AMIDOL IR

Contents

Formal Definition

Formally, the IR is a 5-tuple,

$$(S, E, L, \Phi, \Lambda, \Delta)$$

where: *

$$S$$

is a finite set of state variables

$$\{s_0, s_1, \ldots, s_{n-1}\}$$

that take on values in

$$\mathbb{N}$$

. *

$$E$$

is a finite set of events

$$\{e_0, e_1, \ldots, e_{m-1}\}$$

that may occur in the model. *

$$L : S \mid E \to \mathbb{N}$$

is the event and state variable labeling function that maps elements of

$$S$$

and

$$E$$

into the original ontology. *

$$\Phi : E \times N_0 \times N_1 \times \ldots \times N_{n-1} \to \{0, 1\}$$

is the event enabling predicate. *

$$\Lambda : E \times N_0 \times N_1 \times \ldots \times N_{n-1} \to (0, \infty)$$

is the transition rate specification. *

$$\Delta : E \times N_0 \times N_1 \times \ldots \times N_{n-1} \to N_0 \times N_1 \times \ldots \times N_{n-1}$$

is the state variable transition function specification.

Informally the IR represents a model using a universal and Turing-complete mathematical language using a formalism based on Generalized Stochastic Petri-nets with inhibitor arcs. Instead of inhibitor arcs, we utilize the more intuitive and compact method of allowing events to have input predicates

$$\Phi$$

which can be evaluated to determine if an event is enabled, and output predicates

$$\Delta$$

which define the side effects of event firing.

Intuitively, the *marking* of a model in the IR given as

$$N_0 \times N_1 \times \ldots \times N_{n-1}$$

is the set values of each state variable. We typically do not represent the entire marking when giving the definition of enabling conditions (input predicates), transition rates associated with an event, or state variable transition function (output predicates), and instead only indicate the markings on which an given input predicate depends, and those state variables whose values change when an event fires due to the output predicate, omitting the "don't care" values.

State Variables

State variables are defined in the IR as objects which have a name, a set of semantic labels, a state variable type (such as int or float), and an initial value given as an expression.

```
{ "state_variable": {
    "name": ["string"],
    "labels": ["string"],
    "type": "sv_type",
    "initial_value": "expression"
  }
}
```

Intuitively, the set of state variables in a model define the current state. While state variables are defined as taking on values in

$$\mathbb{N}$$

in the formal definition, this does not restrict them from representing real numbers to arbitrary precision in modern computer hardware. In practice, they are implemented as integers, and floating point numbers by solvers.

Events

Events are defined in the IR as objects which have a name, a set of semantic labels, a rate given as an expression, and an input predicate and output predicate, defined either as objects, or undefined. In the cases where the input predicate is undefined, it is interpreted as being always true. In the case where the output predicate is undefined, the event is interpreted as having no effect on the state of the model when it fires.

```
{ "event": {
    "name": ["string"],
    "labels": ["string"],
    "rate": "expression",
    "input_predicate": {},
    "output_predicate": {},
  }
}
```

Events can be interpreted in two ways: discrete and continuous. In both cases events define the ways in which a model can change state by altering the value of state variables. In the discrete case, events fire at discrete times defined by their rates, changing the value of the model at those times as defined by their output predicates. In the continuous case, events define flow relations which alter the state in proportion to their output predicates scaled by their rates.

Currently in AMIDOL, all rates are considered exponential. AMIDOL can be extended at a later date to account for general distributions.

Input Predicates

```
{ "input_predicate": {
    "enabling_condition": "boolean_expression"
  }
}
```

Output Predicates

The transition function is specified as a partially defined state change vector

```
{ "output_predicate": {
    "transition_function": [{"sv_name": "string", "function": "expression"}]
  }
}
```

Output predicates define the state change vector associated with event firing through partial definition as a list of state variable names, followed by an expression

$$e$$

indicating their change when the event is fired under the discrete interpretation of events. Given an event with rate

$$\mu$$

the continuous interpretation of event firing utilizes a scaled version of this expression given by

$$\mu \cdot e$$

.

$$\frac{dS}{dt} = -\frac{\beta IS}{N}$$

$$\frac{dI}{dt} = \frac{\beta IS}{N} - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

For example, the “infect” event of an SIR model given above would have rate

$$\beta \frac{SI}{N}$$

, and the partial state transition function [{“S”: -1}, {“I”: 1}].

