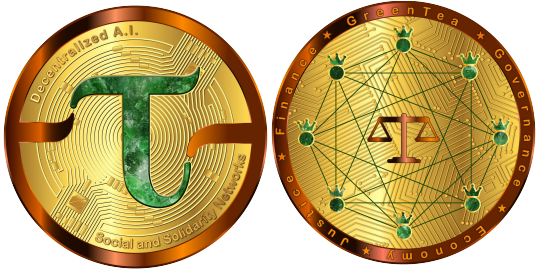
Démocratisation, Coordination et Financement

GreenTea, le langage des Organisations Communautaires

Constitution des Intelligences Collectives
Augmentées par l'Intelligence Artificielle
Enrichies par les Technologies de la Finance



Le Livre Blanc *de* GreenTea

August 22, 2022

Web 3 & Web 3

Web 3 : Utilisation Pair-à-Pair — Réseaux Sociaux

↪ Plateformes multi-acteurs, multi-agents

Web 3 : Infrastructure Pair-à-Pair — Blockchain

↪ dApp, DEX

Plateformes Académiques

JADE

- SDK de développement SMA impératif
- Implante les normes FIPA
- Fournit des architectures de référence

JACAMO

- Une approche déclarative pour la conception de SMA
  - ↪ Agents cognitifs
  - ↪ Artefacts environnementaux
  - ↪ Organisations explicites

GAMA

- Modélisation et Simulation de SMA spatialement explicites

Plateformes Industrielles

AKKA

- SDK pour applications concurrentes, distribuées et résilientes
- A base d'acteurs $\rightsquigarrow$ pas d'autonomie

DAML

- DSL pour les contrats intelligents
- Exécuté sur une blockchain

Conclusion

	JADE	JACAMO	GAMA	AKKA	DAML
Conception	∼	✓	✓	×	×
Déploiement	✓	✓	×	✓	✓
Analyse	∼	∼	∼	∼	∼
Capitalisation Réutilisation	∼	✓	×	×	∼
Accessibilité Fédérativité	×	×	×	×	∼
Sécurité	×	✓	×	×	∼
Décentralisation Respect de la vie privée	✓	✓	×	✓	✓

GreenTea

Un langage de domaine :

- Un **petit jeu d'instructions** pour un domaine précis : *les contrats intelligents*
- Distingue la **description** (partition) de l'**interprétation** (du pianiste) *(tagless final)*
- A l'intersection de :
  - ↪ **La théorie des catégories** / **Les systèmes multi-agents** / **La sémantique de Kripke**

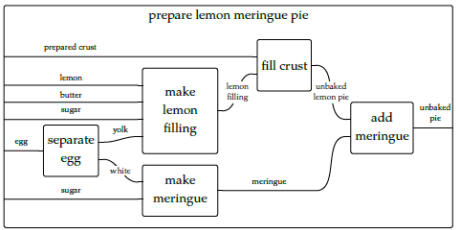
Implémenté en :

- **Scala 3**, avec l'écosystème **typelevel** (https://typelevel.org/)
  - ↪ permet une intégration de directives au niveau du compilateur et un contrôle de celui-ci
- La librairie **Cats** (https://typelevel.org/cats/)
  - ↪ importe la théorie des catégories (Haskell) dans Scala 3
- La librairie **Cats-Effect** (https://typelevel.org/cats-effect/)
  - ↪ gère l'exécution séquentielle, concurrentielle, asynchrone, ...

Quatre objets : le tea, l'interpréteur, le cache et le holon

- un TEA est automatiquement inféré depuis une expression en scala
  - ↪ val *increment* : Int $\Rightarrow$ Int = (n : Int) $\Rightarrow$ n + 1
  - ↪ val $increment_{tea}$: Int $\rightsquigarrow$ Int = (n : Int) $\Rightarrow$ n + 1

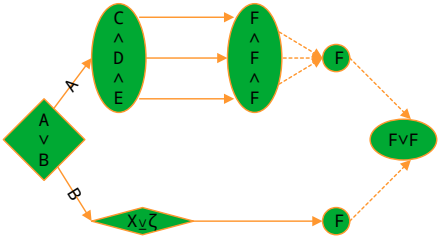
L'agnosticisme de la Théorie des Catégories



Catégories Monoïdales

- Une catégorie est une relation réflexive et transitive
  - ↪ Une relation entre deux éléments est appelée *morphisme* et est notée : $A \rightsquigarrow B$
  - ↪ Deux morphismes de $A \rightsquigarrow B$ et de $B \rightsquigarrow C$ se compose en $A \rightsquigarrow C$ par transitivité
  - ↪ La relation reflexive est appelée *identité* : id_A
- Les morphismes peuvent s'agréger par des opérateurs de groupe

