UNIVERSITÄT KOBLENZ · LANDAU

Formal Methods for Multi-agent Systems

Sergey Dedukh

Contents

Introduction and motivation

Challenge:

In complex environments how to ensure that agents will behave as we expect them to, or at least will not behave in unacceptable or undesirable ways?

Possible answer:

High level abstractions that can lead to simpler techniques for design and development.

Introduction and motivation

Formal methods help

- In understanding of system design at a level higher than a specific implementation
- To debug specifications and validate system implementations
- In the long run, in helping developing a clearer understanding of problems and solutions

Theoretical background

Predicate and Modal Logic

Predicate logic

- Atomic propositions
- Connectives: $\wedge$, $\vee$, $\neg$, $\rightarrow$

Modal logic

- $\Box$ – necessity operator
- $\Diamond$ – possibility operator
- Sets of possible worlds with accessibility relation

Dynamic and Temporal Logic

Dynamic logic (modal logic of action)

- Adds different kinds of actions
- Necessity and possibility are based upon different types of actions available

Temporal logic (Logic of time)

- Linear vs. Branching
- Discrete vs. Dense
- Moment-based vs. Period-based

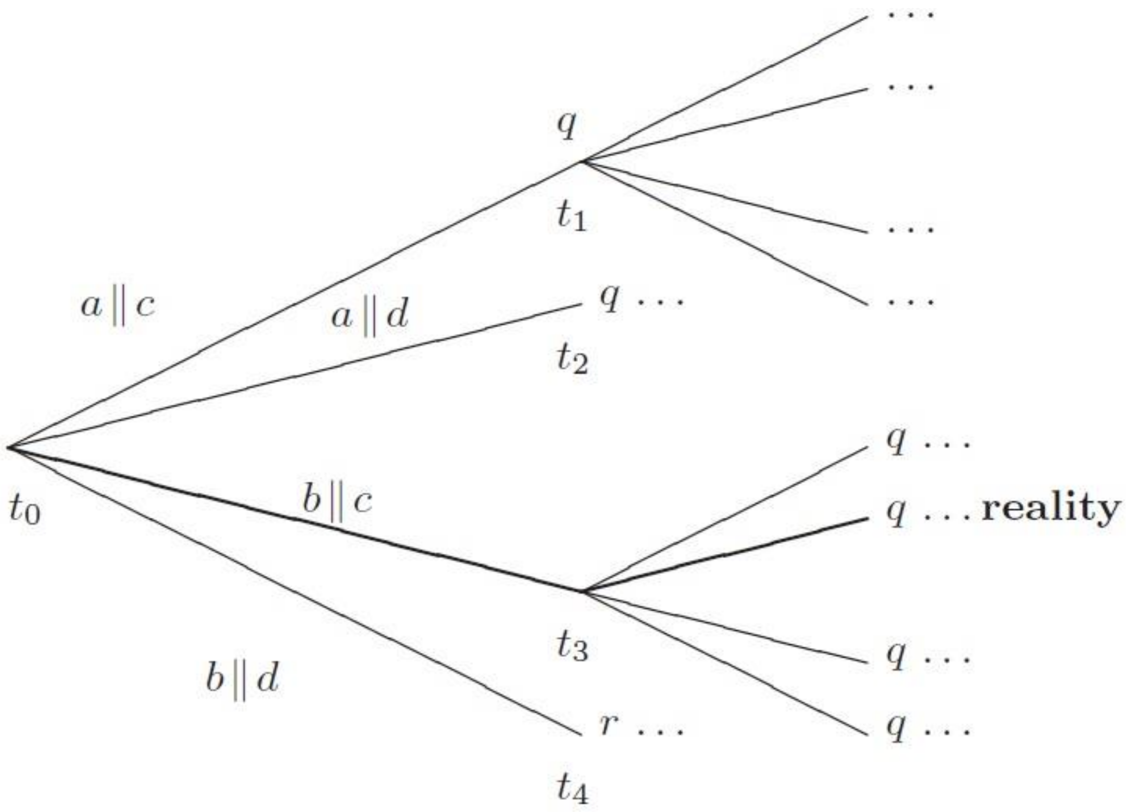
Linear Temporal Logic

Linear temporal logic

- $p \cup q$ – true at moment t *iff* q *holds at future moment and* p *holds on all moments between* t *and selected occurrence of* q.
- Fp – p holds sometimes in the future on a given path
- Gp – p always holds in the future on a given path
- Xp – p holds in the next moment
- Pq – q held in a past moment

