Specification of the ReLog Language

ReLog is a logical programming language which aims to support efficient reprogramming for wireless sensor network applications. The design of ReLog leverages on the observation that a WSN application program can be represented as a composition of data processing and system operations. Data processing focuses on dealing with data according to the application business logic and system operations are responsible for the inputs and outputs of data processing. The logical programming language has shown its potential to deal with data processing. Additionally, logical programs are naturally compact and easy to modify. Therefore, ReLog uses *facts* and *rules* from the traditional logical programming language to express data processing, and extends with *events* and *actions* to deal with system operations.

1 Grammar

In this section, we introduce grammar of the ReLog language in the form of extended **BNF**. We first give some conventions as follows.

- Items existing 0 or more times are enclosed in curly brackets or suffixed with an asterisk, such as {item} or item*, respectively.
- Items existing 1 or more times are suffixed with an addition (plus) symbol, e.g., item$^+$.
- Items existing 0 or 1 times are enclosed in square brackets, e.g., [item].
- *Terminals* appear in italics and **non-terminals** appear in bold.

1.1 Program

A typical ReLog program consists of four parts including *configuration definition*, *predicate declaration*, *clause definition*, and *shell definition*.

Program ::= **ConfigurationDef PredicateDec ClauseDef ShellDef**

1.2 Configuration Definition

To increase flexibility, ReLog allows users to specify some configurations through optional annotations. For example, the annotation '# sys_dutyCycle = 10' indicates that the sensor needs set their duty cycle to 10%.

Annotations used for specifying configurations should be at the beginning of the program. In the current version, ReLog allows users to set duty cycle, communication channel, transmission power, and choose time synchronization service, data collection protocol, and data dissemination protocol.

ConfigurationDef ::= **Configuration***

Configuration ::= **OctothorpeSym ConfigurationPara IsSym ConfigurationVal**

ConfigurationKey ::= **DutyCycleKey** | **ChannleKey** | **TxPowerKey** | **TimeSyncKey**
| **CollectionKey** | **DisseminationKey**

ConfigurationVal ::= **Integer** | **BuildInMethod**

1.3 Predicate Declaration

ReLog requires users to declare predicates explicitly. Each declaration consists of a predicate name and optional attributes. Due to memory limitation in sensors, the out-of-date facts should be cleared from time to time. We observe that most of the facts (i.e., application data) have predicable lifetime. Therefore, ReLog allows users to specify predefined data management policies through attributes.

Particularly, the *unique* attribute indicates that a fact will be deleted if a new fact of the same predicate comes. The *volatile : timer* attribute indicates that the facts of the predicate will be deleted if the associated timer fires.

PredicateDec ::= **PredicatePrompt Predicate**$^+$

PredicatePrompt ::= **PredicateKey ColonSym**

Predicate ::= **PredicateName [DataAttribute] DotSym**

DataAttribute ::= **AtSym AttributeName {CommaSym AttributeName}**

AttributeName ::= **UniqueKey** | **VolatileKey ColonSym TimerName**

1.4 Clause Definition

ReLog uses facts and rules to deal with data processing in WSN applications.

ClauseDef ::= **ClausePrompt Clause**$^+$

ClausePrompt ::= **ClauseKey ColonSym**

Clause ::= **Fact DotSym** | **Rule DotSym**

1.4.1 Fact

Facts are used to represent the application data. A fact is an atom with all parameters of constants. For example, the fact $address(sys\_nodeID)$ represents that the address of a sensor in the application is the sensor's ID.

Fact ::= **PredicateName LParenSym Constants RParenSym**

Constants ::= **Constant {CommaSym Constant}**

Constant ::= **Integer** | **Float** | **RootKey** | **InfinityKey** | **NodeIdKey** | **BroadcastKey**
| **TrueKey** | **FalseKey** | ε

1.4.2 Rule

Rules are used to implement data calculation. A rule consists of a deduction symbol (:-), an atom on the left side of the symbol called the head, and one or more atoms on the right side called the body. To facilitating programming, ReLog allows users to use relational expressions as body atoms. The body defines some preconditions, which if true, instantiates the head to a fact. For example, the rule

$$message(Src, Value) : -reading(Value), address(Src), Value > 100$$

creates a fact of *message* if there exist the facts of *reading* and *address* and the value of the *reading* fact is greater than 100.

