Conditional Rewriting Logic

From abstract model to concrete applications

Niccolò Piazzesi

Università degli studi di Pisa

February 11, 2022

Università di Pisa

Introduction and main concepts

What is a concurrent system?

Modeling concurrent system is one of the most studied problems in Computer Science.

Many proposed answers:

- Petri Nets
- CCS
- CSP
- Actors
- $\dots$

The need for unification

External fragmentation

Hard to relate different approaches, each with their own concepts, models and issues.

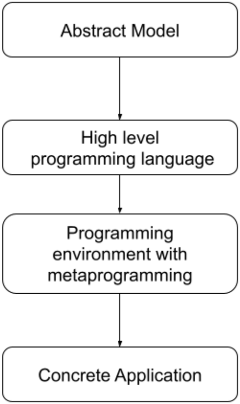
Internal fragmentation

Sometimes, fragmentation appears also within a specific approach (e.g. how can we unify operational and denotational semantics of CCS?).

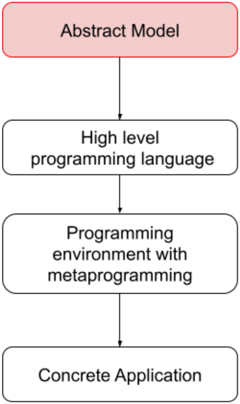
Concurrency in other areas

A related problem is the integration of concurrency with other paradigms (OO, Functional, ...) without using complex *ad hoc* solutions.

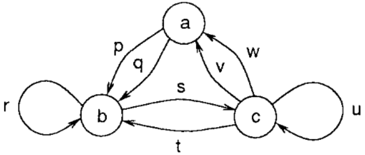
The strategy



The strategy



A first example




```
mod LTS is
    sort State .
    ops a,b,c : State .
    rl p : a => b .
    rl q : a => b .
    rl r : b => b .
    rl r : b => b .
    rl v : c => a .
    rl w : c => a .
    rl t : c => b .
    rl u : c => c .
endm
```

A labelled transition system and its code in Maude, a language based on rewriting logic.

Rewrite Theory

$$(\Sigma, E, L, R)$$

- Σ: ranked alphabet of function symbols
- E: set of Σ-equations
- L: set of *labels*
- R: set of pairs $R \subseteq L \times (T_{\Sigma,E}(\mathbf{X})^2)^+$

$T_{\Sigma,E}(\mathbf{X})$ denotes the set of E-equivalence classes of $\Sigma - terms$ with variables in $\mathbf{X}$.

Elements of R are called *rewrite rules*.

$$r : [t] \rightarrow [t'] \ if \ [u_1] \rightarrow [v_1] \wedge \cdots \wedge [u_n] \rightarrow [v_n]$$

Sequent entailment

$$\mathcal{R} \vdash [t] \to [t']$$

A rewrite theory $\mathcal{R}$ *entails* a *sequent* $[t] \to [t']$ *iff* $[t] \to [t']$ can be obtained by finite application of *rules of deduction*.

Rules of deduction

1. *Reflexivity*. For each $[t] \in T_{\Sigma,E}(\mathbf{X})$,

$$\frac{}{[t] \to [t]}$$

2. *Congruence*. For each $f \in \Sigma_n, n \in \mathbb{N}$,

$$\frac{[t_1] \to [t'_1] \;\; \cdots \;\; [t_n] \to [t'_n]}{[f(t_1, \cdots, t_n)] \to [f(t'_1, \cdots, t'_n)]}$$

3. *Replacement*. For each rule

$$\begin{aligned} r : [t(\overline{x})] \to [t'(\overline{x})] \; if \\ [u_1(\overline{x})] \to [v_1(\overline{x})] \wedge \cdots \wedge [u_n(\overline{x})] \to [v_n(\overline{x})] \end{aligned}$$

in R,

$$\frac{\begin{array}{ccc} [w_1] \to [w'_1] & \cdots & [w_n] \to [w'_n] \\ [u_1(\overline{w}/\overline{x})] \to [v_1(\overline{w}/\overline{x})] & \cdots & [u_k(\overline{w}/\overline{x})] \to [v_k(\overline{w}/\overline{x})] \end{array}}{[t(\overline{w}/\overline{x})] \to [t'(\overline{w}'/\overline{x})]}$$

4. *Transitivity*

$$\frac{[t_1] \to [t_2] \quad [t_2] \to [t_3]}{[t_1] \to [t_3]}$$

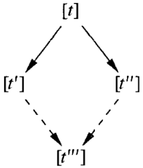
$\mathcal{R} = (\Sigma, E, L, R)$

A (Σ, E) sequent $[t] \to [t']$ is called a *concurrent $\mathcal{R}$-rewrite* iff it can be derived from $\mathcal{R}$ by finite application of rules (1)-(4).