UNIVERSITÄT KOBLENZ · LANDAU

Branching Temporal Logic



Branching Temporal and Action Logic

Branching-time temporal and action language

- Ap – p holds in all paths at the present moment
- Ep – p holds in some path at the present moment
- Rp – p holds in the real path at the present moment

Formal methods for a single agent

The BDI Model

Formalizing agent's cognitive specifications:

- Bel – belief
- Des – desire (goal if achievable and consistent)
- K_h – know-how
- Int – intention

Example:

$DesAFwin \wedge IntEFbuy \wedge \neg BelAFwin$

Constraints for Intentions

1. Satisfiability

   $xIntp \rightarrow EFp$

2. Temporal Consistency

   $(xIntp \wedge xIntq) \rightarrow xInt(Fp \wedge Fq)$

3. Persistence does not entail success

   $EG((xIntp) \wedge \neg p)$ is satisfiable

4. Persist while succeeding

Abstract Architecture

Inputs to the system

- Events are received via an event queue
- Events of two kinds
  - External (environmental)
  - Internal

Outputs from the system

- Atomic actions performed by execute() function

Abstract BDI Interpreter

```
BDI-interpreter
initialize-state();
do
    options := option-generator(event-queue,B,G,I);
    selected-options := deliberate(options,B,G,I);
    update-intentions(selected-options,I);
    execute(I);
    get-new-external-events();
    drop-successful-attitudes(B,G,I);
    drop-impossible-attitudes(B,G,I);
until quit.
```

Plans

(a)

Type: drink-soda
Invocation:
g-add(quenched-thirst)
Precondition: have-glass
Add List: {quenched-thirst}
Body:

$(1) \xrightarrow{\text{have-soda}} (2) \xrightarrow{\text{drink}} (3)$

(b)

Type: drink-water
Invocation:
g-add(quenched-thirst)
Precondition: have-glass
Add List: {quenched-thirst}
Body:

$(1) \xrightarrow{\text{open-tap}} (2) \xrightarrow{\text{drink}} (3)$

(c)

Type: get-soda
Invocation:
g-add(have-soda)
Precondition: true
Add List: {have-soda}
Body:

$(1) \xrightarrow{\text{open-fridge}} (2) \xrightarrow{\text{get-soda}} (3)$

Generate Options

option-generator(trigger-events)

options := $\{\}$;
for trigger-event $\in$ trigger-events **do**
  for plan $\in$ *plan-library* **do**
    if matches(invocation(plan),trigger-event) **then**
      if provable(precondition(plan),B) **then**
        options := options $\cup$ $\{$plan$\}$;
return(options).

Deliberate options

```
deliberate(options)
if length(options) ≤ 1 then return(options);
else metalevel-options := option-generator(b-add(option-set(options)));
  selected-options := deliberate(metalevel-options);
  if null(selected-options) then
    return(random-choice(options));
else return(selected-options).
```

Example trace

Bel	Goal	Int	*done*	*succeeded*
glass	–	–	–	–
unchanged	quench	–	–	g-add(quench)
unchanged	unchanged	{ soda; drink}	–	g-add(soda)
¬ remove-soda	unchanged	–	fridge	fridge, g-add(quench)
unchanged	unchanged	{ drink}	tap	tap
quench	–	–	drink	drink