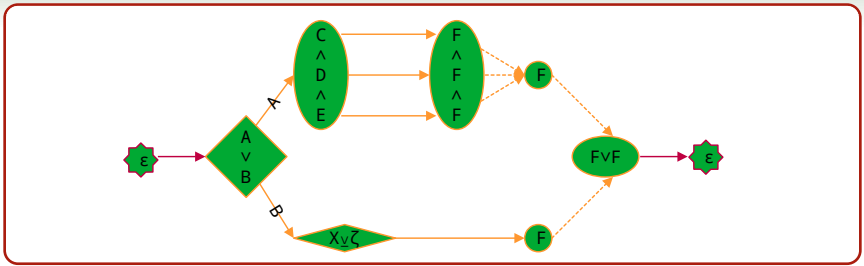
L'agnosticisme de la Théorie des Catégories



Un tea est structuré en *Tamis*

- Les TEA s'assemblent à l'aide de `and`, `xor`, `ior` et `andThen`.
- Un *tamis* représente graphiquement toutes les *Exécutions* possibles
  - ↪ Une certaine *Exécution* sera effectuée en fonction de l'entrée
  - ↪ L'entrée est une proposition de logique booléenne
- Le parallélisme est systémique / il est possible d'aller en arrière (cats-effect)

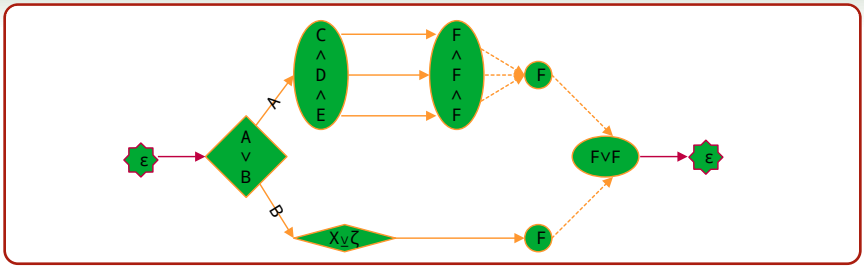
L'agnosticisme de la Théorie des Catégories



Les effets de bords grâce à la Catégorie de Ductor

- Une catégorie de Ductor étends une catégorie avec quatre objets
  - ↪ Un *environnement*, ϵ, n'appartenant pas au domaine initial de la catégorie
  - ↪ Des *sources* de `A` : des morphismes $\epsilon \rightsquigarrow A$, notés $+[A]$
  - ↪ Des *puits* de `B` : des morphismes $B \rightsquigarrow \epsilon$, notés $-[B]$
  - ↪ Tout *effet*, définit par un triplet $(+[A], A \rightsquigarrow B, -[B]) : \epsilon \rightsquigarrow \epsilon$
- Par propriété la catégorie de Ductor est fermée par composition

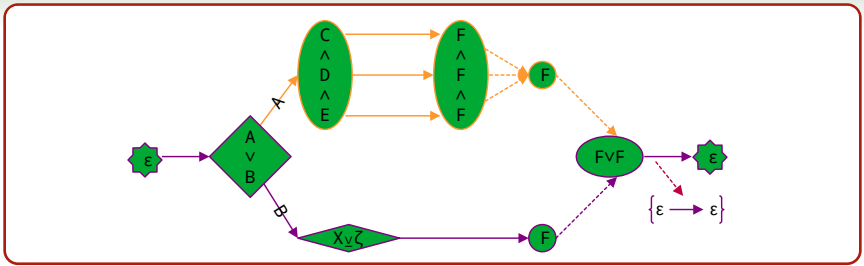
L'agnosticisme de la Théorie des Catégories



Pourquoi?

- Dans l'espace des endomorphismes les monades sont des monoïdes
- Un couple $(+[A], -[B])$ est appelé une *API* de A vers B
  - ↪ Une API encapsule les effets de bords
  - ↪ Une API gère l'accès concurrentiel (e.g. Sémaphore)
  - ↪ Une API gère l'acquisition et la libération de ressources (e.g Try/Finally)

L'agnosticisme de la Théorie des Catégories



Multi-interprétabilité

- Au cours d'une exécution, un TEA produit une structure δ de TEA
  - ↪ Un TEA est une fabrique de TEA (e.g. une DAO est une fabrique de contrat-intelligents)
  - ↪ Un TEA peut être enrichit par des observateurs qui rendent compte de son activité
    - ↪ rapport d'activité, de performance, d'analyse comptable ou financière, tests unitaires, ...

