Programming and symbolic computation in Maude

Francisco Durán, Steven Eker, Santiago Escobar,
Narciso Martí-Oliet, José Meseguer, Rubén Rubio,
and Carolyn Talcott

Presented by Paul Krogmeier

November 19, 2020

Tower of Hanoi

```
mod HANOI is
 protecting NAT-LIST .

 sorts Post Hanoi Game .
 subsort Post < Hanoi .

 op (_) [_] : Nat NatList -> Post [ctor] .
 op empty : -> Hanoi [ctor] .
 op __    : Hanoi Hanoi -> Hanoi [ctor assoc comm id: empty] .

 vars S T D1 D2 N : Nat .
 vars L1 L2    : NatList .
 vars H H'     : Hanoi .

 crl [move] : (S) [L1 D1] (T) [L2 D2] => (S) [L1] (T) [L2 D2 D1] if D2 > D1 .
 rl [move] : (S) [L1 D1] (T) [nil] => (S) [L1] (T) [D1] .
endm
```

Tower of Hanoi

```
mod HANOI is
 protecting NAT-LIST .

 sorts Post Hanoi Game .
 subsort Post < Hanoi .

 op (_) [_] : Nat NatList -> Post [ctor] .
 op empty : -> Hanoi [ctor] .
 op __    : Hanoi Hanoi -> Hanoi [ctor assoc comm id: empty] .

 vars S T D1 D2 N : Nat .
 vars L1 L2    : NatList .
 vars H H'     : Hanoi .

 crl [move] : (S) [L1 D1] (T) [L2 D2] => (S) [L1] (T) [L2 D2 D1] if D2 > D1 .
 rl [move] : (S) [L1 D1] (T) [nil] => (S) [L1] (T) [D1] .
endm
```

Demo with `rewrite`, `search`, and `srewrite`

Loop of Hanoi

```
Maude> srew (0)[3 2 1] (1)[nil] (2)[nil] using move ; move .

Solution 1
result Hanoi: (0)[3 2 1] (1)[nil] (2)[nil]

Solution 2
result Hanoi: (0)[3] (1)[1] (2)[2]

Solution 3
result Hanoi: (0)[3 2] (1)[nil] (2)[1]

Solution 4
result Hanoi: (0)[3] (1)[2] (2)[1]

Solution 5
result Hanoi: (0)[3 2] (1)[1] (2)[nil]

No more solutions.
```

Strategy expressions

Strategy expression: a transformation $\alpha : T_\Sigma \to \mathcal{P}(T_\Sigma)$

Strategy expressions

Strategy expression: a transformation $\alpha : T_\Sigma \to \mathcal{P}(T_\Sigma)$

To invoke a strategy:

```
srewrite (0)[3 2 1] (1)[nil] (2)[nil]
  using <Strategy expression> .
```

Strategy expressions

Strategy expression: a transformation $\alpha : T_\Sigma \to \mathcal{P}(T_\Sigma)$

To invoke a strategy:

```
srewrite (0)[3 2 1] (1)[nil] (2)[nil]
  using <Strategy expression> .
```

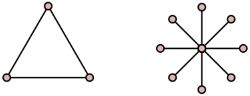
Define complex strategies from simple ones with combinators

Combinators and informal semantics

Strategy ζ	Results $[\![\zeta]\!](\theta,t)$
`idle`	$\{t\}$
`fail`	$\emptyset$
$rlabel[\rho]$	$\{t' \in T_\Sigma \mid t \rightarrow_{\rho(l)\rightarrow\rho(r)} t'$ for any $l \rightarrow^{rlabel} r \in R\}$
$\alpha ; \beta$	$\bigcup_{t' \in [\![\alpha]\!](\theta,t)} [\![\beta]\!](\theta,t')$
$\alpha \mid \beta$	$[\![\alpha]\!](\theta,t) \cup [\![\beta]\!](\theta,t)$
$\alpha *$	$\bigcup_{n=0}^{\infty} [\![\alpha]\!]^n(\theta,t)$
`match` P `s.t.` C	$\begin{cases} \{t\} & \text{if matches}(P,t,C,\theta) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$
$\alpha ? \beta : \gamma$	$\begin{cases} [\![\alpha ; \beta]\!](\theta,t) & \text{if } [\![\alpha]\!](\theta,t) \neq \emptyset \\ [\![\gamma]\!](\theta,t) & \text{if } [\![\alpha]\!](\theta,t) = \emptyset \end{cases}$
`matchrew` P `s.t.` C `by` X_1 `using` α_1, ..., X_n `using` α_n	$\bigcup_{\sigma \in \text{matches}(P,t,C,\theta)} \Big(\bigcup_{t_1 \in [\![\alpha_1]\!](\sigma,\sigma(X_1))} \cdots \bigcup_{t_n \in [\![\alpha_n]\!](\sigma,\sigma(X_n))} \sigma[x_1/t_1, \ldots, x_n/t_n](P) \Big)$
$slabel(t_1,\ldots,t_n)$	$\bigcup_{(lhs,\delta,C) \in \text{Defs}} \bigcup_{\sigma \in \text{matches}(slabel(t_1,\ldots,t_n),lhs,C,\text{id})} [\![\delta]\!](\sigma,t)$