$\mathcal{R}$ is *terminating* if there is no infinite chain of one step rewrites.

$[t']$ is an $\mathcal{R} - normal\ form$ of $[t]$ if $[t] \to [t']$ is an $\mathcal{R}$-rewrite and there is no $[t'] \to [t'']$ one-step $\mathcal{R}$-rewrite.

$\mathcal{R}$ is *Church-Rosser* or *confluent* if



Reflection

Rewriting logic is **reflective**.

There is a finitely presented rewrite theory $\mathbf{U}$ such that, for any finitely presented rewrite theory $\mathbf{T}$ (including $\mathbf{U}$ itself) we have the following equivalence:

$$T \vdash [t] \to [t'] \Longleftrightarrow \mathbf{U} \vdash \langle \overline{T}, \overline{[t]} \rangle \to \langle \overline{T}, \overline{[t']} \rangle$$

Since $\mathbf{U}$ is representable in itself, we can create a "reflective

tower" with any number of levels of reflection:

$$T \vdash [t] \to [t'] \Leftrightarrow \mathbf{U} \vdash \langle \overline{T}, \overline{[t]} \rangle \to \langle \overline{T}, \overline{[t']} \rangle \Leftrightarrow \mathbf{U} \vdash \langle \overline{\mathbf{U}}, \overline{\langle \overline{T}, \overline{[t]} \rangle} \rangle \to \langle \overline{\mathbf{U}}, \overline{\langle \overline{T}, \overline{[t']} \rangle} \rangle \cdots$$

A logic of action

Traditional view

Rewriting $\rightarrow$ series of one-step *sequential* rewrites.

Operational notion,
focus on the sequential computation itself.

Rewriting logic

Rewriting $\rightarrow$ *logical deduction.*

Concurrency implicit in the model.

A logic of action

Traditional view

Rewriting $\rightarrow$ series of one-step *sequential* rewrites.

Operational notion,
focus on the sequential computation itself.

Rewriting logic

Rewriting $\rightarrow$ *logical deduction.*

Concurrency implicit in the model.

Rewriting logic is a logic to reason about change in a concurrent system.

A term $[t]$ is a *proposition*, built up using the connectives in Σ, that asserts being in a certain ***state*** having a certain ***structure***.

Logic deduction $\Leftrightarrow$ **Concurrent computation**

Semantic model

The model

$$R = (\Sigma, E, L, R)$$

The model

$R = (\Sigma, E, L, R)$

We seek a category $\mathcal{T}_{\mathcal{R}}(\mathcal{X})$:

- Objects: $[t]_E$
- Arrows: $[t_1]_E \xrightarrow{\pi} [t_2]_E$

The model

$R = (\Sigma, E, L, R)$

We seek a category $\mathcal{T}_{\mathcal{R}}(\mathcal{X})$:

- Objects: $[t]_E$
- Arrows: $[t_1]_E \xrightarrow{\pi} [t_2]_E$

Idea: decorate sequents with terms representing proofs.

For simplicity we will consider the *unconditional* case.

Rules of generation

1. *Identities.* For each $[t] \in T_{\Sigma,E}(\mathbb{X})$,

$$\frac{}{[t] : [t] \to [t]}$$

2. $\Sigma - structure$. For each $f \in \Sigma_n, n \in \mathbb{N}$,

$$\frac{\alpha_1 : [t_1] \to [t_1'] \quad \cdots \quad \alpha_n : [t_n] \to [t_n']}{f(\alpha_1, \cdots, \alpha_n) : [f(t_1, \cdots, t_n)] \to [f(t_1', \cdots, t_n')]}$$

3. *Replacement.* For each rewrite rule $r : [t(\overline{x})] \to [t'(\overline{x})]$ in $\mathbb{R}$,

$$\frac{\alpha_1 : [w_1] \to [w_1'] \qquad \cdots \qquad \alpha_n : [w_n] \to [w_n']}{r(\alpha_1, \cdots, \alpha_n) : [t(\overline{w}/\overline{x})] \to [t'(\overline{w}'/\overline{x})]}$$

4. *Composition*

$$\frac{\alpha : [t_1] \to [t_2] \quad \beta : [t_2] \to [t_3]}{\alpha ; \beta : [t_1] \to [t_3]}$$

Each generation rule defines a different operation taking proof terms as arguments and returning a new proof term.