Formal methods for multiple agents

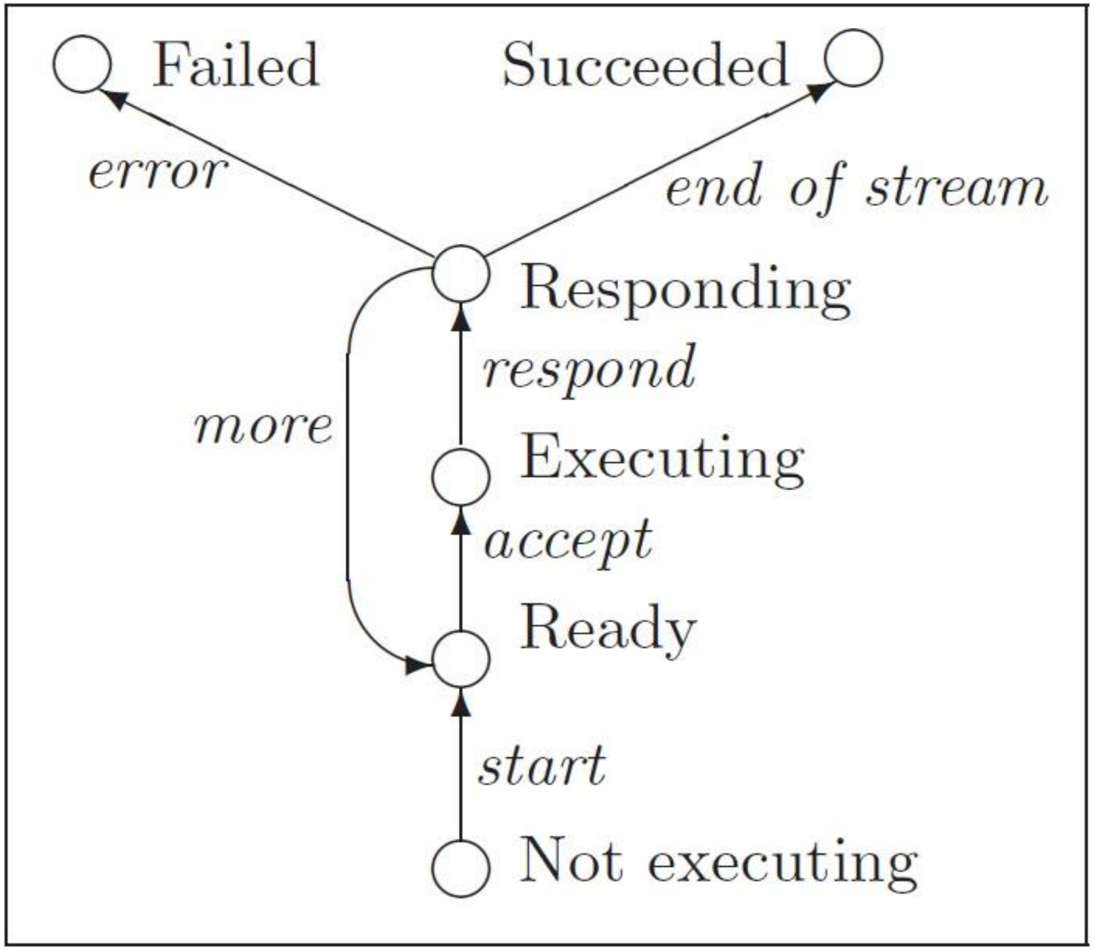
Coordination Architecture

Agents are represented as small “skeletons”, which include only events significant for coordination

Event classes:

- Flexible – may be delayed or omitted
- Inevitable – may be delayed
- Immediate
- Triggerable – based on external request

Agent Skeleton – Example



Specification language

Linear-time language with restrictions

- $\cdot$ – “before” temporal operator
- $\odot$ – concatenation of two traces first of which is finite

Coordination Relationships

	Name	Description	Formal notation
R1	e is required by f	If f occurs, e must occur before or after f	$e \vee \overline{f}$
R2	e disables f	If e occurs, then f must occur before e	$\overline{e} \vee \overline{f} \vee f \cdot e$
R3	e feeds or enables f	f requires e to occur before	$e \cdot f \vee \overline{f}$
R4	e conditionally feeds f	If e occurs, it feeds f	$\overline{e} \vee e \cdot f \vee \overline{f}$
R5	Guaranteeing e enables f	f can occur only if e has occurred or will occur	$e \wedge f \vee \overline{e} \wedge \overline{f}$
R6	e initiates f	f occurs iff e precedes it	$\overline{e} \wedge \overline{f} \vee e \cdot f$
R7	e and f jointly require g	If e and f occur in any order, then g must also occur (in any order)	$\overline{e} \vee \overline{f} \vee g$
R8	g compensates for e failing f	if e happens and f does not, then perform g	$(\overline{e} \vee f \vee g) \wedge (\overline{g} \vee e) \wedge (\overline{g} \vee \overline{f})$

Summary

- Formal methods helps in understanding the design and used for validation purposes
- Different types of logics are being used in formal methods
- BDI concept is used to represent agent’s logical structure
- In multi-agent systems “agent skeleton” concept is used for coordination

Questions?