L'agnosticisme de la Théorie des Catégories

Orientée Objet

- `A` **est un** Oiseau : Oiseau ← A
- `A` : implémente *vole()*

↪ `def faireVoler(a:A) = a.vole()`

Théorie des Catégories

- `A` **a un** Oiseau : ∃ `Oiseau[A]`
- `Oiseau[A]` *habille* `A` avec `vole(a)`

↪ `def faireVoler(a:A)(using Oiseau[A]) = Oiseau[A].vole(a)`

En Théorie des Catégories, `A` est une donnée contractable

- Un contrat est un "constructeur de type" (morphisme dans l'espace des types)
  - ↪ `Oiseau[_] : A ⇒ Oiseau[A]`
- Une *Exécution* précise les *contrats* que `A` doit implémenté
  - ↪ *Donnée* : Option (null), Either (tryable), Future (asynchrone), ...
  - ↪ *Opérateur* : Comparable (ordre), Monoïd (agrégable), ...
  - ↪ *Calcul* : Functor (variable), Applicative (parallèle), Monad (séquentiel), ...
- La vérification de la conformité des contrats écrit pour `A` est automatisée
  - ↪ Tests par théorie (déjà pré-écrit) et non pas unitaires ⇝ gain de temps
- La Théorie des Catégories offre une flexibilité dans l'interprétation du code :
  - ↪ La fonction à exécuté n'est pas encodée dans l'objet mais spécifiée à l'exécution
  - ↪ Il est facile de la faire varier (method template pattern)
  - ↪ On peut parfois la créer de rien

L'agnosticisme de la Théorie des Catégories

Agnosticisme d'application

- Une application est construite en *assemblant* des fonctionnalités *atomiques*
  - ↪ on produit naturellement une "boite à Lego" industrialisable de fonctionnalités réutilisables

Agnosticisme d'environnement

- L'**API** distingue l'*obtention* des ressources de leur *utilisation*
  - ↪ un TEA se focalise sur transformer de façon *pure* une entrée en sortie

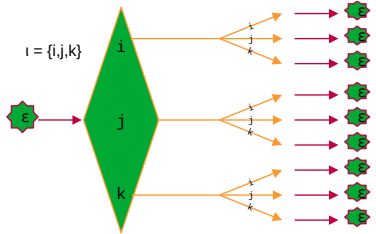
Agnosticisme d'interprétation

- L'**interpréteur** centralise les *implémentations* des contrats (*lettres grecs*)
  - ↪ Un même TEA peut être exécuté dans une variété de contextes
    - ↪ (concurrentielle/séquentiel, carte graphique/navigateur, JADE/mocké, ...)

Agnosticisme de système

- Un **holon** encode le système multi-agent (société + méthode de consensus)
  - ↪ un TEA est une description d'opérations décorrélées de la société

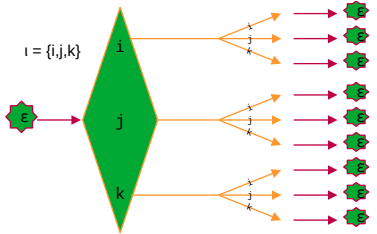
La Pluralité des SMA



Pluriel par nature

- Un tea $A \rightsquigarrow B$ définit un calcul pour chaque rôle i vers chaque rôle j d'un espace ι
  - ↪ $i \mapsto i$: un comportement au sein d'un agent/un agent au sein d'un SMA
  - ↪ $i \mapsto j$: un message
- Avec un *environnement* de **pages blanches**,
  - ↪ un *effet* est la description d'un **protocole de coordination multi-agent**

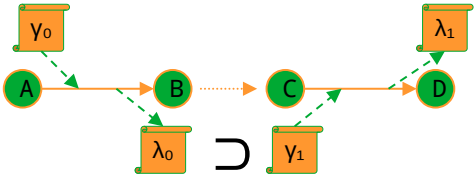
La Pluralité des SMA



L'holonisme grâce à la Flèche de Ductor

- Un holon généralise un TEA en rendant son **entrée plurielle** :
  - ↪ Multiple Instruction Single Data : $(A \rightsquigarrow_\iota B) \Rightarrow (A \rightsquigarrow_{org[\iota]} B)$
  - ↪ Single Instruction Multiple Data : $(A \rightsquigarrow_\iota B) \Rightarrow (Struct[A] \rightsquigarrow_\iota B)$
- La résolution de la pluralité (**consensus**) est définie par le holon fournit
  - ↪ exemples formés à partir des patrons de conception de la Théorie des Catégories : runNcompute (vote), computeNRun (election), runNfold (calcul parallele), foldNrun (world café),...
- Un *holon* appliqué à un *effet* permet l'**execution d'un SMA**

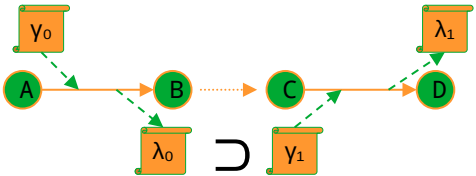
La Rationalité de la Sémantique de Kripke