$\mathcal{P}_{\mathcal{R}}(X)$: graph generated by rules (1)-(4)

- Nodes: $T_{\Sigma,E}(X)$
- Arrows: $f \in \Sigma_n, n \in \mathbb{N}$, $\quad r \in R$, $\quad \_;\_$

$\mathcal{P}_{\mathcal{R}}(X)$ provides an *algebra of proof terms.*

Each generation rule defines a different operation taking proof terms as arguments and returning a new proof term.

$\mathcal{P}_{\mathcal{R}}(X)$: graph generated by rules (1)-(4)

- Nodes: $T_{\Sigma,E}(X)$
- Arrows: $f \in \Sigma_n, n \in \mathbb{N}$, $\quad r \in R$, $\quad \_;\_$

$\mathcal{P}_{\mathcal{R}}(X)$ provides an *algebra of proof terms.*

Still a syntactic structure, need a way to characterize "equal" proof terms.

Since proofs in Rewriting Logic corresponds to concurrent computations, what we are asking is:

When are two concurrent computations essentially the same?

$\mathcal{T}_{\mathcal{R}}(X)$

Given a rewrite theory $\mathcal{R}$, the model $\mathcal{T}_{\mathcal{R}}(X)$ is the quotient of $\mathcal{P}_{\mathcal{R}}(X)$ modulo the following equations:

1. *Category*.
   a. Associativity. For all α, β, γ, $(\alpha; \beta); \gamma = \alpha; (\beta; \gamma)$.
   b. Identities. For each $\alpha : [t] \to [t']$, $\alpha; [t'] = \alpha, \quad [t]; \alpha = \alpha$.
2. *Functoriality*.. For each $f \in \Sigma_n, n \in \mathbb{N}$,
   a. Preservation of composition. For all $\alpha_1, \cdots, \alpha_n, \beta_1, \cdots, \beta_n$, $f(\alpha_1; \beta_1, \cdots, \alpha_n; \beta_n) = f(\alpha_1, \cdots, \alpha_n); f(\beta_1, \cdots, \beta_n)$.
   b. Preservation of identities. $f([t_1], \cdots, [t_n]) = [f(t_1, \cdots, t_n)]$.
3. *Axioms* in E. For $t(x_1, \cdots, x_n) = t'(x_1, \cdots, x_n)$ an axiom in E, for all $\alpha_1, \cdots, \alpha_n, \quad t(\alpha_1, \cdots, \alpha_n) = t'(\alpha_1, \cdots, \alpha_n)$.
4. *Exchange*. For each rewrite rule $r : [t(\overline{x})] \to [t'(\overline{x})]$ in $\mathbb{R}$,

$$\frac{\alpha_1 : [w_1] \to [w_1'] \qquad \cdots \qquad \alpha_n : [w_n] \to [w_n']}{r(\overline{\alpha}) = r(\overline{[w]}); t'(\alpha) = t(\overline{\alpha}); r(\overline{[w']})}$$

Exchange law: Rewriting at the top using r and rewriting below using $\overline{\alpha}$ are independent processes that can be done in any order or simultaneously.

Since proof term describe concurrent computations, these equations provide an *equational theory* of **true concurrency**. We can tell when two different description describe the same ***abstract computation***.

Exchange law: Rewriting at the top using r and rewriting below using $\overline{\alpha}$ are independent processes that can be done in any order or simultaneously.

Since proof term describe concurrent computations, these equations provide an *equational theory* of **true concurrency**. We can tell when two different description describe the same ***abstract computation***.

$$
\begin{tikzcd}
\mathcal{T}_R(X)^n & \langle [w_1], \cdots, [w_n] \rangle \arrow[d, "\langle \alpha_1, \cdots, \alpha_n \rangle"] \\
& \langle [w'_1], \cdots, [w'_n] \rangle
\end{tikzcd}
\qquad
\begin{tikzcd}
{[t(\overline{w})]} \arrow[r, "r(\overline{w})"] \arrow[d, "{[t(\overline{\alpha})]}"'] \arrow[dr, dotted, "r(\overline{\alpha})"] & {[t'(\overline{w})]} \arrow[d, "{[t'(\overline{\alpha})]}"] & \mathcal{T}_R(X) \\
{[t(\overline{w}')]} \arrow[r, "r(\overline{w}')"'] & {[t'(\overline{w}')]} &
\end{tikzcd}
$$

Because of equation 2, terms $[t(\overline{x})]$ and $[t'(\overline{x})]$ can be regarded as functors $\mathcal{T}_{\mathcal{R}}(X)^n \to \mathcal{T}_{\mathcal{R}}(X)$.

