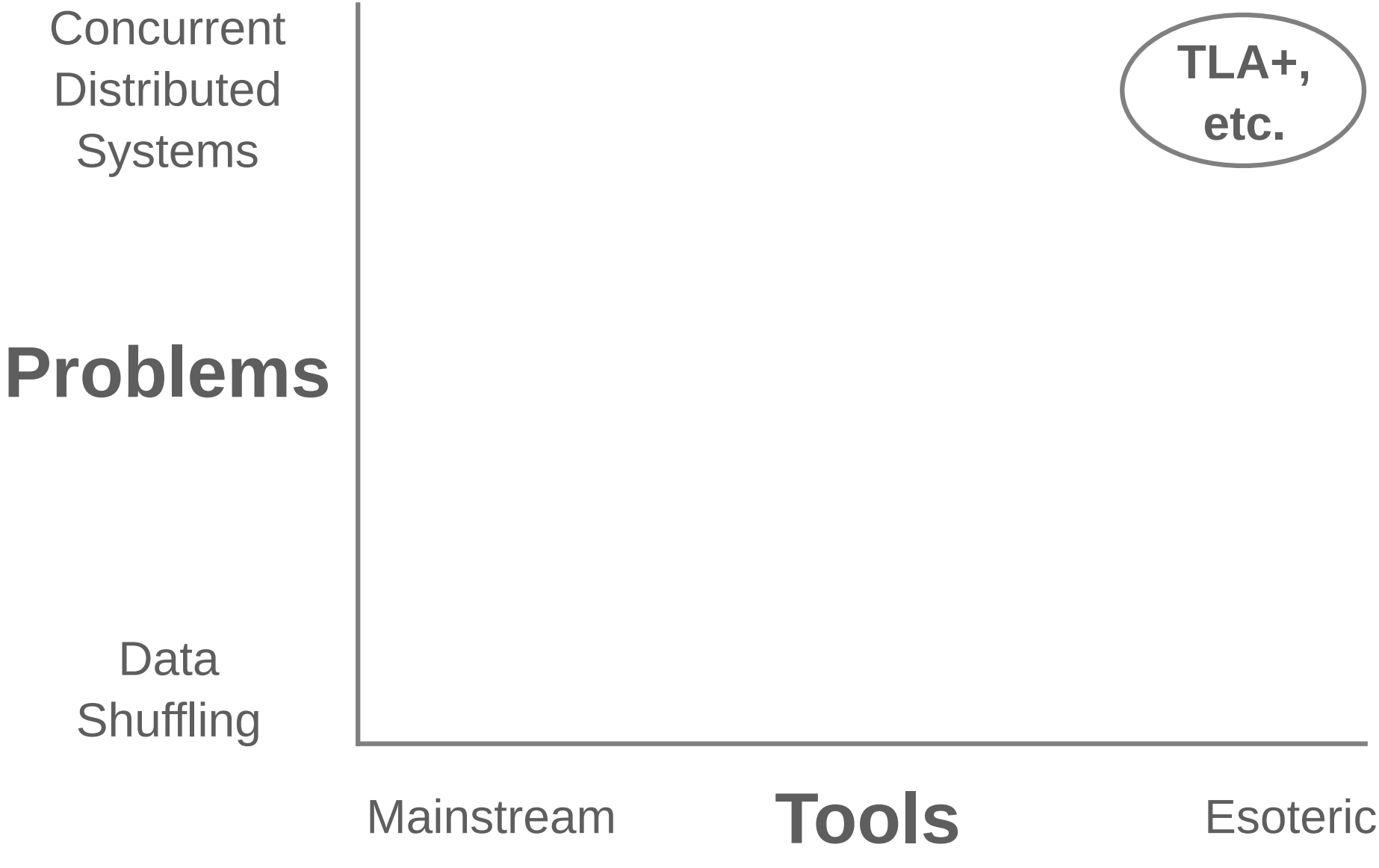
