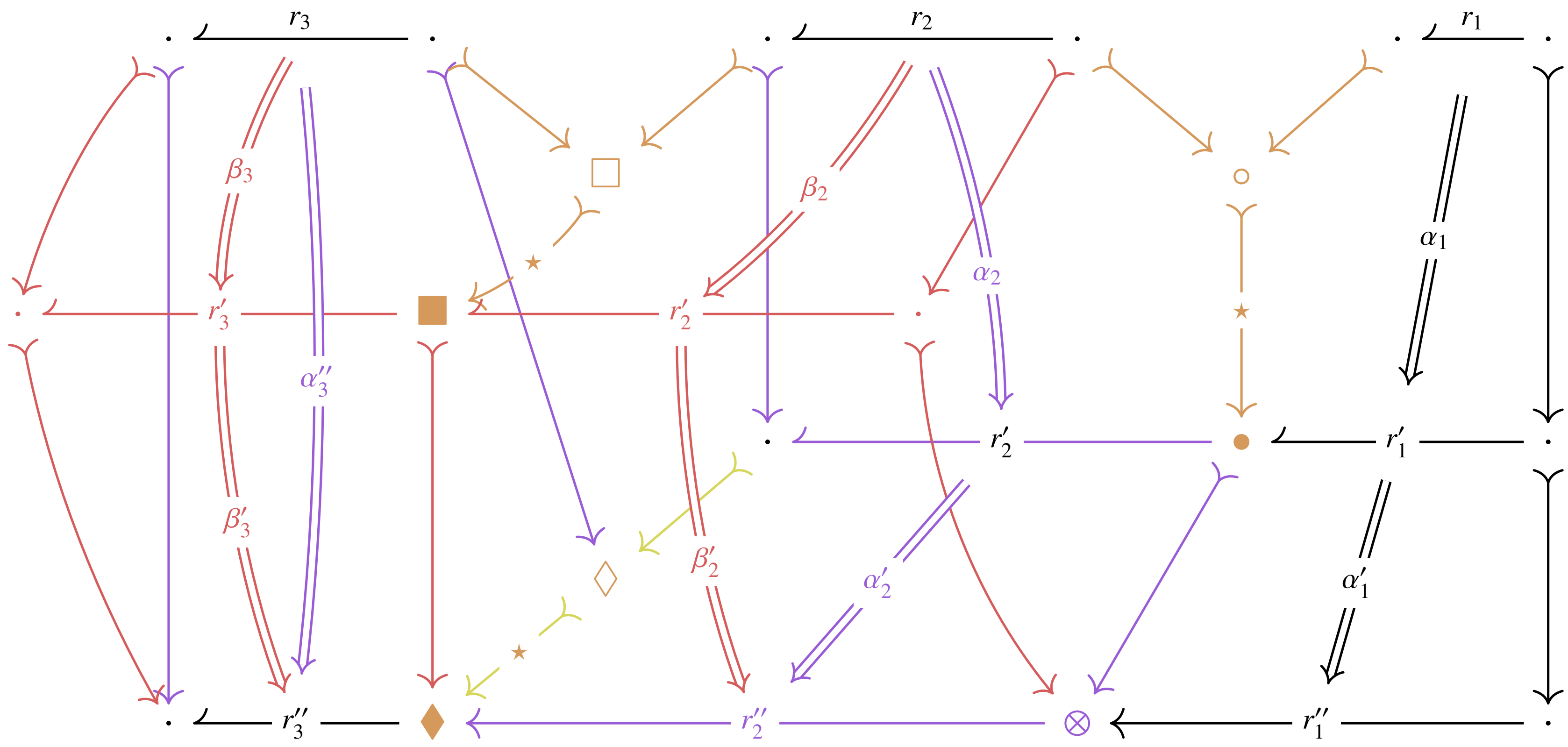
crDCs satisfy a (*universal!*) **Associativity Theorem** – PROOF SKETCH



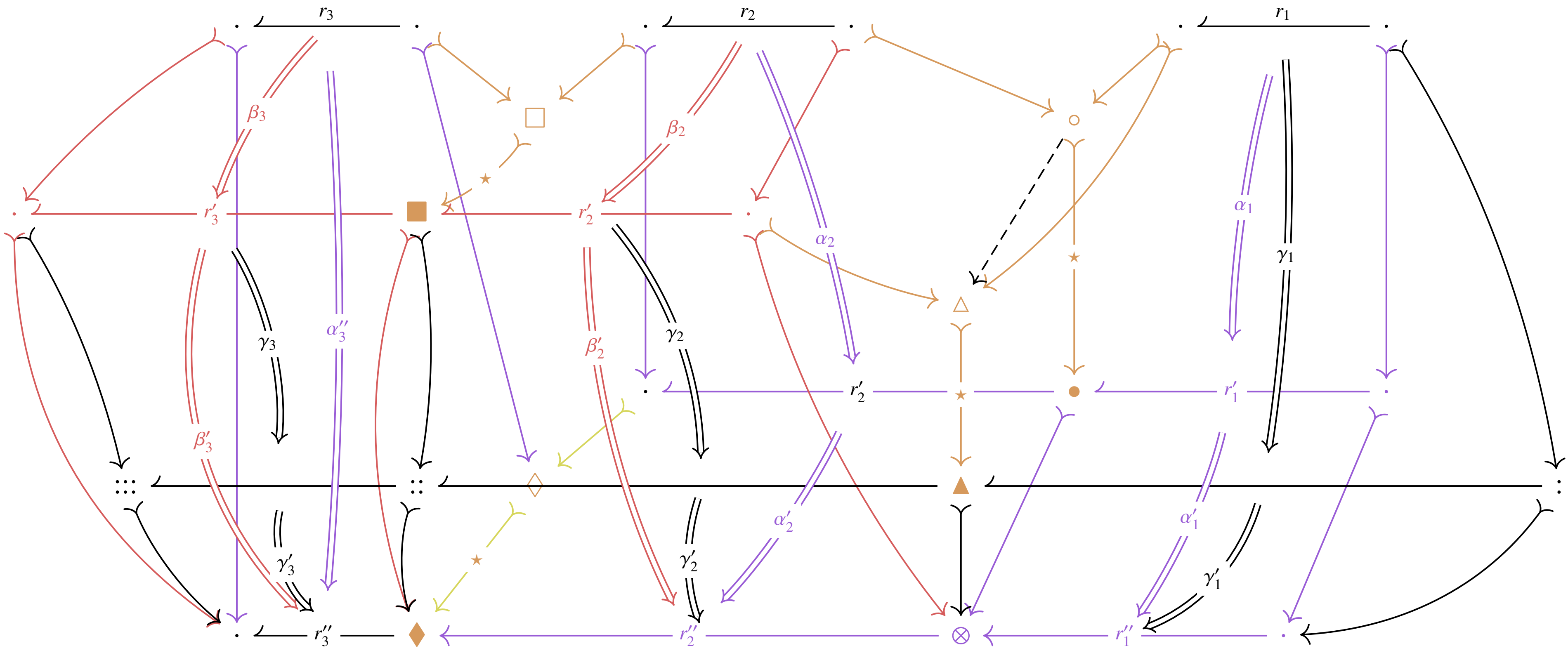
crDCs satisfy a (*universal!*) **Associativity Theorem** — PROOF SKETCH



crDCs satisfy a (*universal!*) **Associativity Theorem** — PROOF SKETCH



crDCs satisfy a (*universal!*) **Associativity Theorem** — PROOF SKETCH



crDCs satisfy a (*universal!*) **Associativity Theorem** — PROOF SKETCH