A rule $r : [t(\overline{x})] \to [t'(\overline{x})]$ is a **natural transformation.**

$\mathcal{R}$-systems

Given a rewrite theory $\mathcal{R} = (\Sigma, E, L, R)$, an $\mathcal{R}$-***system*** $\mathcal{S}$ is a category $\mathcal{S}$ together with:

1. a family of functors $\{f_{\mathcal{S}} : \mathcal{S}^n \to \mathcal{S} | f \in \Sigma_n\}$ satisfying the equations in E.
2. for each rewrite rule $r : [t(\overline{x})] \to [t'(\overline{x})]$ in R, a natural transformation $r_{\mathcal{S}}$ from $t_{\mathcal{S}}$ to $t'_{\mathcal{S}}$.

An $\mathcal{R} - homomorphism\ F : \mathcal{S} \to \mathcal{S}'$ between two $\mathcal{R}$-systems is a functor $F : \mathcal{S} \to S'$ such that:

- it is a Σ-algebra homomorphism,i.e, $f_{\mathcal{S}} * F = F^n * f_{\mathcal{S}'}$ for each $f \in \Sigma_n$
- for each rewrite rule $r : [t(\overline{x})] \to [t'(\overline{x})]\ if\ C$ in r we have:

$$r_{\mathcal{S}} * F = F^n * r_{\mathcal{S}'}$$

$\mathcal{R}$-systems

Given a rewrite theory $\mathcal{R} = (\Sigma, E, L, R)$, an $\mathcal{R}$-***system*** $\mathcal{S}$ is a category $\mathcal{S}$ together with:

1. a family of functors $\{f_{\mathcal{S}} : \mathcal{S}^n \to \mathcal{S} | f \in \Sigma_n\}$ satisfying the equations in E.
2. for each rewrite rule $r : [t(\overline{x})] \to [t'(\overline{x})]$ in R, a natural transformation $r_{\mathcal{S}}$ from $t_{\mathcal{S}}$ to $t'_{\mathcal{S}}$.

An $\mathcal{R} - homomorphism\ F : \mathcal{S} \to \mathcal{S}'$ between two $\mathcal{R}$-systems is a functor $F : \mathcal{S} \to S'$ such that:

- it is a Σ-algebra homomorphism,i.e, $f_{\mathcal{S}} * F = F^n * f_{\mathcal{S}'}$ for each $f \in \Sigma_n$
- for each rewrite rule $r : [t(\overline{x})] \to [t'(\overline{x})]\ if\ C$ in r we have:

$$r_{\mathcal{S}} * F = F^n * r_{\mathcal{S}'}$$

$\mathcal{R}$-Sys: category of models for the rewrite theory $\mathcal{R}$.

$\mathcal{T}_{\mathcal{R}} \Rightarrow$ **Initial Object** in $\underline{\mathcal{R}\text{-Sys}}$

$\mathcal{T}_{\mathcal{R}}(X) \Rightarrow$ **Free Object** in $\underline{\mathcal{R}\text{-Sys}}$

Satisfaction relation:

$$\mathcal{S} \models [t(\overline{x})] \to [t'(\overline{x})] \quad \Rightarrow \quad \exists\ \alpha : t_{\mathcal{S}} \to t'_{\mathcal{S}}$$

Soundness: For $\mathcal{R}$ rewrite theory,

$$\mathcal{R} \vdash [t(\overline{x})] \to [t'(\overline{x})] \quad \Rightarrow \quad \mathcal{R} \models [t(\overline{x})] \to [t'(\overline{x})]$$

Completeness: For $\mathcal{R}$ rewrite theory,

$$\mathcal{R} \models [t(\overline{x})] \to [t'(\overline{x})] \quad \Rightarrow \quad \mathcal{R} \vdash [t(\overline{x})] \to [t'(\overline{x})]$$

All proven in [1].

