Configuration Languages
Overview and Semantics of CUE

Eloi Perdereau   Jacques Noyé

December 14, 2023

VELVET days
Nantes

Introduction

- Configuration is everywhere
  - from YAML to API schemas
- Cross domain, many approaches, no standard definition
- Our view: *It's all just about manipulating trees.*
  1. Define structures (tree with references).
  2. Use predicates for verification.
  3. Use functions for operations.

Family of "Programmable Configuration Languages"

- Abstractions with greater safety guarantees
- e.g. BCL, Jsonnet, Dhall, Nickel, CUE

CUE as a Case Study

- Why ? Production ready and of scientific interest
- Declarative Composition paradigm
  - Types = Values, interpreted as *sets*,
  - Single namespace,
  - Based on Unification (by refinement, no variables),
  - Order insensitive and Immutable.

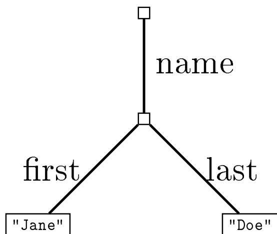
Configure, Unify, Execute

1. Trees
2. CUE Core Semantics: K-CUE
3. More on CUE and Beyond

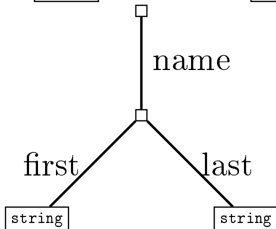
Outline

1. Trees
2. CUE Core Semantics: K-CUE
3. More on CUE and Beyond

Data Trees

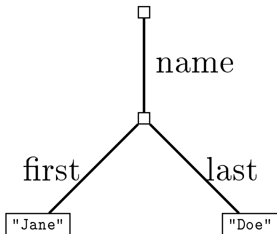



```
name: {
  first: "Jane"
  last: "Doe"
}
```



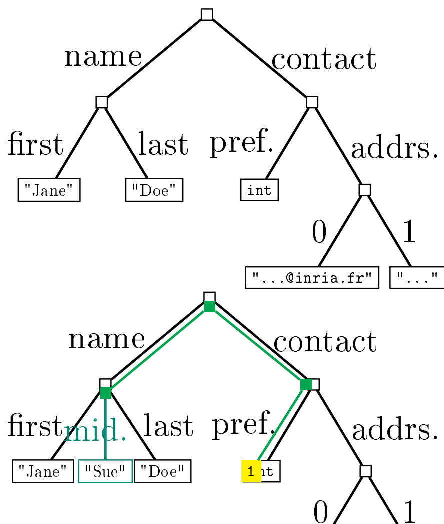

```
name: {
  first: string
  last: string
}
```

Data Trees




```
name: {
  first: "Jane"
  last: "Doe"
}
```

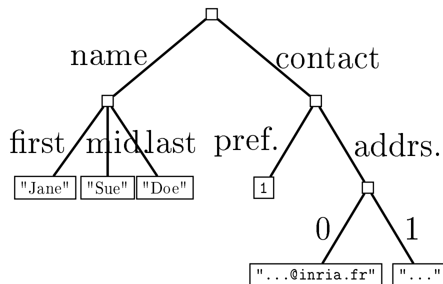
Unification: Match by Labels




```
name: {
  first: "Jane"
  last: "Doe"
}
contact: {
  preferred: int
  addresses: [ /* ... */ ]
}
```

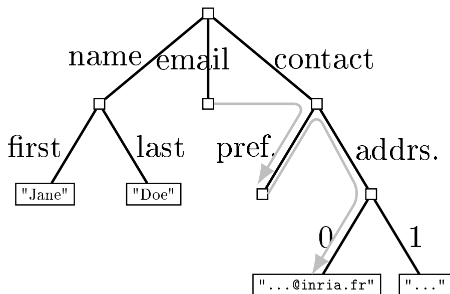
```
name: {
  middle: "Sue"
}
contact: {
  preferred: 1
}
```

Merged Tree




```
name: {
  first: "Jane"
  last: "Doe"
  middle: "Sue"
}
contact: {
  preferred: 1
  addresses: [ /* ... */ ]
}
```

