STRATEGO:
A Model-Based Decision Procedure:

Jin Hyun Kim
jin.kim@gnu.ac.kr
8 Oct 2019

Contents

Programming Language History

1940년 이전 [편집]

초기의 프로그래밍 언어는 현대의 컴퓨터에서 거슬러 올라간다. 초

1801년에 발명된 자카드식 문직기는 자동으로 장식 패턴을 발생시

1842년~1843년의 9개월의 기간에 걸쳐 에이다 러브레이스는 이탈
자세하게 일련의 주석으로 추가했는데, 이는 일부 역사가들이 세계

허먼 홀러리스는 그가 열차의 차장이 기차표 위의 천공 카드의 위치

최초의 컴퓨터 부호들은 이들을 응용하여 만들어졌다.

1940년대 [편집]

이 시기에 개발된 일부 중요 언어는 다음을 포함한다:

- 1943년 - Plankalkül
- 1943년 - 에니악 코딩 시스템[2]
- 1949년~1954년 - 나중에 유니박으로 발전함.[3]

1950년대 ~ 1960년대 [편집]

1950년대에는 다음의 세 가지 현대의 프로그래밍 언어가 설계되었

- 포트란 (1954년)
- 리스프 (1958년)
- 코볼 (1959년)

그 밖의 주요 언어들은 다음과 같다:

- 1951년 - 지역 어셈블리어(Regional Assembly Language)
- 1952년 - 오토코드
- 1954년 - IPL (리스프의 선구자)
- 1955년 - FLOW-MATIC (코볼의 선구자)
- 1957년 - COMTRAN (코볼의 선구자)
- 1958년 - 알골 58
- 1959년 - FACT (코볼의 선구자)
- 1959년 - RPG
- 1962년 - APL
- 1962년 - 시뮬라
- 1962년 - SNOBOL
- 1963년 - CPL (C의 선구자)
- 1964년 - BASIC
- 1964년 - PL/I
- 1967년 - BCPL (C의 선구자)

1968년대 ~ 1978년대 [편집]

1960년대 말에서 1970년대 말의 기간 동안 다음의 다섯 가지 각

- 시뮬라(Simula, 1960년대 말 발명)
- C
- 스몰토크 (1970년대 중반)
- 프롤로그 (1972년)
- ML

그 밖의 주요 언어로는 다음과 같다.

- 1968년 - 로고
- 1969년 - B (C의 선구자)
- 1970년 - 파스칼
- 1970년 - 포스 (Forth)
- 1975년 - 스킴 (Scheme)
- 1978년 - SQL (처음에는 쿼리 언어일 뿐이었으나, 훗날 프

1980년대 [편집]

이 기간 동안 개발된 주요 언어들은 다음과 같다:

- 1980년 - C++ (처음 이름은 C with classes였으나 1983년
- 1983년 - 에이다
- 1984년 - 커먼 리스프
- 1984년 - MATLAB
- 1985년 - 에펠(Eiffel)
- 1986년 - 오브젝티브-C
- 1986년 - 얼랭
- 1987년 - 펄
- 1988년 - Tcl
- 1988년 - 매스매티카(Mathematica)
- 1989년 - FL (Backus);

1980년대 [편집]

이 기간 동안 개발된 주요 언어들은 다음과 같다:

- 1980년 - C++ (처음 이름은 C with classes였으나 1983년 7월 이름이 C++로 변
- 1983년 - 에이다
- 1984년 - 커먼 리스프
- 1984년 - MATLAB
- 1985년 - 에펠(Eiffel)
- 1986년 - 오브젝티브-C
- 1986년 - 얼랭
- 1987년 - 펄
- 1988년 - Tcl
- 1988년 - 매스매티카(Mathematica)
- 1989년 - FL (Backus);

1990년대 [편집]

이 기간 동안 개발된 주요 언어들은 다음과 같다:

- 1990년 - 하스켈
- 1991년 - 파이썬
- 1991년 - 비주얼 베이직
- 1991년 - HTML (마크업 언어)
- 1993년 - 루비
- 1993년 - 루아
- 1994년 - CLOS (ANSI 커먼 리스프의 일부)
- 1995년 - 자바
- 1995년 - 델파이 (오브젝트 파스칼)
- 1995년 - 자바스크립트
- 1995년 - PHP
- 1996년 - WebDNA
- 1997년 - Rebol
- 1999년 - D

현재의 경향 [편집]

2000년 이후로 개발된 주요 언어들은 다음과 같다:

- 2000년 - 액션스크립트
- 2001년 - C#
- 2001년 - 비주얼 베이직 닷넷
- 2002년 - F#
- 2003년 - 그루비
- 2003년 - 스칼라
- 2003년 - 팩터
- 2007년 - Clojure
- 2009년 - 고
- 2011년 - 다트
- 2014년 - 스위프트
- 2015년 - 러스트

