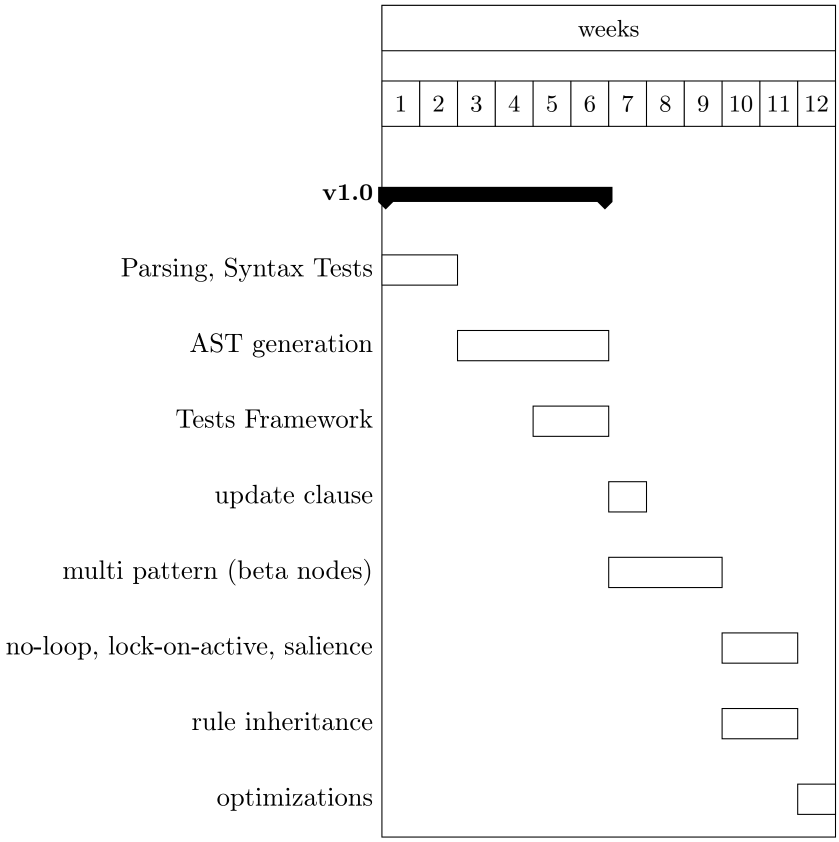
Lity rule engine spec v1.0

mingc@skymizer.com

June 26, 2018

Abstract

This document describes Lity rule engine v1.0 features. For Drools semantics, please refer to [1]. For the Rete algorithm, please consults [2].



Part I
Usage

1 Definitions

Fact A struct that can be pattern matched when `fireAllRules()` is called.

Working Memory A set of fact. All facts in working memory will be pattern matched when `fireAllRules()` is called. Lity compiler will create internal state variables to track the state of working memory.

2 grammar

The grammar extends solidity's grammar.

Note that `StringLiteral`, `Identifier`, `Expression`, `Statement` are defined in solidity's grammar file.

```
Rule = 'rule' StringLiteral 'when' '{' RuleLHS '}' 'then' '{' RuleRHS '}'
RuleLHS = ( ( Identifier ':' )? FactMatchExpr ';' )
FactMatchExpr = Identifier '(' ( FieldExpr (',' FieldExpr)* )? ')'
FieldExpr = ( Identifier ':' Identifier ) | Expression
RuleRHS = Statement*
```

Listing 1: Rule definition grammar.

Note that we don't support empty or multiple statement in `RuleLHS` at v1.0.

3 Internal functions exposed to user

3.1 Working memory manipulation functions

To manipulate working memory. These functions will be provided to user.

```
/// @return wm_index the fact inserted
///         wm stands for Working Memory
function factInsert(FactType1 storage f) internal returns (uint256);
function factInsert(FactType2 storage f) internal returns (uint256);
function factInsert(FactTypeN storage f) internal returns (uint256);
// ...

/// removes a fact from working memory
function factDelete(uint256 wm_index) internal;
```

Listing 2: Internal working memory manipulation functions.