2-category models

Lawvere: Algebras as functors.
$\mathbf{T} = (\Sigma, E)$ Σ-algebra A satisfying E
Given $[t(\overline{x})], A_t : A^n \to A$ extends to the product preserving functor

$$\hat{A} : \underline{\mathcal{L}_{\mathbf{T}}} \to \underline{Set}$$

$\underline{\mathcal{L}_{\mathbf{T}}}$

- Objects: natural numbers
- Arrows: $[t(\overline{x})] : n \to 1$
- Product: n is the product of 1 with itself n times and has projections $[x_1] \cdots [x_n]$.
- Composition given by substitution.

$$m \xrightarrow{([u_1],\cdots,[u_n])} n \xrightarrow{[t]} 1 \quad \Rightarrow \quad [t(\overline{u}/\overline{x})] : m \to 1$$

We define the category $\underline{Mod}(\underline{\mathcal{L}_{\mathbf{T}}}, \underline{Set})$ with objects functors like $\hat{A}$ and arrows natural transformation between them.

$$A \mapsto \hat{A} \quad \Rightarrow \quad \mathbf{Alg}_{\Sigma,E} \cong \underline{Mod}(\underline{\mathcal{L}_{\mathbf{T}}}, \underline{Set})$$

$\mathcal{L}_\mathcal{R}$

We can generalize the Lawvere constructions for rewriting logic by taking Cat instead of Set as the "ground" of existence.

$R = (\Sigma, E, L, R)$ $\quad$ $\mathcal{R}$-system $\mathcal{S}$.
Given a rule $r : [t] \to [t']$, the assignment to a natural transformation $r_\mathcal{S} : t_\mathcal{S} \to t'_\mathcal{S}$ extends naturally to a 2-product preserving 2-functor

$$\hat{\mathcal{S}} : \underline{\mathcal{L}_R} \to \underline{Cat}.$$

$\underline{\mathcal{L}_\mathcal{R}}$: • Objects: natural numbers • Arrows: $[t(\overline{x})] : n \to 1$

$\underline{\mathcal{L}_\mathcal{R}}(n, 1)$: • Objects: $[t(\overline{x})] : n \to 1$ • Arrows:$[\alpha] : [t(\overline{x})] \to [t'(\overline{x})]$

$$\begin{tikzcd}
m \arrow[r, bend left=50, "{([u_1],\ldots,[u_n])}"] \arrow[r, bend right=50, "{([v_1],\ldots,[v_n])}"'] \arrow[r, phantom, "\Downarrow {([\alpha_1],\ldots,[\alpha_n])}"] & n \arrow[r, bend left=50, "{[t]}"] \arrow[r, bend right=50, "{[t']}"'] \arrow[r, phantom, "\Downarrow {[\beta]}"] & 1
\end{tikzcd}$$

We define the category $\underline{Mod}(\underline{\mathcal{L}_\mathcal{R}}, \underline{Cat})$ as the category of canonical 2-product preserving 2-functors from $\underline{L_R}$ to $\underline{Cat}$.

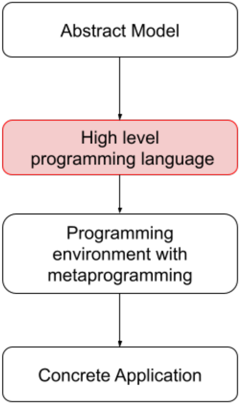
$$\mathcal{S} \mapsto \hat{\mathcal{S}} \quad \Rightarrow \quad \underline{\mathcal{R}\text{-Sys}} \cong \underline{Mod}(\underline{\mathcal{L}_\mathcal{R}}, \underline{Cat})$$

Computational view of $\mathcal{R}$-systems

$$\begin{array}{lcr}
System & \leftrightarrow & Category \\
State & \leftrightarrow & Object \\
Transition & \leftrightarrow & Morphism \\
Procedure & \leftrightarrow & Natural \quad transformation \\
Distributed\ Structure & \leftrightarrow & Algebraic\ Structure
\end{array}$$

Maude

The strategy



From model to programming language

Rewriting programming is not a new idea.

Implicit in algebraic simplification, present in various forms in many formalisms (pure Lisp,λ-calculus,...).

Traditional view:

- Semantics based on equational logic.
- *Functional interpretation*.
- Programming *algebras*.

New view:

- Semantics based on rewriting logic.
- Rewriting rules as concurrent state changes.
- Programming *concurrent systems.*

Program concurrent systems while maintaining a rigorous mathematical semantics.

Functional interpretation can still be maintained, simple and rigorous integration with other programming paradigms.

Maude: module semantics given in terms of an *initial machine* linking its operational and denotational semantics.

$$[\![\_]\!] : \mathcal{S} \to \mathcal{M}$$

What is Maude

Maude

- Programming language based on rewriting logic
  - Built by SRI international and University of Illinois
  - Includes a functional language as a sublanguage based on membership equational logic.

Maude is:

What is Maude

Maude

- Programming language based on rewriting logic
  - Built by SRI international and University of Illinois
  - Includes a functional language as a sublanguage based on membership equational logic.