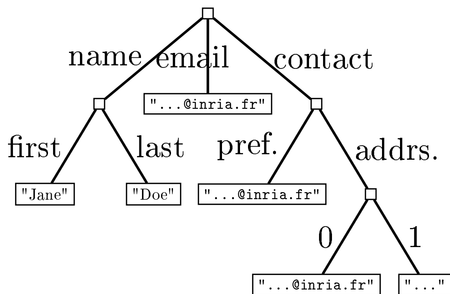
References



References represents node **copies**.

```
name: {
  first: "Jane"
  last: "Doe"
}
email: contact.preferred
contact: {
  preferred: contact.addresses[0]
  addresses: [
    "jane.doe@inria.fr",
    "jane.doe@imt-atlantique.fr",
  ]
}
```

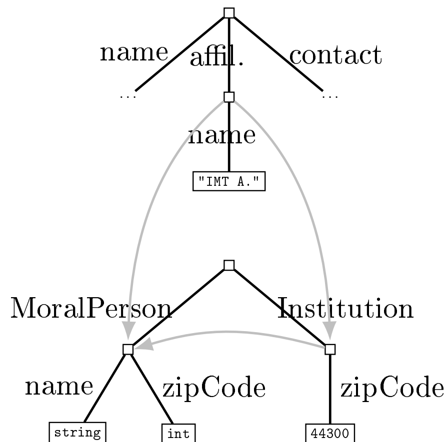
References Resolved



References represents node **copies**.

```
name: {
  first: "Jane"
  last: "Doe"
}
email: "jane.doe@inria.fr"
contact: {
  preferred: "jane.doe@inria.fr"
  addresses: [
    "jane.doe@inria.fr",
    "jane.doe@imt-atlantique.fr",
  ]
}
```

Embedded References in Conjunctions

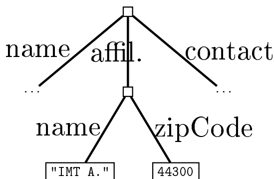


References can be *embedded*.

```
name: _ // ...
contact: _ // ...
affiliation: {
  name: "IMT Atlantique"
  MoralPerson
  Institution
}
```

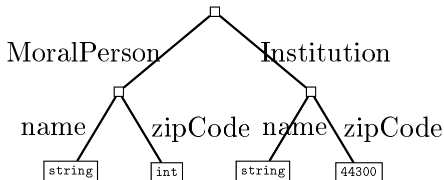
```
MoralPerson: {
  name: string
  zipCode: int
}
Institution: {
  zipCode: 44300
  MoralPerson
}
```

Embedded References Resolved



References can be *embedded*.

```
name: _ // ...
contact: _ // ...
affiliation: {
  name: "IMT Atlantique"
  zipCode: 44300
}
```




```
MoralPerson: {
  name: string
  zipCode: int
}
Institution: {
  name: string
  zipCode: 44300
}
```

Outline

1 Trees

2 CUE Core Semantics: K-CUE

3 More on CUE and Beyond

K Semantic Framework

- Based on *Matching Logjc* [Roş17, CLR21],
- inspired by *Rewriting Logic*: J. Meseguer, G. Roşu [Mes92, MR07, Ros15].
- Semantic of PLs
  - Small & Big step, Evaluation Contexts, Modular SOS, CHAM
  - Formalizations: C, Java, JS, P4, Smart Contracts...
- Executable: parser, interpreter, debugger
- Program verification

Working with the K Framework

- *Syntax* in BNF.
- (K) *Configuration*:
  - Represents elements of an abstract machine.
  - *Cells* holding syntax and data structures,
- *Computation* (the *k* cell):
  - Similar to a computation stack,
  - Sequential tasks with operator $\curvearrowright$,
  - Extends user syntax.
- *Rules* ($\Rightarrow$):
  - Pattern matching on the (K)Configuration,
  - Contextual, allows concurrent rewritings.

Heating and Cooling Rules

- Decomposition of *evaluation context* reduction.
- Example with left-to-right sum. Pattern matching on the k cell

$$
\begin{aligned}
e1 + e2 &\Rightarrow e1 \curvearrowright \square + e2 && \text{heat } e1\\
v1 \curvearrowright \square + e2 &\Rightarrow v1 + e2 && \text{cool } v1\\
v1 + e2 &\Rightarrow e2 \curvearrowright v1 + \square && \text{heat } e2\\
v2 \curvearrowright v1 + \square &\Rightarrow v1 + v2 && \text{cool } v2\\
v1 + v2 &\Rightarrow sum(v1, v2)
\end{aligned}
$$

- Standardized in K to model evaluation strategies.