For each struct type `T_i` defined in the contract, we generate a corresponding `factInsert(T_i f)` implementation. The structs are passed into these function using storage data location (pass-by-reference semantic), this means that only references of facts are stored in working memory.

We can use `factDelete` and `factInsert` to simulate `factUpdate`.

3.2 fireAllRules()

`fireAllRules()` is another internal function exposed to users. When users want to fire all rules with current facts in working memory, they can call this function.

4 Usage Example

```
contract C {
    struct Person {
        int age;
        bool eligible;
        address addr;
    }
    rule "sendMoneyToAdult"
    when {
        p: Person(age >= 18, eligible == true);
    } then {
        p.addr.send(10);
        p.eligible = false;
    }
    mapping (address => uint256) addr2idx;
    Person[] ps;
    function addPerson(int a) {
        ps.push(Person(a, true, msg.sender));
        addr2idx[msg.sender] = factInsert(ps[ps.length-1]);
    }
    function deletePerson() {
        factDelete(addr2idx[msg.sender]);
    }
    function pay() {
        fireAllRules();
    }
}
```

Listing 3: A Lity example with rule engine.

Part II
Implementation

5 Working memory manipulation functions implementation

```
mapping(uint256 => uint256) wm_index_to_fact_type_id;
mapping(uint256 => FactType1) fact_table_1;
mapping(uint256 => FactType2) fact_table_2;
// ...
mapping(uint256 => FactTypeN) fact_table_N;
uint256 first_unused_working_memory_index = 1;

function factInsert(FactType1 storage f) internal returns (uint256 assigned_index) {
   wm_index_to_fact_type_id[first_unused_working_memory_index] = 1;
   fact_table_1[first_unused_working_memory_index] = f;
   assigned_index = first_unused_working_memory_index;
   first_unused_working_memory_index += 1;
}

function factDelete(uint256 wm_index) internal {
    uint256 table_index = wm_index_to_fact_type_id[wm_index];
    assert(table_index != 0);
    wm_index_to_fact_type_id[wm_index] = 0;
}
```

Listing 4: Working memory manipulation function implementations

Note that state variables in Listing 4 should not be visible to user.

6 fireAllRules() implementation

Our approach is similar to Rete algorithm. But much simpler.

7 Transform Rule ASTs to Rete Network

For each `FieldExpr`, we generate an Alpha Node. Alpha nodes generated by the same rule will form a chain.

```
Person[] facts_Person;

function fireAllRules() internal {
    // update facts_Person
    collect_Person();
    do {
        rule1();
        for (uint256 i = 0; i < rule1_ret.length; i++) {
            execute_rule1(facts_Person[rule1_ret[i][0]]);
        }
        // if there are other rules...:
        // rule2();
        // for (uint256 i ...)
    } while (false);
}

function collect_Person() internal {
    delete facts_Person;
    for (uint256 i = 0; i < first_unused_wm_index; i++) {
        if (wm_index_to_fact_type_id[i] != 1)
            continue;
        facts_Person.push(fact_table_1[i]);
    }
}

uint256[1][] rule1_ret;
function rule1() internal {
    alpha2();
    rule1_ret = alpha2_ret;
}

uint256[1][] alpha2_ret;
function alpha2() internal {
    alpha1();
    delete alpha2_ret;
    for (uint256 i = 0; i < alpha1_ret.length; i++){
        if (facts_Person[alpha1_ret[i][0]].eligible == true)
            alpha2_ret.push([i]);
    }
}

uint256[1][] alpha1_ret;
function alpha1() internal {
    // Type Node reached
    delete alpha1_ret;
    for (uint256 i = 0; i < facts_Person.length; i++)
        if (facts_Person[i].age >= 18)
            alpha1_ret.push([i]);
}

function execute_rule1(Person storage p) internal {
    p.addr.send(10);
    p.eligible = false;
}
```

Listing 5: The code structure

Listing 5 implements the usage example in listing 3.

References

[1] M. Salatino, M. D. Maio, and E. Aliverti, *Mastering JBoss Drools 6 for Developers.* Packt Publishing, 2016.

[2] C. L. Forgy, “On the Efficient Implementation of Production Systems.” PhD Thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1979.