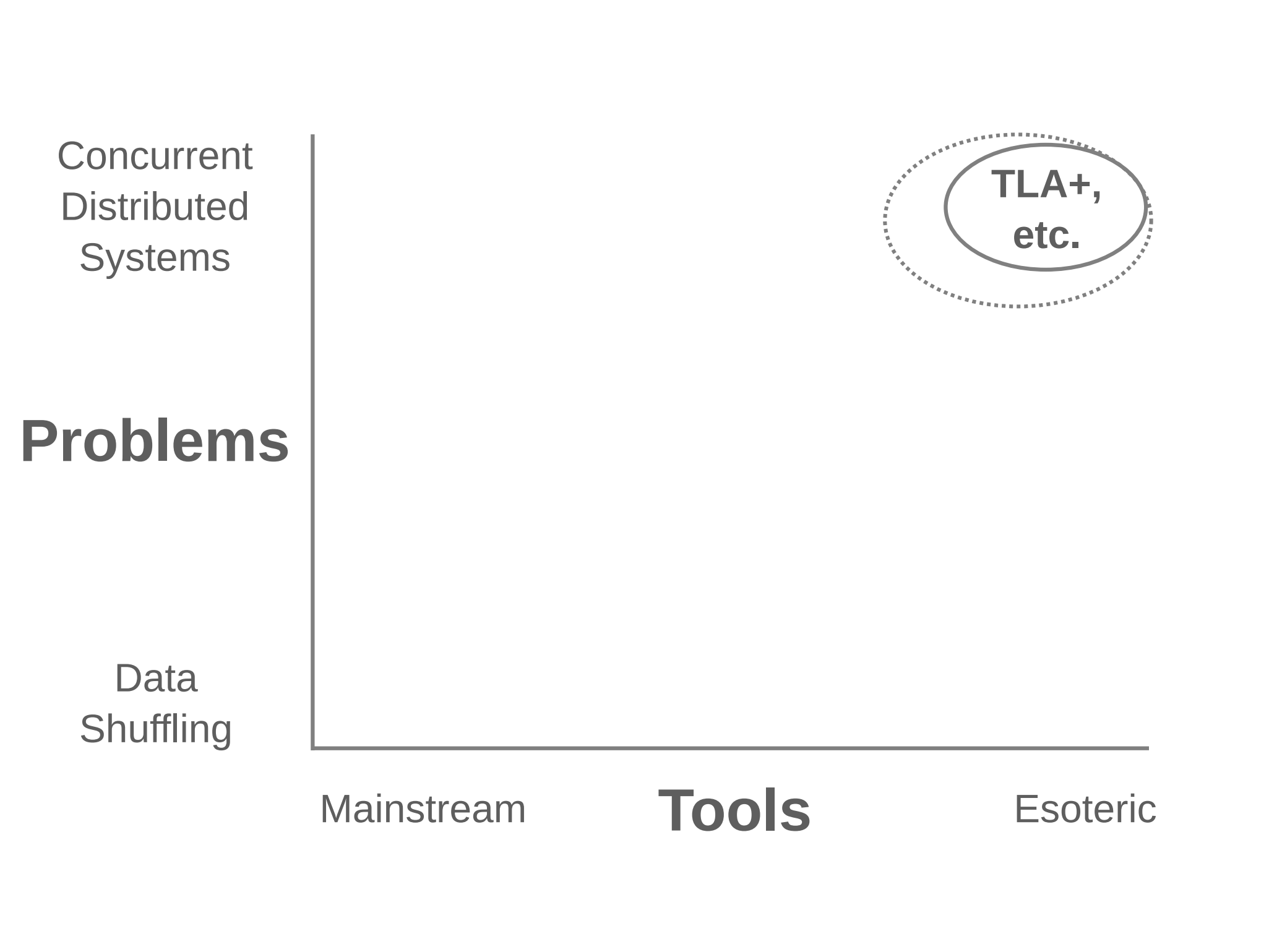
Formal Methods in the Enterprise