K-CUE Syntax and (K)Configuration

Syntax

t ::=		Term
	c	Constant
	r	Reference
	$x : t$	Field
	$t \,.\, x$	Projection
	$\{\ ts\ \}$	Conjunction

```
syntax Term ::=
                Const
              | Lbl "^" Int
              | Lbl ":" Term
              | Term "." Lbl
              | "{" Terms "}"
```

(K)Configuration

$\langle K \rangle_k$
$\langle Map[Int \mapsto Map[Lbl \mapsto Term]] \rangle_{env}$
$\langle List[Lbl] \rangle_{path}$

```
configuration  // init. state
    <k> $PGM:Term </k>
    <env> 0 |-> (.Map) </env>
    <path> .List </path>
```

Example

tmp.cue

```
x: y: 1
p: {
    x
    z: x.y
}
```

krun tmp.cue --depth 16

```
<k> x^3 . y ~> z : []
~> [] ( x^1 ) ( z : x^3 . y ) ~> { [] } ~> p : []
~> [] ( p : { ... } ) ( x : y : 1 ) ~> { [] } ~> .
</k>
<env> 0 |-> ( x |-> y : 1
              p |-> { ( x^1 ) ( z : x^3 . y ) } )
      1 |-> ( z |-> x^3 . y )
      2 |-> ( .Map ) </env>
<path> z, p </path>
```

Heating and Cooling Subterms: Fields and Projections

$x : t \Rightarrow t \curvearrowright x : \square$

$t \curvearrowright x : \square \Rightarrow x : t$

```
// heat / cool a field
rule <k> (X:Lbl : T:Term) => (T ~> X : []) ...</k>
rule <k> (T:Term ~> X:Lbl : []) => (X : T) ...</k>
```

$t.x \Rightarrow t \curvearrowright \square.x$

$t \curvearrowright \square.x \Rightarrow t.x$

```
// heat / cool a projection
rule <k> (T:Term . X:Lbl) => (T ~> [] . X) ...</k>
rule <k> (T:Term ~> ([] . X)) => (T . X) ...</k>
```

Conjunctions and References

Upon cooling conjunctions, unify with `nu()` function.

```
// heat / cool all terms in conjunction
rule <k> { Ts:Terms } => Ts ~> { [] } ...</k>
rule <k> Ts:Terms ~> { [] } => { nu(Ts) } ...</k>

// heat / cool one term within a conjunction
rule <k> T:Term Ts:Terms => T ~> [] T Ts ...</k>
rule <k> T:Term ~> [] Ts:Terms => T Ts ...</k>
```

Resolve references by a lookup in `env` and lifting indices in the result.

```
rule <k> X :Lbl ^ D :Int => lift(D -Int 1, 1, T ) ...</k>
     <env>... (N -Int D ) |-> (( X |-> T ) _:Map)  ...</env>
```

Reference Resolution *is not* Replacement

1.

```
// input
x: x: a: 1
x
```

2.

```
// replace ‘x‘
x: x: a: 1
x: a: 1
```

3.

```
// unify ; output
x: {
  a: 1
  x: a: 1
}
```

Wrong!

References are Accumulative

1.

```
// input
x: x: a: 1
x
```

2.

```
// replace but keep ‘x‘ ; unify
x: {
  a: 1
  x: a: 1
}
x
```

3.

```
// replace but keep ‘x‘ ; unify
x: {
  a: 1
  x: a: 1
}
a: 1
x
```

Value = Fix point

See discussion #2731 here

Evaluation Token '@'

- Add a prefix token @ to the syntax.

```
syntax KItem ::= "@" Term
```

meaning "run one step of evaluation": resolve reference(s) once.

Heating rules apply on @-terms.

```
rule <k> @ X:Lbl : T:Term   => @ T ~> X : [] ...</k>
```

Resolving references ends the evaluation step of the current term.

```
rule <k> @ X:Lbl ^ D:Int =>  lift(D -Int 1, 1, T) ...</k>
     <env>... (N -Int D) |-> ((X |-> T) _:Map)  ...</env>
```

Cooling rules apply on @-free terms.

```
rule <k> T:Term ~> X:Lbl : [] => X : T  ...</k>
```

