
Foundations for the Run-time Monitoring of Reactive Systems

-Fundamentals of the MaC language

Mahesh Viswanathan

*University of Illinois Urbana
Champaign, USA*

`vmahesh@cs.uiuc.edu`

Moonzoo Kim

*Pohang University of Science and
Technology, South Korea*

`moonzoo@postech.ac.kr`



Motivation and Focus

- Motivation
 - As more computer systems are deployed in daily life, more needs on the correctness of the computer systems
 - Monitoring, as a complementary method to formal verification and testing, can increase assurance of systems
- Two Focuses
 - Theoretical description on the power of monitoring
 - Accurate description of monitorable languages
 - Subset of the class of safety languages equivalent to \mathcal{M}_i^0
 - Practical framework of monitoring
 - Monitoring and Checking (MaC) architecture
 - Proof of expressiveness for general monitoring purpose



- The Class of Monitorable Languages M
- M in the Arithmetic Hierarchy
- ω -automata with storage
- The Monitoring and Checking (MaC) framework
- Conclusion



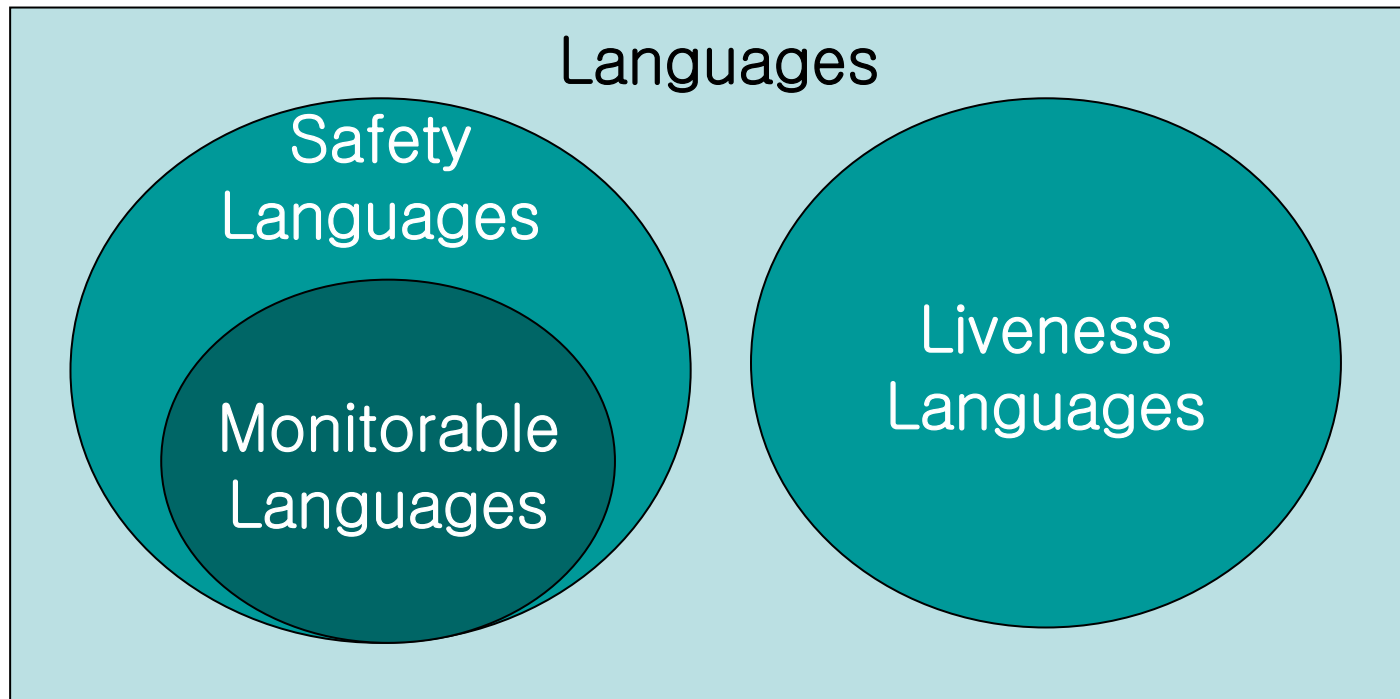
Safety Languages

- A **language** is a set of program executions
 - An **execution** of a program is an infinite sequence of program states S
- **Safety languages** are languages that require that nothing bad happens during an execution
 - Monitor should be able to reject faulty execution after looking at a **finite** prefix
 - A language L is a safety language if $\sigma \in L$
iff $\forall i \exists \beta$ s.t. $\sigma(0,i) \cdot \beta \in L$



Relationship among Languages

- Safety languages \neq monitorable languages



Monitorable Languages

- Example

- $\Sigma = \{0, 1, a, b\}$

- $H_* = \{x \cdot a \cdot y \mid x, y \in \{0, 1\}^*\},$

the Turing Machine encoded by x
halts on input y .