Cache au niveau des types

- Une table de hachage dont les clés sont des types
- Le compilateur peut vérifier la présence de clés
- Sémantique ensembliste : inclusion, ajout, union, intersection

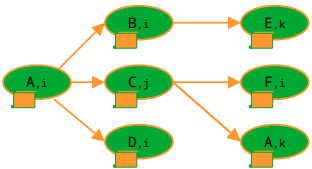
La Rationalité de la Sémantique de Kripke



Utilisation

- Assertions (require/ensure)
  - ↪ La composition de TEA nécessite que le cache en entrée soit inclus dans le cache en sortie
- Machine à état
  - ↪ Un état peut être passé et mise à jour en marge du calcul principal
- Représentation mentale du monde
  - ↪ e.g FIPA ACL Semantic Language
- Information globale sur l'exécution
  - ↪ e.g Log d'activité

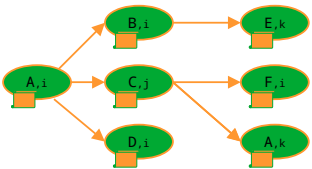
La Rationalité de la Sémantique de Kripke



Modèle de Kripke

- Un univers Ω : l'ensemble des nœuds (identifiant d'agent, entrée, cache)
- Une relation d'accessibilité $A \triangleright B$ faites :
  - ↪ d'une composante logique : la capacité de faire le calcul, un TEA, $A \rightsquigarrow B$
  - ↪ d'une composante déontique : l'autorisation de faire le calcul
- une relation d'évaluation $\omega \models P$ valide si P est dans le cache de A

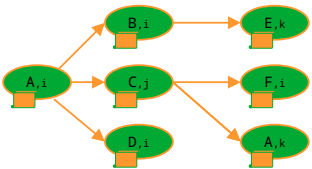
La Rationalité de la Sémantique de Kripke



La logique aléthique

- Au sujet des **successeurs** d'un nœud soit :
  - ↪ **Tous** vérifient une propriété P
  - ↪ **Certains** vérifient une propriété P
  - ↪ **Au moins un** vérifie une propriété P
  - ↪ **Aucun** ne vérifie une propriété P

La Rationalité de la Sémantique de Kripke



La logique temporelle

- Au sujet des **descendant** d'un noeud soit :
  - ↪ **Tous** leurs successeurs vérifient une propriété P, *ainsi que leurs descendants*
  - ↪ **Certains** de leurs successeurs vérifient une propriété P, *ainsi que leurs descendants*
  - ↪ **Au moins un** de leurs successeurs vérifie une propriété P, *ainsi que ses descendants*
  - ↪ **Aucun** de leurs successeurs ne vérifie une propriété P, *ainsi qu'aucun de leurs descendants*

La Rationalité de la Sémantique de Kripke

La sémantique de Kripke permet d'exprimer :

- L'évolution de propriétés dans le temps / des préférences de groupe
- Le droit / La comptabilité
- La connaissance / Les croyances / Les opinions
- ...

Elle est directement encodé dans le jeux d'instructions fournit :

- `inform.{yes,maybe,failed,impossible,refused}`
- `request.{execution,delegation,observation,cancelation}`
- `*.{forall, forsome}.{values,neighbors}`
- ...

En conclusion :

- Un TEA fournit bien une infrastructure compatible à la Sémantique de Kripke
- Certaines vérifications sont faites à la compilation, lors de l'assemblage (Hoare)
- L'intégration de primitives de raisonnement et de vérification avancées est en cours

Avantages

	GreenTea	JADE	JACAMO	GAMA	AKKA	DAML
Conception	✓	∼	✓	✓	×	×
Déploiement	✓	✓	✓	×	✓	✓
Analyse	✓	∼	∼	∼	∼	∼
Capitalisation Réutilisation	✓	∼	✓	×	×	∼
Accessibilité Fédérativité	✓	×	×	×	×	∼
Sécurité	✓	×	✓	×	×	∼
Décentralisation Respect de la vie privée	✓	✓	✓	×	✓	✓

Avantages

Welfare Engineering

- Distinction description/interprétation
  - ↪ un même TEA sert à la coordination/prédiction/optimisation/analyse

Capitalisation & Réutilisation

- les TEA sont fonctionnellement atomique et réassemblable

Accessibilité & Fédérativité

- Les TEA sont des organigrammes ⇒ facilement compréhensible
- Approche nocode avec un outils de type moteur BPMn
- Contraintes d'architecture force un code industriel,
  - ↪ transférable de la recherche à l'industrie
- L'atomicité des productions permet un modèle de licence hybride

Sécurisé, décentralisable et respectueux de la vie privée

- Grâce à la logique de Hoare, à la sémantique de Kripke et aux systèmes multi-agents