Breadth-first Evaluation Strategy

Add cell $\langle \mathit{Term} \rangle_{\mathit{fix}}$ and $\langle \mathit{String} \rangle_{\mathit{status}}$ to the (K)Configuration.
Fix point algorithm on the complete program.

```
rule <k> T:Term ~> . => @ T </k>
     <fix> . => T </fix>
     <status> "init" => "running" </status>

rule <k> T:Term ~> . => @ T </k>
     <fix> T0:Term => T </fix>
     <status> "running" </status>
  requires T =/=K T0

rule <k> T:Term ~> . => clean(T) </k>
     <fix> T0:Term => . </fix>
     <status> "running" => "stop" </status>
   requires T ==K T0
```

Cycles with Projections can be Hard

```
a: {
  b: {
    c: a
    d: c.e
  }.d
  e: 1
}
```

```
a.c: structural cycle:
    ./tmp.cue:4:8
```

No...

```
a: {
  b: {
    c: a & _
    d: c.e
  }.d
  e: 1
}
```

```
a: {
    b: 1
    e: 1
}
```

See issue #2476 here

Outline

1. Trees
2. CUE Core Semantics: K-CUE
3. More on CUE and Beyond

Built-in Functions and Predicates

Built-in functions, arithmetic

```
import ("strings", "math")
hi: "hello"
HI: strings.ToUpper(hi)
hiCalc: len(hi)*math.Sqrt(2)
```

```
hi:     "hello"
HI:     "HELLO"
hiCalc: 7.0710678118654755
```

Range simplification

```
range: int
range: <10
range: >=0 & <20

n: range
n: 8
```

```
range: uint & <10
n:     8
```

Disjunctions and Comprehensions

Disjunctions and hidden fields

```
_invRange: <0 | >10
n1: invRange & -1
// n2: invRange & 6
```

```
n1: -1
// n2: invalid value 6
//   (out of bound <0):
// n2: invalid value 6
//   (out of bound >10):
```

Comprehensions on lists

```
import "list"
ns: [ -8, 1, 2, 3, 42 ]
naturals: [
  for v in ns if v >= 0 { v }
]
sum: list.Sum(naturals)
```

```
ns: [ -8, 1, 2, 3, 42 ]
naturals: [1, 2, 3, 42]
sum: 48
```

More Comprehensions, `let` declaration and Interpolation

```
import "strings"
let fruits = {
  one: "apple"
  two: "pear"
}
invertedFruits: {
  for k, v in _fruits {
    "(v)": "\(k)\(k)"
  }
}
```

```
invertedFruits: {
  apple: "oneone"
  pear:  "twotwo"
}
```

More

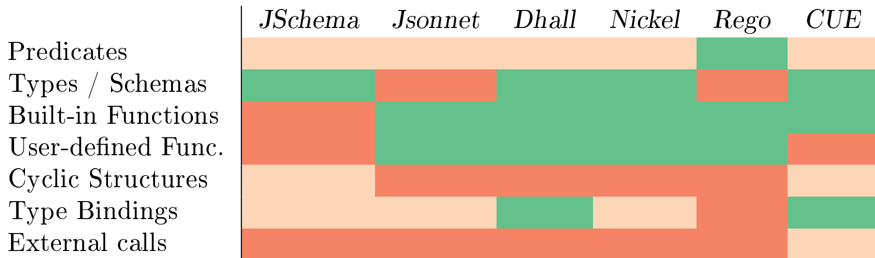
In the Language

- Default values
- Optional fields
- Closed structs
- Constraints on keys
- Regular expressions
- Packages
- *Call external Go/WASM*

Tooling

- Import and export
  YAML, *JsonSchema*, *OpenAPI*, *Protobuf*, *Go* structs
- Go library (*bindings*)
- Modules
- Predefined transformations
  `eval` | `def` | `vet` | `trim` | `export` | `fmt`
- Scripting layer

Language Capacities - Attempt at a Comparison



- Absent by choice or not implemented
- Limited
- Available by design choice

Some Open Questions

- Unsafe executions
- *Rego* $\cup$ *CUE*
- Logic variables
- Distribution and Concurrency

Resources

[CLR21] Xiaohong Chen, Dorel Lucanu, and Grigore Roşu. Matching logic explained. *Journal of Logical and Algebraic Methods in Programming*, 120:100638, April 2021.

[Mes92] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, April 1992.

[MR07] José Meseguer and Grigore Roşu. The rewriting logic semantics project. *Theoretical Computer Science*, 373(3):213–237, April 2007.

[Ros15] Grigore Rosu. From Rewriting Logic, to Programming Language Semantics, to Program Verification. In Narciso Martí-Oliet, Peter Csaba Ölveczky, and Carolyn Talcott, editors, *Logic, Rewriting, and Concurrency: Essays Dedicated to José Meseguer on the Occasion of His 65th Birthday*, Lecture Notes in Computer Science, pages 598–616. Springer International Publishing, Cham, 2015.