- Membership of H_* (the halting problem) is undecidable

- We can define a safety language

$$H_\omega = (H_* \cdot b^\omega) \cup (\{0, 1\}^* \cdot a \cdot \{0, 1\}^\omega) \cup \{0, 1\}^\omega$$



Monitorable Languages (cont.)

- A language $L \subseteq S^\omega$ is **monitorable** iff
 - L is a safety language
 - $S^* \setminus \text{pref}(L)$ is **recursively enumerable**
- Let us call the class of monitorable languages \mathbf{M}



Definition of Π_1^0

- Arithmetic Hierarchy
 - Hierarchy of languages over the class of recursive relation C
 - L is in Π_n^0 if and only if for some $R \in C$

$$L = \{ \alpha \mid \forall v_1 \exists v_2 \dots Q_n i. R(v_1, v_2, \dots, \alpha(0, i)) \}$$

- An infinite language L is in Π_1^0 if and only if for some recursive relation R

$$L = \{ \alpha \mid \forall i. R(\alpha(0, i)) \}$$



$$M = \Pi_1^0$$

- $\Pi_1^0 \Rightarrow M$
 - Suppose $L \in \Pi_1^0$
 - 1. L is a safety language from the def of Π_1^0
 - 2. $u \in \Sigma^* \setminus \text{pref}(L)$ iff $\neg R u$
 $\Rightarrow \Sigma^* \setminus \text{pref}(L)$ is recursively enumerable.
(one way proof done)
- $M \Rightarrow \Pi_1^0$



ω -automata with storage

- Storage type
- ω -automata with a storage
- Equivalence results



- A storage type is a 5-tuple $X = (C, C_0, P, F, [])$
 - C : a set of storage configurations
 - C_0 : a set of initial storage configuration
 - P : a set of predicate symbols
 - F : a set of function symbols
 - $[]$: a function that defines the semantics of P and F
 - $[p]: C \rightarrow \{\text{true}, \text{false}\}$
 - $[f]: C \rightarrow C$
- Ex. Accumulator storage type
 - $AC = \{N, \{0\}, \{zero\}, \{+_k, -_k\}, []\}$



ω -automata with a storage

- An X -automata is a 5-tuple $(Q, \Sigma, \delta, q_0, c_0)$
 - $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times BE(P) \times Q \times F^*$
- The class of ω -languages accepted by X -automata will be denoted by XL



ω -automata with a storage N_m

- N -fold product of a storage type X : X^n
- The storage type of m integer

$$N_m = (C, C_0, P, F, [])$$

$$- C = N^m, C_0 = \{ \langle 0, \dots, 0 \rangle \}, P = \{ zero_i / 0 \leq i < m \},$$

$$- F = \{ ADR_{i,j}, SBR_{i,j}, ADC_{i,k}, SBC_{i,k}, MLC_{i,k}, QC_{i,k}, RMC_{i,k} \}$$

- $N_*L = \bigcup_m N_m L$



ω -automata with a storage

- Characteristics of an X -automata
 - d -:deterministic
 - r -:real-time (no ε transition)
 - f -:finite delay (no infinite runs on a finite word)
- The following classes of ω -languages are equivalent
 - 1) $M = \Pi_1^0$
 - 2) $df-N_*L$
 - 3) $dr-N_*L$