Maude is:

A declarative programmming language.

What is Maude

Maude

- Programming language based on rewriting logic
  - Built by SRI international and University of Illinois
  - Includes a functional language as a sublanguage based on membership equational logic.

Maude is:

A declarative programmming language.

An executable formal specification language.

What is Maude

Maude

- Programming language based on rewriting logic
  - Built by SRI international and University of Illinois
  - Includes a functional language as a sublanguage based on membership equational logic.

Maude is:

A declarative programmming language.

An executable formal specification language.

A formal verification system.

Maude modules

```
fmod VENDING—MACHINE—SIGNATURE is
   sorts Coin Item Marking .
   subsorts Coin Item < Marking .
 op __ : Marking Marking —>
   Marking [assoc comm id: null] .
 op null : —> Marking .
 op $ : —> Coin [format (r! o)] .
 op q : —> Coin [format (r! o)] .
 op a : —> Item [format (b! o)] .
 op c : —> Item [format (b! o)] .
endfm
```

```
load vending—machine—
     signature.maude
mod VENDING—MACHINE is
    including VENDING—MACHINE—
         SIGNATURE .
    var M : Marking .
 rl [add—q] : M => M q .
 rl [add—$] : M => M $ .
 rl [buy—c] : $ => c .
 rl [buy—a] : $ => a q .
 rl [change] : q q q q => $ .
 endm
```

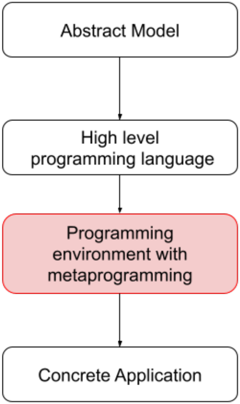
```
Maude> rew [3] $ $ q q . result Marking: $ $ $ q q q q

       frew[2] $ $ q q . result (sort not calculated): ($ q) ($ $) q q

       search [4, 10] $ q q q =>+ a c c M:Marking
           such that M:Marking =/= null .

       Solution 1 (state 108) M --> q q q q
           ...
```

The strategy



The meta level

Maude exploits the reflective nature of rewriting logic.

Every Maude *module* M can be represented as a *term* $\overline{M}$ of *Sort* **Module**.

$$t = f(a, g(b)) \quad \text{module FOO}$$
$$\Downarrow$$
$$\bar{t} =' f[\{'a\}\text{Foo},' g[\{'b\}\text{Foo}]]$$

The META-LEVEL module efficiently supports reflection, providing a number of functions to perform metalevel computation in the universal theory (*descent functions*).

```
op meta-apply : Module Term Qid Substitution MachineInt -> ResultPair .
op metaReduce : Module Term ~> ResultPair [special (...)] .
op metaRewrite : Module Term Bound ~> ResultPair [special (...)] .
op metaFrewrite : Module Term Bound Nat ~> ResultPair[special (...)] .
```

Metaprogramming with reflection

- Strategy languages.

  Execution strategies are internal to the language itself, one can easily program and define new strategies for a specific theory.

- Expressing languages and logics.

  Maude can be used to represent many languages and logics.

  Thanks to reflection, these representations can be *reified*, allowing programmers to execute them in Maude itself.

$$f : \mathbf{Module}_{\mathcal{L}} \to \mathbf{Module}$$

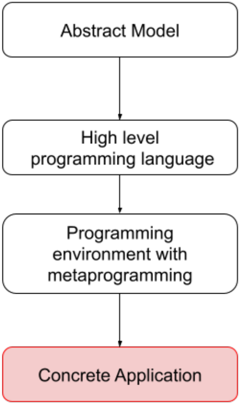
- Extending Core Maude.

  Reflection is key for language extensions such as **Full Maude** or **Real Time Maude**.

- All the tools for verification, reachability analysis, simulation...

Applications

The strategy



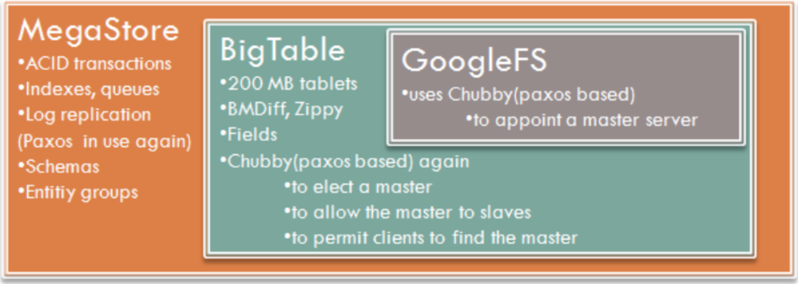
Applications

- Formalization of programming languages
  - C
  - Java, JVM
  - Scheme
  - K framework
  - ...
- Security: uncovering unknown attacks on web browsers
- Logical framework
  - Barendregt's lambda cube
  - Linear logic
  - Modal logic
  - ...
- Biology : Pathway logic
- Cloud transactions: Google's Megastore, Cassandra,...