Names of State Variables and Events

In order to support composition via the operations of state sharing, and event sharing, the AMIDOL IR represents names of state variables and events as lists of strings. The strings in this list must be unique to the individual state variable or event, or the model is invalid and an exception should be raised. For example:

```
{ "state_variable": { "name": ["S"], "labels": [], "type": "int",
    "initial_value": "0"}},
{ "state_variable": { "name": ["I", "Infected"], "labels": [], "type": "int",
    "initial_value": "0"}}
```

Is a valid model, the strings in each name list are unique. But the following example should have an exception raised:

```
{ "state_variable": { "name": ["S"], "labels": [], "type": "int",
    "initial_value": "0"}},
{ "state_variable": { "name": ["I", "S"], "labels": [], "type": "int",
    "initial_value": "0"}}
```

as the string "S" is shared by the name lists of both state variables. Names are lists to allow for easy name resolution when composing two models. Given the following Models:

Model A

```
{ "state_variable": { "name": ["S"], "labels": [], "type": "int",
    "initial_value": "0"}}
```

Model B

```
{ "state_variable": { "name": ["Susceptible"], "labels": [], "type": "int",
    "initial_value": "0"}}
```

Composing A and B on `["S", "Susceptible"]` results in the model:

Model AcB

```
{ "state_variable": { "name": ["S", "Susceptible"], "labels": [], "type": "int",
    "initial_value": "0"}}
```

Any events in Model A which referred to "S" are treated as if they refer to the new shared state variable with name `["S", "Susceptible"]`.

Labels

Elements of the IR have labels associated with them, used to propagate domain knowledge from the formulation in a VDSOL to the mathematical elements of the IR.

Constants

Constants in the IR allow the definition of identifiers which refer to literal values, mostly useful for setting model wide scaling factors, initial conditions, etc.

```
{ "constant": {
    "name": "string",
    "value": "literal"
  }
}
```

Expressions

Expressions in AMIDOL are defined by a simple syntax over state variable names, constant names, and literal constants.

```
expression ::= term "+" expression | term "-" expression |
               term
term       ::= "(" expression ")" | term "*" expression | term "/" expression |
               atom
atom       ::= identifier | literal
identifier ::= sv_name | constant_name
literal    ::= integer | float
```

Boolean Expressions

Input predicates utilize boolean expressions, which are defined by the following syntax which builds on the syntax given above.

```
boolean_expression ::= "(" boolean_expression ")" | "NOT" boolean_expression |
                       boolean_expression "AND" boolean_expression |
                       boolean_expression "OR" boolean_expression | boolean_term
boolean_term ::= expression relation expression | "TRUE" | "FALSE"
```

Reward Variables

The AMIDOL IR allows the specifications of reward variables as a partial definition of a model, and the composition of these reward variables with other models to define measures of interest which can be solved by the executable translation of a total model in the IR. Given a model

$$M = (S, E, L, \Phi, \Lambda, \Delta)$$

we define two basic types of rewards structures, rewards over state variable values (rate rewards), and rewards over events (impulse rewards).

Rate Rewards

A rate reward is formally defined as a function

$$\mathcal{R} : P(S, \mathbb{N}) \to \mathbb{R}$$

where

$$q \in P(S, \mathbb{N})$$

is the reward accumulated when for each

$$(s, n) \in q$$

the marking of the state variable

$$s$$

is

$$n$$

. Informally a rate reward variable

$$x$$

accumulates a defined reward whenever a subset of the state variables take on prescribed values.

```
{ "rate_reward": {
    "name": "string",
    "sv_name": "sv_name string",
    "reward": "expression",
    "temporal_type": "instant_of_time"|"interval_of_time"|"time_averaged_interval_of_time"|'
    "temporal_domain": ["float"]
  }
}
```

The IR definition of a rate reward associates the identifier of a single state variable with a temporal type and time domain, and a reward expression. The reward variable begins with an initial value of zero and functions as an accumulator, accumulating value equal to the evaluation of the reward expression as specified by the temporal type and time domain.

Impulse Rewards

An impulse reward is formally defined as a function

$$\mathcal{I} : E \to \mathbb{R}$$

where

$$e \in E, \mathcal{I}_e$$

is the reward for the completion of

$$e$$

. Informally an impulse reward variable

$$x$$

accumulates a defined reward whenever the event

$$e$$

fires.

```
{ "impulse_reward": {
    "name": "string",
    "ev_name": "ev_name string",
    "reward": "expression",
    "temporal_type": "instant_of_time"|"interval_of_time"|"time_averaged_interval_of_time"|'
    "temporal_domain": ["float"]
  }
}
```

The IR definition of an impulse reward associates the identifier of a single event with a temporal type and time domain, and a reward expression. The reward variable begins with an initial value of zero and functions as an accumulator, accumulating value equal to the evaluation of the reward expression as specified by the temporal type and time domain.