Note that parameters of the head atom could be arithmetic expressions, while parameters of body atoms could be variables only. Meanwhile, a predicate in the body of a rule may contain the @*passive* attribute, which indicates that the arrival of new facts of this predicate will not trigger the evaluation of the rule.

Rule ::= **HeadAtom IfSym BodyAtoms**

HeadAtom ::= **PredicateName LParenSym ArithmeticExps RParenSym**

ArithmeticExps ::= **ArithmeticExp {CommaSym ArithmeticExp}**

ArithmeticExp ::= **ArithmeticTerm TermOp ArithmeticExp**

TermOp ::= **PlusOp** | **MinusOp**

ArithmeticTerm ::= **ArithmeticFactor FactorOp ArithmeticTerm**

FactOp ::= **MultiOp** | **DivOp**

ArithmeticFactor ::= **Variable** | **Constant** | **LParenSym ArithmeticExp RParenSym**
| **Function** | **OneArrayOp Variable**

Function ::= **FunctionName LParenSym ArithmeticExps RParenSym**

OneArrayOp ::= **IncreOp** | **DecreOp**

BodyAtoms ::= **BodyAtom {CommaSym BodyAtom}**

BodyAtom ::= **PredicateName [ExeAttribute]LParenSym Variables RParenSym**
|**RelationExp**

ExeAttribute ::= **AtSym PassiveKey**

Variables ::= **Variable {CommaSym Variable}**

RelationExp ::= **LParameter RelationOp RParameter**

LParameter ::= **Variable**

RParameter ::= **Variable** | **Integer** | **Float**

RelationOp ::= **GrtOp** | **GeqOp** | **LssOp** | **LeqOp** | **EqOp** | **NeqOp**

1.5 Shell Definition

The ReLog language uses event and action to handle system operations. Each statement in the shell part consists of an event and one or more actions that appear on the left and right of the statement's triggering symbol ($\rightarrow$), respectively. The action(s) will be triggered to execute if the associated event occurs. For example, the statement

$$generate(message(Src, Value)) -> send(1, < Src, Value >)$$

indicates that if a new fact of *message* arrives, the content of the fact will be sent to the base station. Each variable in the action will be initialized with the value of the variable with the same name in the event.

ShellDef ::= **ShellPrompt ShellStatement**$^+$

ShellPrompt ::= **ShellKey ColonSym**

ShellSatement ::= **Event TriggerSym Actions**

1.5.1 Event

There are four types of events in the ReLog language.

- The *boot* event occurs when the execution starts. This event facilitates the initialization of the execution.
- The *timer* event occurs when the associated timer expires. The timer's name is used to differentiate different timer events.
- The *fact generation* event occurs when a new fact of the predicate in the event is generated. The variables of the atom in the event will be initialized with the constants in the newly generated fact.
- The *message receiving* event occurs when a message is received. This message consists of two parts. The first part is the message type which is used to differentiate different types of messages, while the second part indicates the payload of the received message.

Event ::= **BootEvent** | **TimerEvent** | **FactGenEvent** | **MessageReceiveEvent**

BootEvent ::= **BootKey LParenSym RParenSym**

TimerEvent ::= **TimerName LParenSym RParenSym**

FactGenEvent ::= **FactGenKey LParenSym EventAtom RParenSym**

EventAtom ::= **PredicateName LParenSym Variables RParenSym**

MessageReceiveEvent ::= **MessageReceiveKey LParenSym Message RParenSym**

Message ::= **MessageType CommaSym Payload**

MessageType ::= **Integer**

Payload ::= **LssOp Variables GrtOp**

1.5.2 Action

Various actions are provided for users to program WSN applications. These actions can be roughly divided into two classes.