Strategizing search for logical separators

Given

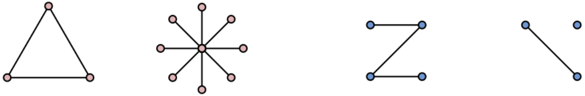


(a) Positive examples X

(b) Negative examples Y

Strategizing search for logical separators

Given

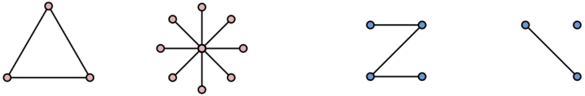


(a) Positive examples X (b) Negative examples Y

Find sentence φ that separates X from Y

Strategizing search for logical separators

Given



(a) Positive examples X (b) Negative examples Y

Find sentence φ that separates X from Y

$\exists x \forall y. E(x, y)$

Strategizing search for logical separators

Imagine a soup of tagged first-order formulas:

Strategizing search for logical separators

Imagine a soup of tagged first-order formulas:

$$\langle\, \varphi_1 \mid (\mathcal{M}, A_1) \mid \mathsf{True} \,\rangle$$
$$\langle\, \varphi_2 \mid (\mathcal{M}, A_2) \mid \mathsf{True} \,\rangle$$

Strategizing search for logical separators

Imagine a soup of tagged first-order formulas:

$$\langle\, \varphi_1 \mid (\mathcal{M}, A_1) \mid \mathsf{True}\,\rangle$$
$$\langle\, \varphi_2 \mid (\mathcal{M}, A_2) \mid \mathsf{True}\,\rangle$$

$$\Downarrow$$

$$\langle\, \varphi_1 \wedge \varphi_2 \mid (\mathcal{M}, A_1 \cap A_2) \mid \mathsf{True}\,\rangle$$

Strategizing search for logical separators

Imagine a soup of tagged first-order formulas:

$$\langle\, \varphi_1 \mid (\mathcal{M}, A_1) \mid \mathsf{True} \,\rangle$$
$$\langle\, \varphi_2 \mid (\mathcal{M}, A_2) \mid \mathsf{True} \,\rangle$$

$$\Downarrow$$

$$\langle\, \varphi_1 \wedge \varphi_2 \mid (\mathcal{M}, A_1 \cap A_2) \mid \mathsf{True} \,\rangle$$

Large state space

Strategizing search for logical separators

Imagine a soup of tagged first-order formulas:

$$\langle\, \varphi_1 \mid (\mathcal{M}, A_1) \mid \mathsf{True} \,\rangle$$
$$\langle\, \varphi_2 \mid (\mathcal{M}, A_2) \mid \mathsf{True} \,\rangle$$

$$\Downarrow$$

$$\langle\, \varphi_1 \wedge \varphi_2 \mid (\mathcal{M}, A_1 \cap A_2) \mid \mathsf{True} \,\rangle$$

Large state space
Can search in restricted class

Key points

Key points

Maude can ensure rewriting respects a user-given strategy

Key points

Maude can ensure rewriting respects a user-given strategy

Explore very large state spaces

Key points

Maude can ensure rewriting respects a user-given strategy

- Explore very large state spaces
- Avoid tampering with clean rewrite theories

Key points

Maude can ensure rewriting respects a user-given strategy

- Explore very large state spaces
- Avoid tampering with clean rewrite theories

I'm using the strategy language to find logical separators

Thank you

References I

[1] Durán, F., Eker, S., Escobar, S., Martí-Oliet, N., Meseguer, J., Rubio, R., and Talcott, C. Programming and symbolic computation in maude. *Journal of Logical and Algebraic Methods in Programming 110* (2020), 100497.