[Roş17] Grigore Roşu. Matching logic. *Logical Methods in Computer Science*, 13(4):1–61, December 2017.

- `https://matching-logic.org/`
- `https://kframework.org/`
- `https://cuelang.org/`
- CUE — A Type System for the Cloud - Craft Conference 2023 Link
- The Configuration Continuum: Using a Unified Model of Configuration to Prevent Outages - CONFLANG @ SPLASH 2023 Link

[1]

[1] This research was supported by the OTPaaS French project.

K-CUE Syntax

$t ::=$		Term
	c	Constant
	x	Reference
	$x : t$	Field
	$t \,.\, x$	Projection
	$\{\ ts\ \}$	Conjunction

$ts ::=$		List of terms
	$t \mid ts$	
$x ::=$	`\w(\w|\d)*`	Id (label)
$c ::=$		Constant
	$\top \mid \bot$	Top, Bottom
	`int` $\mid$ `bool` $\mid \cdots$	Type
	$1 \mid 2 \mid \cdots$	Ints
	`true` $\mid$ `false`	Bool

$h ::=$		Holed Term
	$x\ :\ \square$	
	$\square\ .\ x$	
	$\{\ \square\ \}$	
	$\square\ ts$	

$r ::= x\hat{}\,Int$	Indexed Ref.
$e ::=$ `@` t	Eval. Token

Requirements for Configuration Languages

Safety Guaranties

- Reproducibility
- No external dependencies
- Static type checking
- Validation, policy checking

Expressivity

- Composition, modularity
- Maintainability
- Schema definitions
- Expressions on values
- Value injection
- Integration with external ecosystem

Configuration Stack

Platform / Application

Validation Layer

- Testing, Mocks
- Code smell
- *Checkov*
- *Elektra*
- ...

Operational Composition

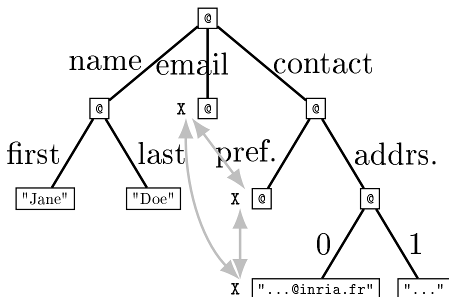
Reconfiguration?

- CI/CD
- Workflows
- *Dagger*
- *Concerto*
- ...

Configuration Language

- Data, Functions, Predicates, Schemas
- Externs, String templates, etc.

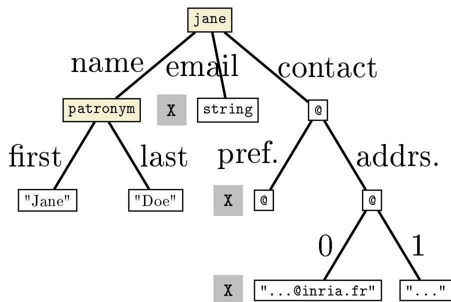
LIFE Variables



`X` *is a **coreferenced** variable*

```
@(
  name => @(
    first => "Jane",
    last => "Doe"
  ),
  email => X ,
  contact => @(
    preferred => X
    addresses => [
      X :"jane.doe@inria.fr",
      "jane.doe@imt-atlantique.fr"
    ]
  )
)?
```

LIFE Sorts : labels on the nodes – ψ-terms



: *Variable tags*
: User-defined *Sorts*

```
jane (
  name => patronym (
    first => "Jane",
    last => "Doe"
  ),
  email => X :string,
  contact => @(
    preferred => X
    addresses => [
      X :"jane.doe@inria.fr",
      "jane.doe@imt-atlantique.fr"
    ]
  )
)?
```

ψ-terms have 3 differentiated semantics

- As a *type*:
  `person(name => @, contact => contact(/* ... */))`
- As a *Function*:
  `get_addr(person(name => @, /* ... */), 1)`
- As a *Predicate*:
  `is_valid_addr(person(name => N, /* ... */))`

Comparison with LIFE's spectacles

	LIFE	CUE	Rego	Dhall	Nickel
Sorted data					
Sorted polymorphism					
Sort checking					
Nominal sorts					
Dependent sorts					
Functions, Predicates					