For reference:
http://maude.cs.illinois.edu/w/index.php/Applications

Google Megastore

- Google's replicated data store.
- Billions of read/write daily transactions.
- Adds (limited) support for transactions in distributed data stores.



Google Megastore

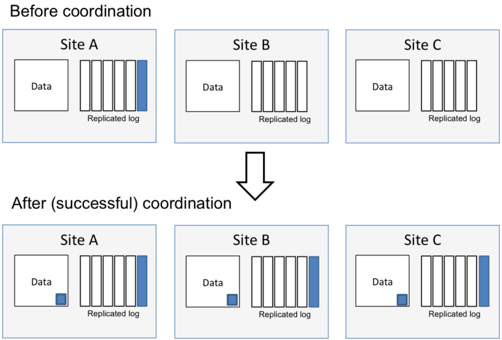
Data stored as key value pairs called *entities.*

Entity groups: set of entities. Each entity group is replicated at different *sites*.

Transaction: series of reads and writes on entities followed by a commit request.

A replicated *transaction log* is maintained for each entity group.

Megastore ensures atomicity and serializability of transactions accessing a single entity group.



Commit protocol

When a transaction t request a commit on site s:

1. s sends a *proposal*, containing a log entry and the next leader to the current site leader l. If another transaction is executing, t is aborted otherwise l sends the proposal to other sites.
2. s waits for an *ack* response from all sites. If some sites fail to respond, an *invalidate* message is sent instead.
3. When all sites acknowledge the proposal or the invalidate message, s requests them to apply t changes. Each site replicating the entity groups append the log entry to the local copy of the transaction log and then updates the local data store.

When some failures happens (a site goes down, messages are lost,...), a new site may propose itself.

MegaStore CGC

Work by Grov and Ölveczky:

- Formalized Google MegaStore in Real-Time Maude.
  - 56 rewrite rules (37 for fault tolerance).
  - Use of simulation and model checking to discover bugs during development and measure performance.
- Introduced MegaStore CGC, an extension that provides consistency for transactions accessing multiple entity groups.

Formal test driven development

1. Express requirements as LTL formulas
2. Develop Real Time Maude model
3. Test model through simulation and model checking
4. Analyse failures and modify the model

Formalizing MegaStore

```
class Site | entityGroups : Configuration, localTransactions : Configuration,
         coordinator : EntGroupLogPosPairSet, egOrderings :
              OrderClassUpdates,
         awaitingOrder : EntGroupUpdateList .

class EntityGroup | entitiesState : EntitySet, transactionLog : LogEntryList,
                 replicas : EntityGroupReplicaSet, proposals :
                      PaxosProposalSet,
                 pendingWrites : PendingWriteList .

class Transaction | operations : OperationList, status : TransStatus,
                 reads : EntitySet, readState : ReadStateSet,
                 writes : OperationList, paxosState : PaxosStateSet .
```

A rewrite rule

```
crl [initiateCommit] :
    < SID' : Site |
        entityGroups EGROUPS,
        localTransactions : LOCALTRANS
            < TID : Transaction | operations : emptyOpList,
                                  writes : WRITEOPS, status : idle
                                  readState : RSTATE, paxosState : PSTATE >
                                       >
    =>
    < SID : Site |
        localTransactions : LOCALTRANS
            < TID : Transaction | paxosState : NEW-PAXOS-STATE,
                                  status : in-paxos > >
    ACC-LEADER-REQ-MSGS
if EIDSET := getEntityGroupIds(WRITEOPS) /\
    NEW-PAXOS-STATE := initiatePaxosState(EIDSET, TID, WRITEOPS,
    SID, RSTATE, EGROUPS)
/\ (createAcceptLeaderMessages(SID, NEW-PAXOS-STATE)) => ACC-LEADER-
     REQ-MSGS
```

Megastore-CGC

Observation:

- A site participates in all updates involving the entity groups it replicates
- Implicit local ordering on these updates, we can make it explicit.

Idea: Keep an ordering list of the transactions accessing the set of entity groups that we want to keep consistent.