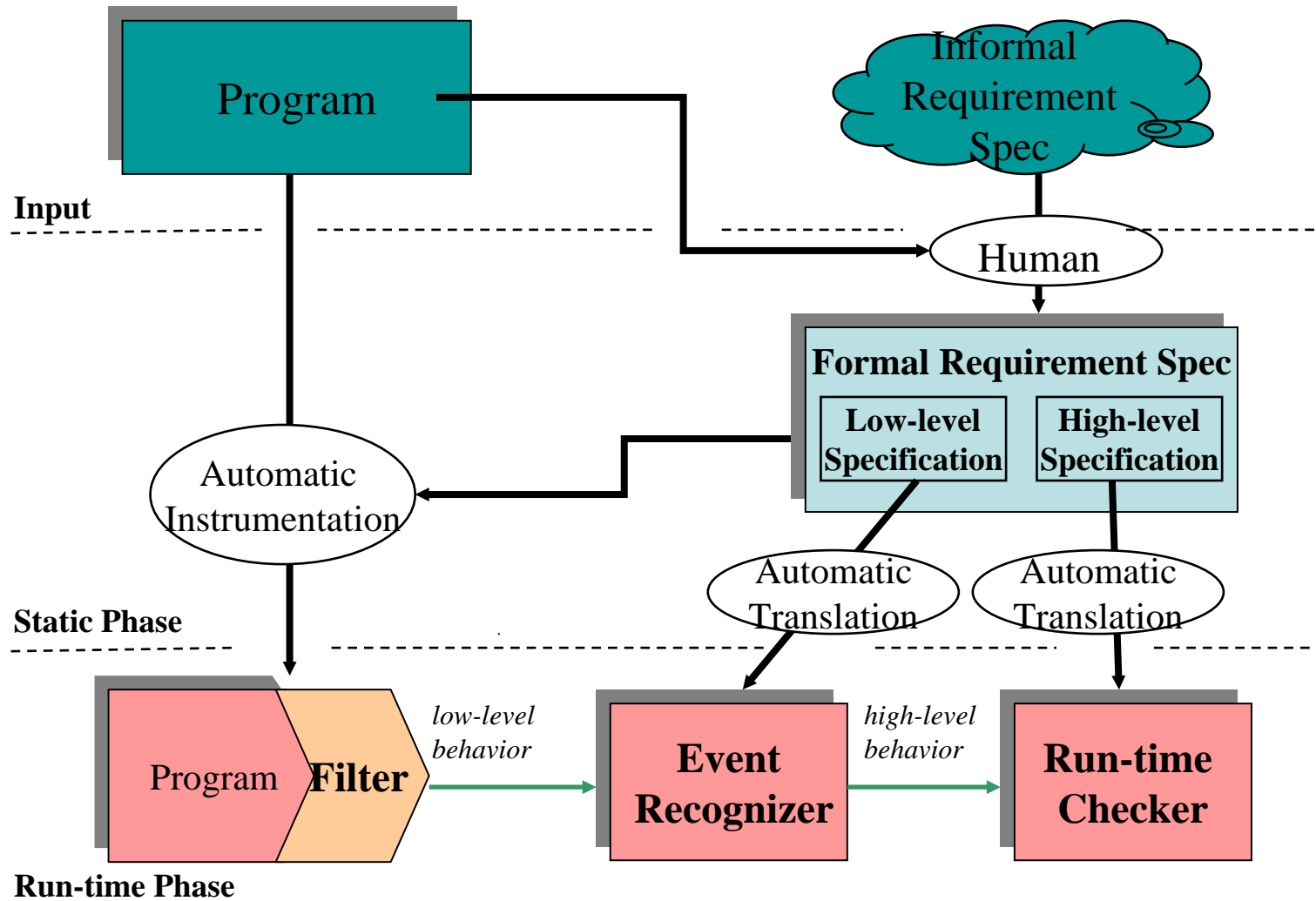


Proof Sketch

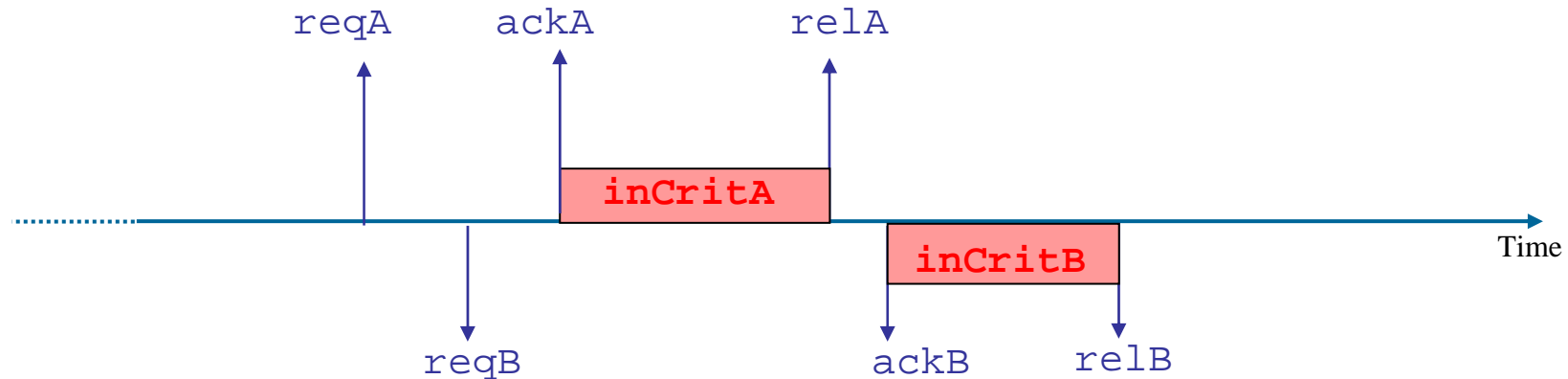
- $M = \mathcal{P}_1^0 \Rightarrow df-N_*L$
 - For a language $L \in \mathcal{P}_1^0$, $\alpha \in L$ iff $\forall i. R(\alpha(0,i))$ where R is a *recursive* language. Therefore, there exists a deterministic finite delay N_m -automaton which accepts exactly the same words as R .
- $df-N_*L \Rightarrow dr-N_*L$
 - The real-time automaton reads an input symbol every time and puts it into the *buffer*, while the actual computation is then performed on the buffered input.
- $dr-N_*L \Rightarrow M = \mathcal{P}_1^0$



Overview of the MaC Architecture



Design of the MaC Language



- Must be able to reason about both **time instants** and information that holds for a **duration of time** in a program execution.
 - **Events** and **conditions** are a natural division, which is also found in other formalisms such as SCR, durational calculus, interval logic, etc
- Need temporal operators combining events and conditions in order to reason about traces.



Logical Foundation

$C ::= c \text{ /defined}(C) \mid [E_1, E_2) \mid \neg C \mid C_1 \vee C_2 \mid C_1 \wedge C_2$

$E ::= e \mid \text{start}(C) \mid \text{end}(C) \mid E_1 \vee E_2 \mid E_1 \wedge E_2 \mid E \text{ when } C$

- Conditions interpreted over 3 values: true, false and undefined.
- $[., .)$ pairs a couple of events to define an interval.
- `start` and `end` define the events corresponding to the instant when conditions change their value.



Meta Event Definition Language (MEDL)