Temporal Characteristics of Reward Variables

Both rate and impulse reward variables measure the behavior of a model

$$M$$

with respect to time. As such, a reward variable

$$\theta$$

is declared as either an instant-of-time variable, an interval-of-time variable, a time-averaged interval-of-time variable, or a steady state variable. An instant of time variable

$$\Theta_t$$

is defined as:

$$\theta_t = \sum_{\nu \in P(S,\mathbb{N})} \mathcal{R}(\nu) \cdot \mathcal{I}_t^{\nu} + \sum_{e \in E} \mathcal{I}(e) \cdot I_t^{e}$$

Intuitively a rate reward declared as an instant-of-time variable can be used to measure the value of a state variable precisely at time

$$t$$

, and an impulse reward declared as an instant-of-time variable can be used to measure whether a given event fired at precisely time

$$t$$

. While the latter is not a particularly useful measure (as the probability of an event with a firing time drawn from a continuous distribution at time

$$t$$

is 0) it is defined primarily for closure reasons, as well as extensions to discrete general distributions.

An interval-of-time variable intuitively accumulates reward over some fixed interval of time

$$[t, t+1]$$

. Given such a variable

$$\theta_{[t,t+1]}$$

we formally define interval-of-time variables as:

$$\theta_{[t,t+1]} = \sum_{\nu \in P(S,\mathbb{N})} \mathcal{R}(\nu) \cdot \mathcal{J}^{\nu}_{[t,t+1]} + \sum_{e \in E} \mathcal{I}(e) N^{e}_{[t,t+1]}$$

where

- $$J^{\nu}_{[t,t+1]}$$

  is a random variable which represents the total time the model spent in a marking such that for each

  $$(s, n) \in \nu$$

  , the state variable

  $$s$$

  has a value of

  $$n$$

  during the period

  $$[t, t+1]$$

  .

- $$I^{e}_{t \to \infty}$$

  is a random variable which represents the number of times an event

  $$e$$

  has fired during the period

  $$[t, t+1]$$

  .

Time-averaged interval of time variables quantify accumulated reward over some interval of time. Such a variable

$$\theta'_{[t,t+1]}$$

is defined formally as:

$$\theta'_{[t,t+1]} = \frac{\theta_{[t,t+1]}}{l}$$

Steady state reward variables are realized by testing for initial transients, and calculating an instant of time variable after a model has reached a stable steady state with high confidence.

Practical considerations of temporal domains

Given a rate or impulse reward with temporal type and domain:

```
{ "temporal_type": "instant_of_time"|"interval_of_time"|"time_averaged_interval_of_time"|"st
"temporal_domain": ["float"]
}
```

The temporal domain is interpreted as follows:

- **Instant of time** - a list of times to sample the reward. For a list

  $$[t_0, \ldots, t_{n-1}]$$

  we create

  $$n$$

  separate instant of time reward variables which all accumulate exactly one observation at the specified time.
- **Interval of time** and **Time averaged interval of time** - the list should include exactly three values,

  $$[t_0, t_{n-1}, s]$$

. A single accumulator is created which accumulates rewards

$$\frac{t_{n-1} - t_0}{s}$$

times at points

$$t_0, (t_0 + s), (t_0 + 2s), \ldots$$

.

- **Steady state** - the list should include exactly one value, which is an estimate of when the system reaches steady state. A single accumulator is created to store the appropriate reward.

Composed Rewards

Composed rewards are defined with rewards that are a special type of expression, a reward variable expression.

```
{ "composed_reward": {
    "name": "string",
    "reward": "rv_expression"
  }
}
```

Reward Expressions

Reward variable expressions differ from standard expressions in the AMIDOL IR in their identifier term, which can only refer to names of other reward variables.

```
rv_expression ::= rv_term "+" rv_expression | rv_term "-" rv_expression |
                  rv_term
rv_term       ::= "(" rv_expression ")" | rv_term "*" rv_expression |
                  rv_term "/" rv_expression |rv_atom
rv_atom       ::= rv_identifier | literal
rv_identifier ::= rate_reward_name | impulse_reward_name | composed_reward_name | constant_r
literal       ::= integer | float
```

Composed rewards allow us to construct a set of arbitrary measures on the set of impulse and rate rewards. Composed rewards need not be solved for during execution of a model, but can be computed after solving a model for the impulse and rate rewards contained in the reward expression for a composed reward variable.

Practical Considerations

Partially Defined Models

A model is considered partially defined, and thus unexecutable if: * It contains expressions which refer to `identifier`s or `rv_identifier`s which are undefined. * If it contains no state variables.

A model without a reward variable is fully defined, but trivially so as it has a null solution.

Transformations

Composition

Turing Completeness

Compactness of Representation

Examples