David McNeil

TLA+ Conf 2024
Monday, April 15, 2024
Seattle, USA


Concurrent
Distributed
Systems
Problems
Data
Shuffling
TLA+,
etc.
Mainstream
Tools
Esoteric


Concurrent
Distributed
Systems
Problems
Data
Shuffling
TLA+,
etc.
Mainstream
Tools
Esoteric

What is Halite?

Collaboration features

Enterprise collaboration

Developer workflows

Summary

halite

Public

forked from Viasat/halite

Pin Watch 0 Fork 0 Star 0

main 1 Branch 0 Tags Go to file t Code

This branch is up to date with `Viasat/halite:main`. Contribute Sync fork

david-mcneil and **Repo Extractor** bom-user: allow ranges to be 'open' on on… aa4923a · 7 months ago 1,272 Commits

doc	project.clj: upgrade clojure, upgrade libraries, trim some u…	8 months ago
resources/com/viasat	jadeite: move up a level in the namespace structure since …	2 years ago
src/com/viasat	bom-user: allow ranges to be 'open' on one end	7 months ago
test	analysis: bug fix for analyzing top-level 'every?' expressions	7 months ago
.gitignore	halite: typing/evaluation of literals	2 years ago
LICENSE	halite: typing/evaluation of literals	2 years ago
README.md	halite: typing/evaluation of literals	2 years ago
project.clj	upgrade to latest clojure	last year

README License

Halite

The successor to Salt.

Halite Documentation

Please note this is a pre-release version and the API is unstable.

About

Halite provides a LISP-like constraint expression language built on top of open source formal modelling tools such as the Choco constraint programming library

Readme

MIT license

Activity

0 stars

0 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

Clojure 100.0%

Halite specs have fields

```
{:spec/Dog$v4 {:fields {:age :Integer,
                        :colors [:Vec :String],
                        :name :String}}}
```

```
{:spec/Dog$v4 {:fields {:age :Integer,
                        :colors [:Vec :String],
                        :name :String}}}
```

Halite instances must structurally match specs

```
{:name "Rex",
 :$type :spec/Dog$v4,
 :age 3,
 :colors ["brown" "white"]}
```

Halite specs have constraints

```
{:tutorials.vending/State$v1
   {:fields {:balance [:Decimal 2],
             :beverageCount :Integer,
             :snackCount :Integer},
    :constraints #{'{:name "balance_not_negative",
                     :expr (>= balance #d "0.00")}
                   '{:name "counts_below_capacity",
                     :expr (and (<= beverageCount 20) (<= snackCount 20))}
                   '{:name "counts_not_negative",
                     :expr (and (>= beverageCount 0) (>= snackCount 0))}}}}
```

```
{:tutorials.vending/State$v1
   {:fields {:balance [:Decimal 2],
             :beverageCount :Integer,
             :snackCount :Integer},
    :constraints #{'{:name "balance_not_negative",
                     :expr (>= balance #d "0.00")}
                   '{:name "counts_below_capacity",
                     :expr (and (<= beverageCount 20) (<= snackCount 20))}
                   '{:name "counts_not_negative",
                     :expr (and (>= beverageCount 0) (>= snackCount 0))}}}}
```

Halite instances must satisfy spec constraints

```
{:$type :tutorials.vending/State$v1,
 :balance #d "0.00",
 :beverageCount 10,
 :snackCount 15}
```

Halite specs can model state machines

```
{:tutorials.vending/Transition$v1
   {:fields {:current :tutorials.vending/State$v1,
             :next :tutorials.vending/State$v1},
    :constraints
      #{'{:name "state_transitions",
          :expr
            (or
              (and (contains? #{#d "0.10" #d "0.25" #d "0.05"}
                              (- (get next :balance) (get current :balance)))
                   (= (get next :beverageCount) (get current :beverageCount))
                   (= (get next :snackCount) (get current :snackCount)))
              (and (= #d "0.50" (- (get current :balance) (get next :balance)))
                   (= (get next :beverageCount) (get current :beverageCount))
                   (= (get next :snackCount) (dec (get current :snackCount))))
              (and (= #d "1.00" (- (get current :balance) (get next :balance)))
                   (= (get next :beverageCount)
                      (dec (get current :beverageCount)))
                   (= (get next :snackCount) (get current :snackCount)))
              (= current next))}}}}
```

What is Halite?