- Expresses requirements using the events and conditions
- Expresses the subset of safety languages.
- Describes the *safety requirements* of the system, in terms of conditions that must always be true, and alarms (events) that must never be raised.
 - property **safeRRC** = IC -> GD;
 - alarm **violation** = start (!safeRRC);
- *Auxiliary variables* may be used to store history.
 - endIC-> { num_train_pass' =
 num_train_pass + 1; }

```
ReqSpec <spec_name>
```

```
/* Import section */  
import event <e>;  
import condition <c>;
```

```
/*Auxiliary variable */  
var int <aux_v>;
```

```
/*Event and condition */  
event <e> = ...;  
condition <c>= ...;
```

```
/*Property and violation */  
property <c> = ...;  
alarm <e> = ...;
```

```
/*Auxiliary variable update*/  
<e> -> { <aux_v'> := ... ; }
```

```
End
```



Expressive Power of MEDL

- MEDL is expressive enough for M
- Proof sketch: for every $dr-N_*$ -automaton A , there exists a MEDL script
 - The input alphabet of A will be all the imported events
 - Auxiliary variable for each of the m storing locations
 - Auxiliary variable state
 - A transition $(q1, a, b, q2, f)$ is transformed into a guard $(a \ \&\& \ E_b)$ when $(state == q1) \rightarrow \{state' := q2; f'; \}$



Conclusion

- We show that the class of monitorable language is *strict subset* of safety languages
- MEDL, the specification language of the MaC framework is *expressive enough* for the class of monitorable languages M
- Working on extension of MEDL for easy property description