- Actions in the first class manipulate devices of timer and radio. These actions could be used to set timer, transmission power, communication channel, and duty cycle, and send messages. For each function, the ReLog language may provides multiple actions to facilitate the programming.
- Actions in the second class manipulate the fact repository of a program. These actions are used to insert facts to or delete facts from the fact repository.

Note that ReLog uses functions rather than actions to manipulate sensing devices and serial ports (e.g., ADC, GPIO, and I2C). This is because data from these devices and ports could only be used to generate facts. Making these functions be parameters of fact repository manipulating actions can make ReLog programs more compact.

Actions ::= **Action {CommaSym Action}**

Action ::= **SetTimerAction** | **SendMsgAction** | **SetTxPowerAction** | **SetChannelAction**
| **SetDutyCycleAction** | **InsertFactAction** | **DeleteFactAction**

SetTimerAction ::= **SetTimerActionName LParenSym TimerParameters RParenSym**

SetTimerActionName ::= **SetTimerKey** | **SetTimerMilliKey**

TimerParameters ::= **TimerName CommaSym Interval CommaSym Count**

Interval ::= **Integer** | **RandomFunc**

RandomFunc ::= **RandomFuncName LParenSym Bound CommaSym Bound RParenSym**

Bound ::= **Integer**

Count ::= **Integer** | **InfinityKey**

SendMsgAction ::= **SendKey LParenSym MsgParameters RParenSym**

MsgParameters ::= [**Address**] **CommaSym Message**

Address ::= **Integer** | **BoradcastKey**

SetTxPowerAction ::= **SetTxPowerKey LParenSym Integer RParenSym**

SetChannelAction ::= **SetChannelKey LParenSym Integer RParenSym**

SetDutyCycleAction ::= **SetDutyCycleKey LParenSym Integer RParenSym**

InsertFactAction ::= **InsertKey ActionAtom**

ActionAtom ::= **PredicateName LParenSym AtomParameters RParenSym**

AtomParameters ::= **AtomParameter {CommaSym AtomParameter}**

AtomParameter ::= **Variable** | **Constant** | **SenseFunc**

SenseFunc ::= **SenseFuncName LParenSym portName RParenSym**

portName ::= **ThermometerKey** | **PhotometerKey** | **ADC0Key** | **ADC1Key**
| **ADC2Key** | **ADC3Key** | **ADC6Key** | **ADC7Key** | **GPIO2Key**
| **GPIO3Key** | **I2CKey**

DeleteFactAction ::= **DeleteKey LParenSym PredicateName RParenSym**

2 Lexical Tokens

In this section, we introduce lexical tokens of the ReLog language.

2.1 Prompts

ReLog uses keywords of *Predicate*, *Clause*, and *Shell* to start the predicate declaration, the clause definition, and the shell definition, respectively.

PredicateKey ::= *Predicate*

ClauseKey ::= *Clause*

ShellKey ::= *Shell*

2.2 Attributes

ReLog uses keywords of *unique*, *volatile*, and *passive* to represent the unique attribute, the volatile attribute, and the passive attribute, respectively.

UniqueKey ::= *unique*

VolatileKey ::= *volatile*

PassiveKey ::= *passive*

2.3 Configuration Parameters

Parameters of configurations all start with the symbol '#' and have the same prefix of '*sys_*'.

DutyCycleKey ::= *sys_dutyCycle*

ChannleKey ::= *sys_channel*

TxPowerKey ::= *sys_TxPower*

TimeSyncKey ::= *sys_timeSync*

CollectionKey ::= *sys_collection*

DisseminationKey ::= *sys_dissemination*

2.4 Events and Actions

ReLog uses keywords of *boot*, *generate*, and *receive* to represent the boot event, the fact generation event, and the message receiving event, respectively.