Collaboration features

Enterprise collaboration

Developer workflows

Summary

Halite specs are versioned

```
{:spec/Dog$v4 {:fields {:age :Integer,
                        :colors [:Vec :String],
                        :name :String}}}
```

Halite specs are namespaced

```
{:tutorials.vending/State$v1
   {:fields {:balance [:Decimal 2],
             :beverageCount :Integer,
             :snackCount :Integer},
    :constraints #{'{:name "balance_not_negative",
                     :expr (>= balance #d "0.00")}
                   '{:name "counts_below_capacity",
                     :expr (and (<= beverageCount 20) (<= snackCount 20))}
                   '{:name "counts_not_negative",
                     :expr (and (>= beverageCount 0) (>= snackCount 0))}}}}
```

Halite specs can refine other specs

```
{:spec/A$v4 {:fields {:b :Integer,
                      :c :Integer,
                      :d :String},
             :refines-to {:spec/X$v4 {:name "refine_to_X",
                                      :expr '{:$type :spec/X$v4,
                                              :x (+ b c),
                                              :y 12,
                                              :z (if (= "medium" d) 5 10)}}}},
 :spec/X$v4 {:fields {:x :Integer,
                      :y :Integer,
                      :z :Integer}}}
```

```
{:spec/A$v4 {:fields {:b :Integer,
                      :c :Integer,
                      :d :String},
             :refines-to {:spec/X$v4 {:name "refine_to_X",
                                      :expr '{:$type :spec/X$v4,
                                              :x (+ b c),
                                              :y 12,
                                              :z (if (= "medium" d) 5 10)}}}},
 :spec/X$v4 {:fields {:x :Integer,
                      :y :Integer,
                      :z :Integer}}}
```

```
(let [a {:$type :spec/A$v4,
         :b 1,
         :c 2,
         :d "large"}]
  (refine-to a :spec/X$v4))


;-- result --
{:$type :spec/X$v4,
 :x 3,
 :y 12,
 :z 10}
```

Halite specs compose

```
{:spec/A$v1 {:fields {:b :spec/B$v1}},
 :spec/B$v1 {:fields {:c :Integer}}}
```

```
{:spec/A$v1 {:fields {:b :spec/B$v1}},
 :spec/B$v1 {:fields {:c :Integer}}}
```

Halite instances compose

```
{:$type :spec/A$v1,
 :b {:$type :spec/B$v1,
     :c 1}}
```

What is Halite?