A transactions is validated only if its read set is consistent with the last update in the list.

S = set of sites. $R(s)$ = entity groups replicated at site s.

Ordering class oc = set of entity groups.

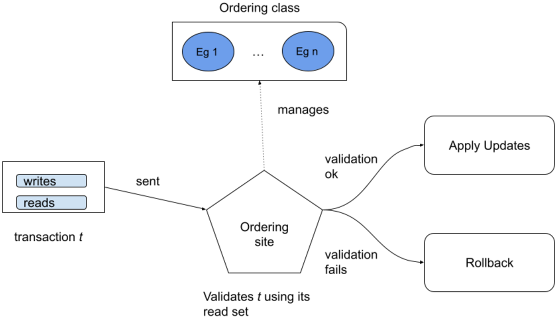
$$\forall oc \in OC\ \exists s \in S \text{ s.t } oc \subseteq R(s)$$

Ordering site os = site replicating all entity groups in oc.

$ol(oc)$: ordering list of oc

Megastore-CGC extends the normal commit protocol in the following way:

- In step 1, t is ordered when it is received by the ordering site os. It gets validated using its read set, which is included in the log entry proposal.
- If validation is succesful, the updated order is included in the apply phase of step 3.
- If the validation fails,step 3 is replaced by a rollback aborting t.



LTL requirements

System behaviour verification done via LTL model checking.

Desired Property:

```
<> [] (allTransFinished /\ entityGroupsEqualOrInvalid
/\ transLogsEqualOrInvalid /\ isSerializable)
```

All replicas are equal Property:

```
op entityGroupsEqualOrInvalid : -> Prop [ctor] .

ceq {< S1 : Site | coordinator : eglp(EG1, LP) ; EGLP,
               entityGroups :
               < EG1 : EntityGroup | entitiesState : ES1 > EGS1 >
    < S2 : Site | coordinator : eglp(EG1, LP) ; EGLP,
               entityGroups :
               < EG1 : EntityGroup | entitiesState : ES2 > EGS2 >
REST} |= entityGroupsEqual = false if ES1 =/= ES2 .

eq {SYSTEM} |= entityGroupsEqualOrInvalid = true [owise] .
```

Performance

Performance estimation done through Maude simulation, comparing MegaStore performance with Megastore-CGC.

(tfrew initState(10) in time <= 1000000 .)

Result:

{< stats(RSite): SiteStatistics | avgLatency : 94579/631, commitCount : 631, conflictAborts : 171, validationAborts : 10, ... > ... }

Analyzing results, almost no difference in performance was found between Megastore and Megastore-CGC.

	Megastore			**Megastore-CGC**			
	Comm.	Abs.	Avg.lat	Comm.	Abs.	Val.abs.	Avg.lat
Site 1	652	152	126	660	144	0	123
Site 2	704	100	118	674	115	15	118
RSite	640	172	151	631	171	10	150

This is also true for "Megastore-friendly" transactions, those accessing only one entity group.

	Megastore			**Megastore-CGC**			
	Comm.	Abs.	Avg.lat	Comm.	Abs.	Val.abs.	Avg.lat
Site 1	684	120	122	679	125	0	120
RSite	674	138	132	677	135	0	130
Site 2	693	111	110	691	113	0	113

Thank you for your attention!

References I

José Meseguer.
Conditional rewriting logic as a unified model of concurrency.
Theoretical computer science, 96(1):73–155, 1992.

José Meseguer.
Twenty years of rewriting logic.
The Journal of Logic and Algebraic Programming, 81(7-8):721–781, 2012.

Manuel Clavel, Francisco Durán, Steven Eker, Santiago Escobar, Patrick Lincoln, Narciso Martı-Oliet, José Meseguer, Rubén Rubio, and Carolyn Talcott.
Maude manual (version 3.1).
SRI International, 2020.

Jon Grov and Peter Csaba Ölveczky.
Increasing consistency in multi-site data stores: Megastore-cgc and its formal analysis.
In *International Conference on Software Engineering and Formal Methods*, pages 159–174. Springer, 2014.

Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, and José Meseguer.
Metalevel computation in maude.
Electronic Notes in Theoretical Computer Science, 15:331–352, 1998.

References III

Manuel Clavel, Fransisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Jose F Quesada.
Using maude.
In *International Conference on Fundamental Approaches to Software Engineering*, pages 371–374. Springer, 2000.

José Meseguer.
Research directions in rewriting logic.
In *Computational Logic*, pages 347–398. Springer, 1999.