Keywords of *setTimer* and *setTimerMilli* are used to represent timer manipulating actions. Keywords of *send*, *setTxPower*, *setChannel*, and *setChannel* are used to represent radio manipulating actions. Keywords of *insert* and *delete* are used to represent fact repository manipulating actions.

BootKey ::= *boot*

FactGenKey ::= *generate*

MessageReceiveKey ::= *receive*

SetTimerKey ::= *setTimer*

SetTimerMilliKey ::= *setTimerMilli*

SendKey ::= *send*

SetTxPowerKey ::= *sendTxPower*

SetChannelKey ::= *setChannel*

SetDutyCycleKey ::= *setDucyCycle*

InsertKey ::= *insert*

DeleteKey ::= *delete*

2.5 Sensing devices and Serial Ports

ReLog uses keywords of *sys_thermometer* and *sys_thermometer* to represent the thermometer and the photometer device, respectively. It also uses keywords of *sys_ADCn*, *sys_GPIOn*, and *I2C* to represent serial ports. These keywords all have the same prefix of '*sys_*'.

ThermometerKey ::= *sys_thermometer*

PhotometerKey ::= *sys_photometer*

ADC0Key ::= *sys_ADC0*

ADC1Key ::= *sys_ADC1*

ADC2Key ::= *sys_ADC2*

ADC3Key ::= *sys_ADC3*

ADC6Key ::= *sys_ADC6*

ADC7Key ::= *sys_ADC7*

GPIO2Key ::= *sys_GPIO2*

GPIO3Key ::= *sys_GPIO3*

I2CKey ::= *sys_I2C*

2.6 Symbols

Symbols in ReLog include prompt symbols, delimiter symbols, and special symbols (:- and ->).

OctothorpeSym ::= #

ColonSym ::= :

CommaSym ::= ,

DotSym ::= .

AtSym ::= @

IfSym ::= :-

TriggerSym ::= ->

LParenSym ::= (

RParenSym ::=)

2.7 Indentifiers

Identifiers in ReLog includes predicate names, function names, timer names, and variables. Predicate name are syntactically identifiers beginning with lower-case letters. Function name are syntactically predicate names with the same prefix of '_'. Timer names syntactically equal to predicate names. Variables are syntactically identifiers beginning with upper-case letters.

PredicateName ::= **LowerCaseLetter String**

FunctionName ::= **Underscore LowerCaseLetter String**

RandomFuncName ::= *_random*

SenseFuncName ::= *_sense*

TimerName ::= **LowerCaseLetter String**

Variable ::= **UpperCaseLetter String**

String ::= **Digit String** | **Letter String** | **Underscore String** | ε

Letter ::= **LowerCaseLetter** | **UpperCaseLetter**

LowerCaseLetter ::= *a* | ... | *z*

UpperCaseLetter ::= *A* | ... | *Z*

Underscore ::= _

Digit ::= 0 | ... | 9

2.8 Constants

Constants in ReLog include (32-bit) integers, floating numbers, logical constants, and system-defined constants. System-defined constants all have the same prefix of '*sys_*'.

Integer ::= /*32-bit integer, such as '0', '1234', '-5678'. We omit the definition here. */

Float ::= /*Real number in forms of floating point and scientific notation, such as '12.34', '0.5678E3'. We omit the definition here. */

RootKey ::= *sys_root*

InifinityKey ::= *sys_infinity*

NodeIdKey ::= *sys_nodeId*

BroadcastKey ::= *sys_broadcast*

TrueKey ::= *true*

FalseKey ::= *false*

2.9 Operators

Operators in ReLog include arithmetical operators and relational operators.

PlusOp ::= +

MinusOp ::= -

MultiOp ::= *

DivOp ::= /

IncreOp ::= ++

DecreOp ::= –

GrtOp ::= >

GeqOp ::= >=

LssOp ::= <

LeqOp ::= <=

EqOp ::= ==

NeqOp ::= !=