Collaboration features

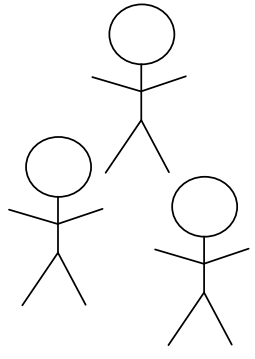
Enterprise collaboration

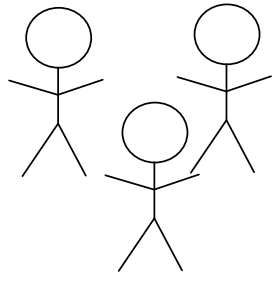
Developer workflows

Summary

Enterprise Collaboration

Hierachical namespace



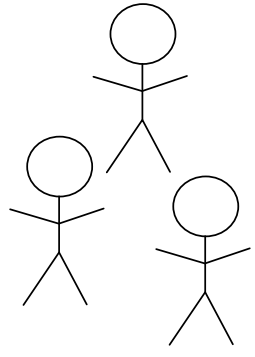


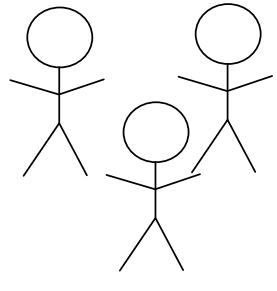
Enterprise Collaboration

Hierachical namespace

```
{:groupA/X$v1 ...}
```

immutable,
versioned specs



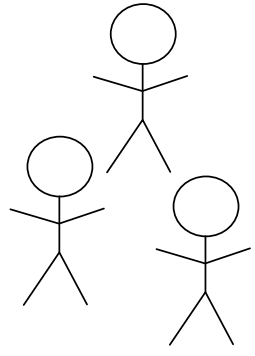


Enterprise Collaboration

Hierachical namespace

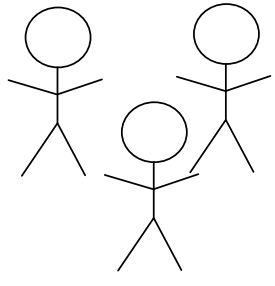
`{:groupA/X$v1 ...}`

immutable,
versioned specs



spec refinement
& composition

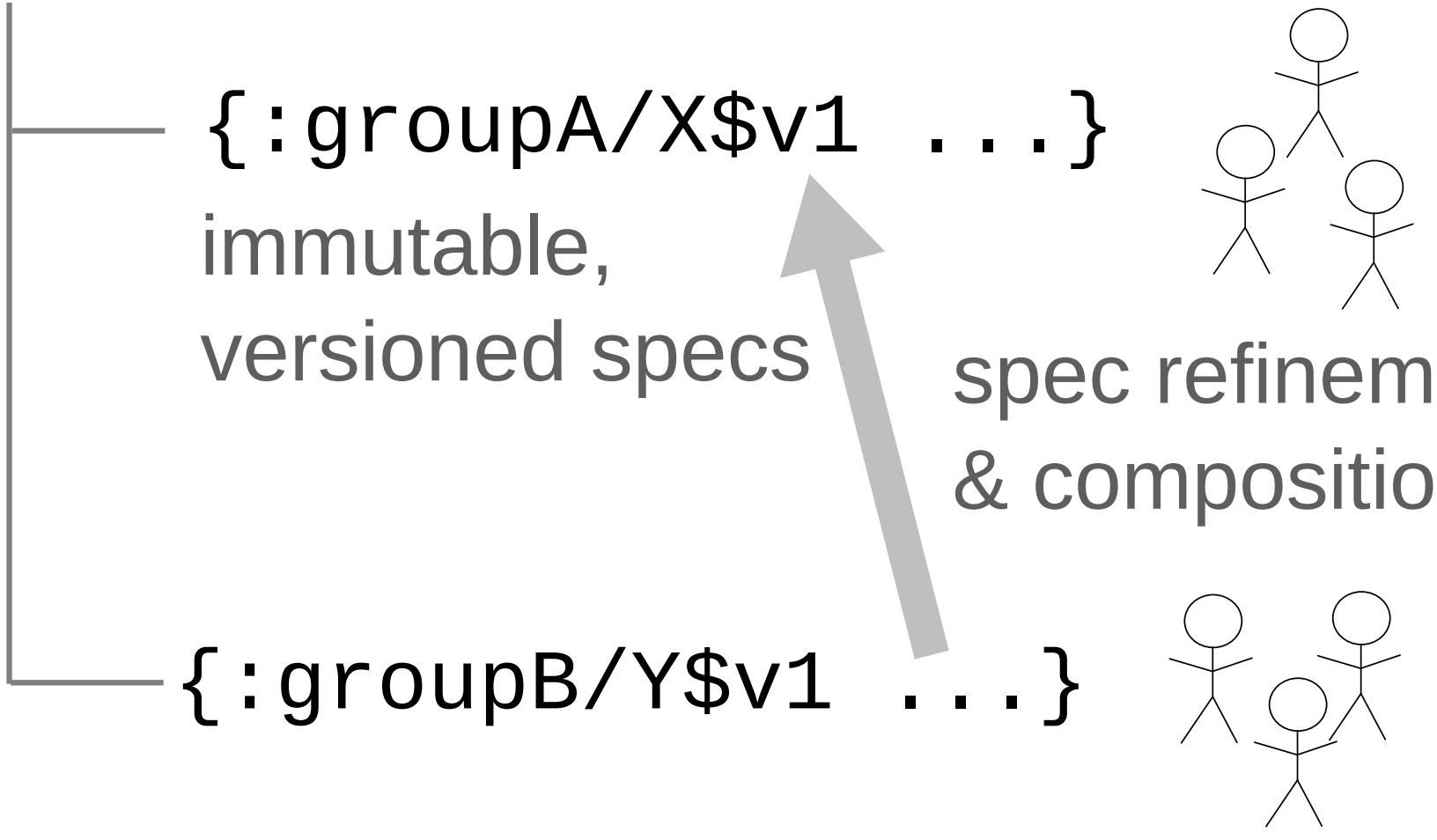
`{:groupB/Y$v1 ...}`



Enterprise Collaboration

Concurrent, distributed spec authoring

Hierachical namespace



What is Halite?
Collaboration features
Enterprise collaboration
Developer workflows
Summary

Halite expressions can be evaluated

```
(let [a {:$type :spec/A$v4,
         :b 1,
         :c 2,
         :d "large"}]
  (refine-to a :spec/X$v4))


;-- result --
{:$type :spec/X$v4,
 :x 3,
 :y 12,
 :z 10}
```

Literate Specs

How to model data fields in specifications.

It is possible to define a spec that does not have any fields.

```
{:spec/Dog$v1 {}}
```

Instances of this spec could be created as:

```
{:$type :spec/Dog$v1}
```

It is more interesting to define data fields on specs that define the structure of instances of the spec

```
{:spec/Dog$v2 {:fields {:age :Integer}}}
```

This spec can be instantiated as:

```
{:$type :spec/Dog$v2,
 :age 3}
```

A spec can have multiple fields

```
{:spec/Dog$v4 {:fields {:age :Integer,
                        :colors [:Vec :String],
                        :name :String}}}
```

```
{:name "Rex",
 :$type :spec/Dog$v4,
 :age 3,
 :colors ["brown" "white"]}
```

Literate Specs

A valid transition representing a dime being dropped into the machine.

```
{:$type :tutorials.vending/Transition$v1,
 :current {:$type :tutorials.vending/State$v1,
           :balance #d "0.00",
           :beverageCount 10,
           :snackCount 15},
 :next {:$type :tutorials.vending/State$v1,
        :balance #d "0.10",
        :beverageCount 10,
        :snackCount 15}}
```

An invalid transition, because the balance cannot increase by $0.07

```
{:$type :tutorials.vending/Transition$v1,
 :current {:$type :tutorials.vending/State$v1,
           :balance #d "0.00",
           :beverageCount 10,
           :snackCount 15},
 :next {:$type :tutorials.vending/State$v1,
        :balance #d "0.07",
        :beverageCount 10,
        :snackCount 15}}
```

```
;-- result --
[:throws
 "h-err/invalid-instance 0-0 : Invalid instance of 'tutorials.vending/Transition$v1', violates constraints \"tu
 :h-err/invalid-instance]
```

Constraint Propagation

```
(propagate {:ws/A$v1 {:fields {:x :Boolean,
                               :y :Boolean},
                      :constraints [["c1" '(if x true y)]]}}

           {:$instance-of :ws/A$v1,
            :x false})

;-- result --
{:$instance-of :ws/A$v1,
 :x false,
 :y true}
```

Search this site.

Documentation

Considerations
Getting Started
Modeling
Declaring variables
Handling constraints
Constraints over integer variables
Building expressions on integer variables
Constraints over set variables
Constraints over graph variables
Constraints over real variables
Solving
Lauching the resolution process
Dealing with solutions

Documentation / Modeling / Declaring variables

Declaring variables

How to declare variables?

A variable is an *unknown*, mathematically speaking. The goal of a resolution is to *assign* a *value* to each variable. The *domain* of a variable –(super)set of values it may take– must be defined in the model.

Choco-solver includes several types of variables: `BoolVar`, `IntVar`, `SetVar` and `RealVar`. Variables are created using the `Model` object. When creating a variable, the user can specify a name to help reading the output.

Integer variables

An integer variable is an unknown whose value should be an integer. Therefore, the domain of an integer variable is a set of integers (representing possible values). To create an integer variable, the `Model` should be used:

Java Python

```java
// Create a constant variable equal to 42
IntVar v0 = model.intVar("v0", 42);
// Create a variable taking its value in [1, 3] (the value is 1, 2 or 3)
IntVar v1 = model.intVar("v1", 1, 3);
// Create a variable taking its value in {1, 3} (the value is 1 or 3)
IntVar v2 = model.intVar("v2", new int[]{1, 3});
```

View page source
Edit this page
Create child page
Create documentation issue
Create project issue

Integer variables
Bounded domain
Enumerated domains
Boolean variables
Set variables
Graph Variables
Real variables
Views: Creating variables from constraints
Arithmetical views
Logical views
Composition
View over real variable

What is Halite?
Collaboration features
Enterprise collaboration
Developer workflows
Summary

Concurrent
Distributed
Systems

TLA+,
etc.

Enterprise coordination
Spec relationships
Immutable versions
Executable specs
Literate specs
Incremental adoption

Problems

Data
Shuffling

Mainstream **Tools** Esoteric

Thank you!