Programming Language History

Mother Tongues

Tracing the roots of computer languages through the ages

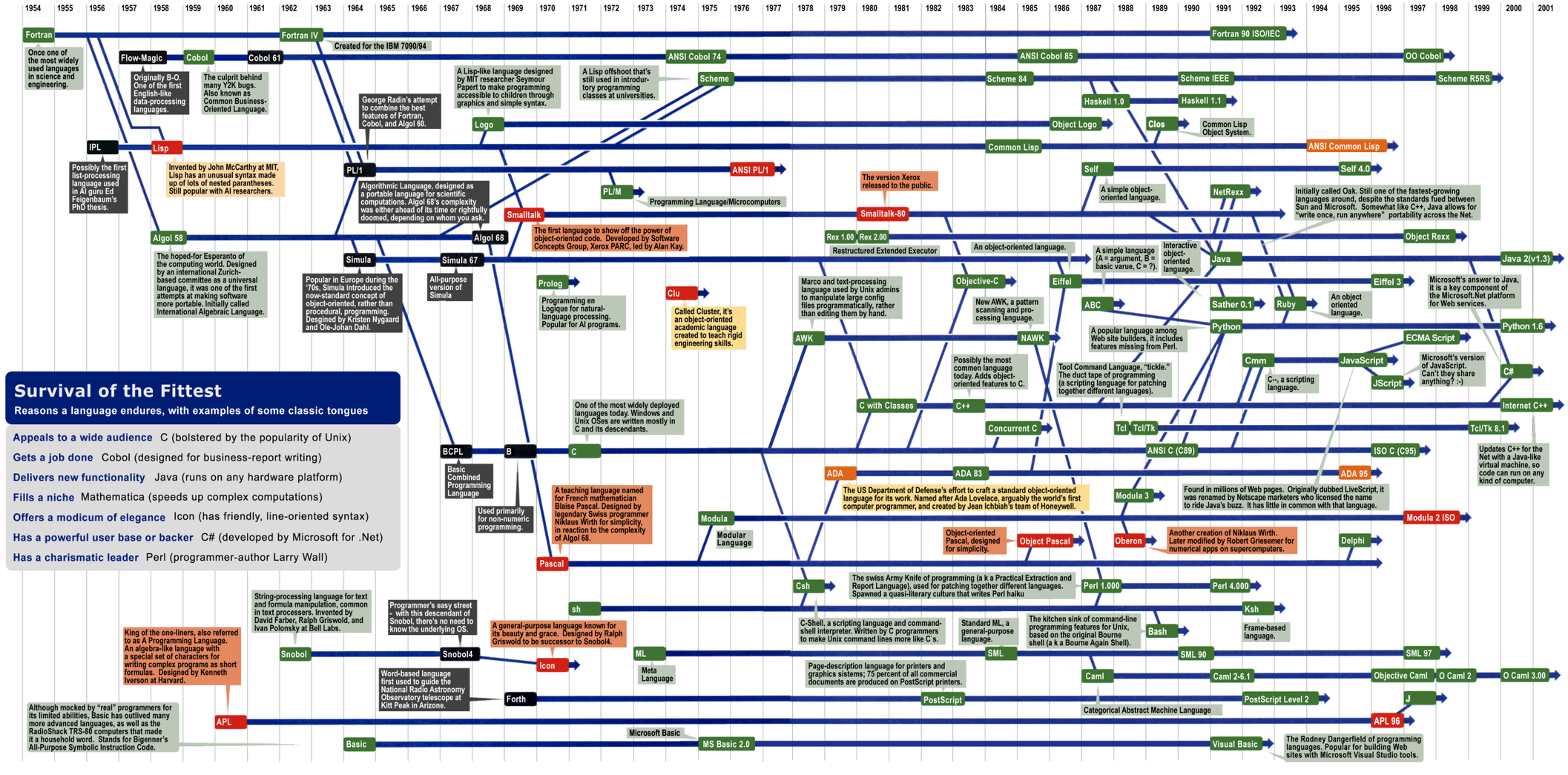
Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers-electronic lexicographers, if you will-aim to save, or at least document the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua frangas. Among the most endangered are Ada, APL, B (the predecessor of C), Lsp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Musuem in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *HTTP://www.informatik.uni-freiburg.de/Java/misc/lang_list.html*. - Michael Mendeno

Key

- 1954 — Year Introduced
- **Active:** thousands of users
- **Protected:** taught at universities; compilers available
- **Endangered:** usage dropping off
- **Extinct:** no known active users or up-to-date compilers
- **Lineage continues**



Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

- **Appeals to a wide audience** C (bolstered by the popularity of Unix)
- **Gets a job done** Cobol (designed for business-report writing)
- **Delivers new functionality** Java (runs on any hardware platform)
- **Fills a niche** Mathematica (speeds up complex computations)
- **Offers a modicum of elegance** Icon (has friendly, line-oriented syntax)
- **Has a powerful user base or backer** C# (developed by Microsoft for .Net)
- **Has a charismatic leader** Perl (programmer-author Larry Wall)

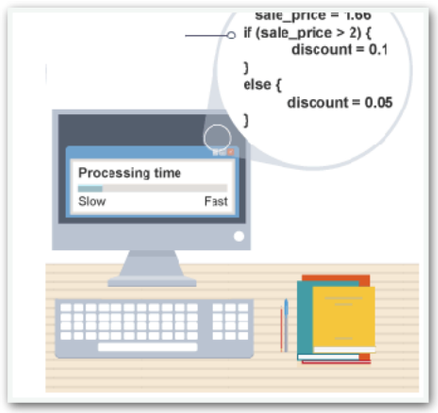
Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

Programming Paradigm Evolution

1Gen

```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

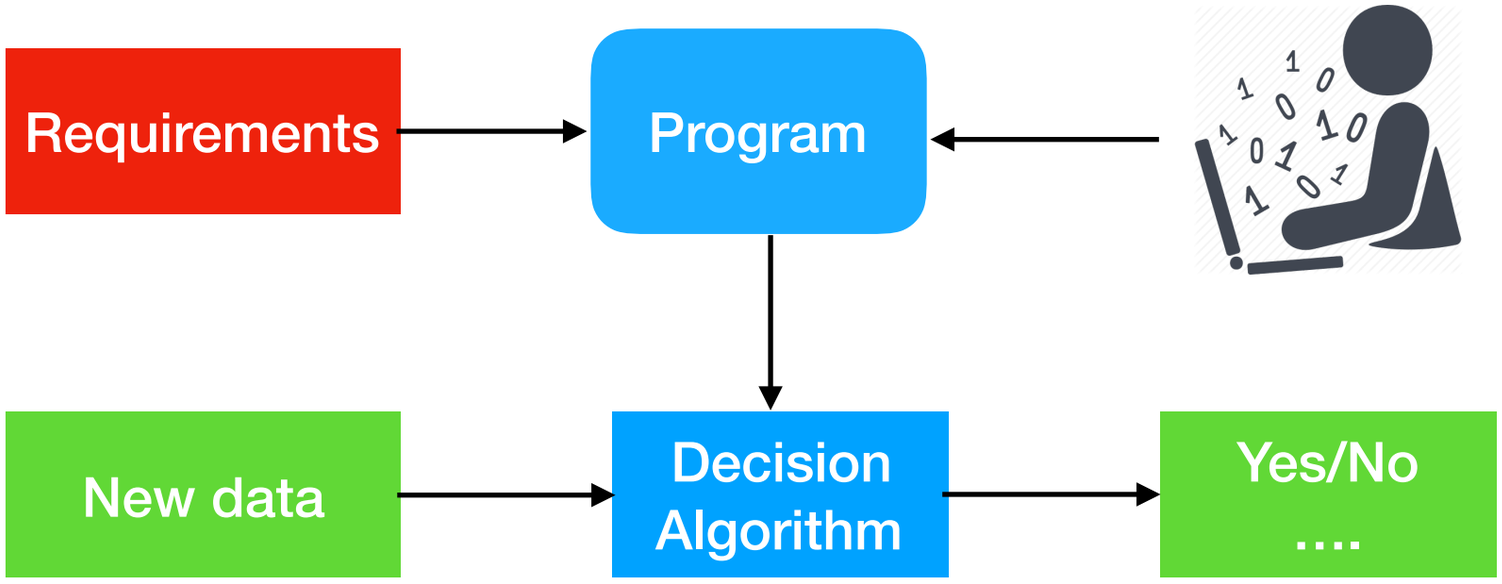
2Gen



3Gen

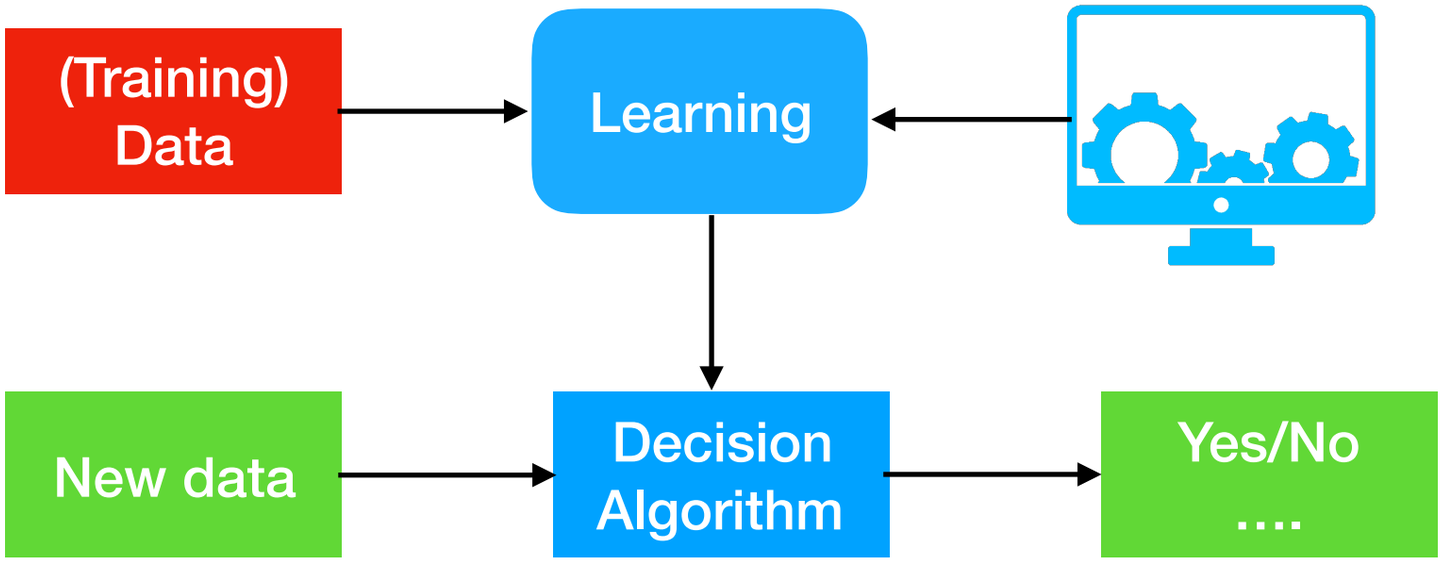
4Gen

Programming



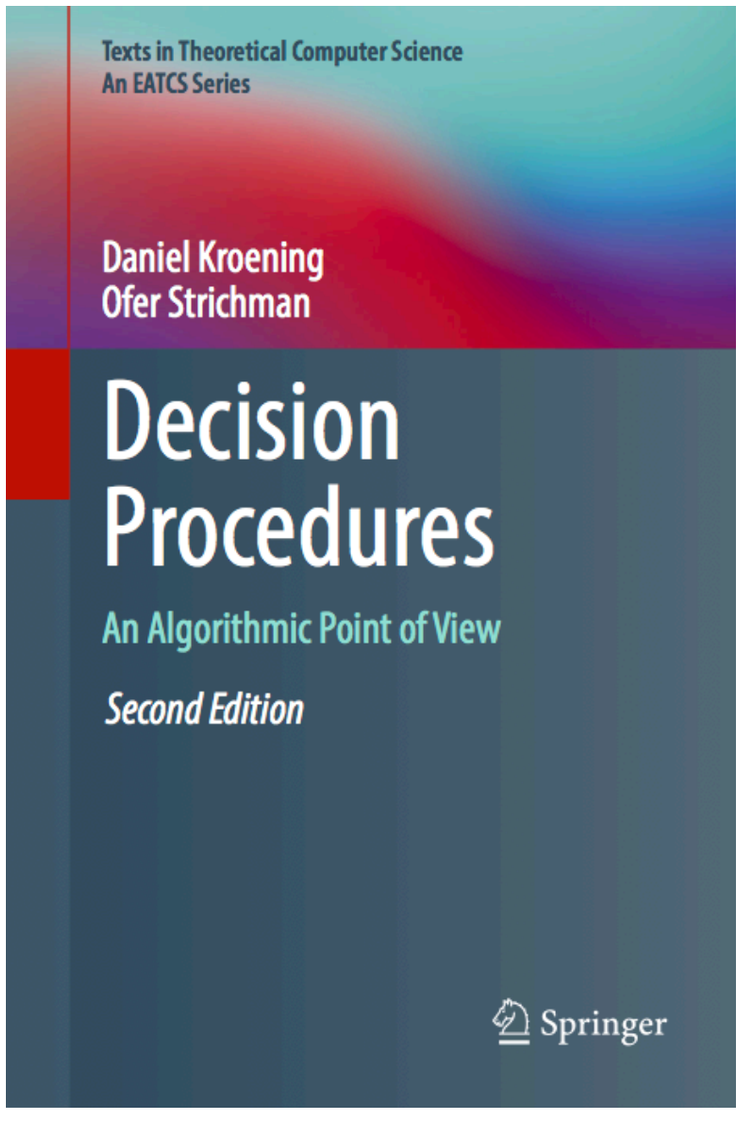
Machine Learning

- Learning to decision



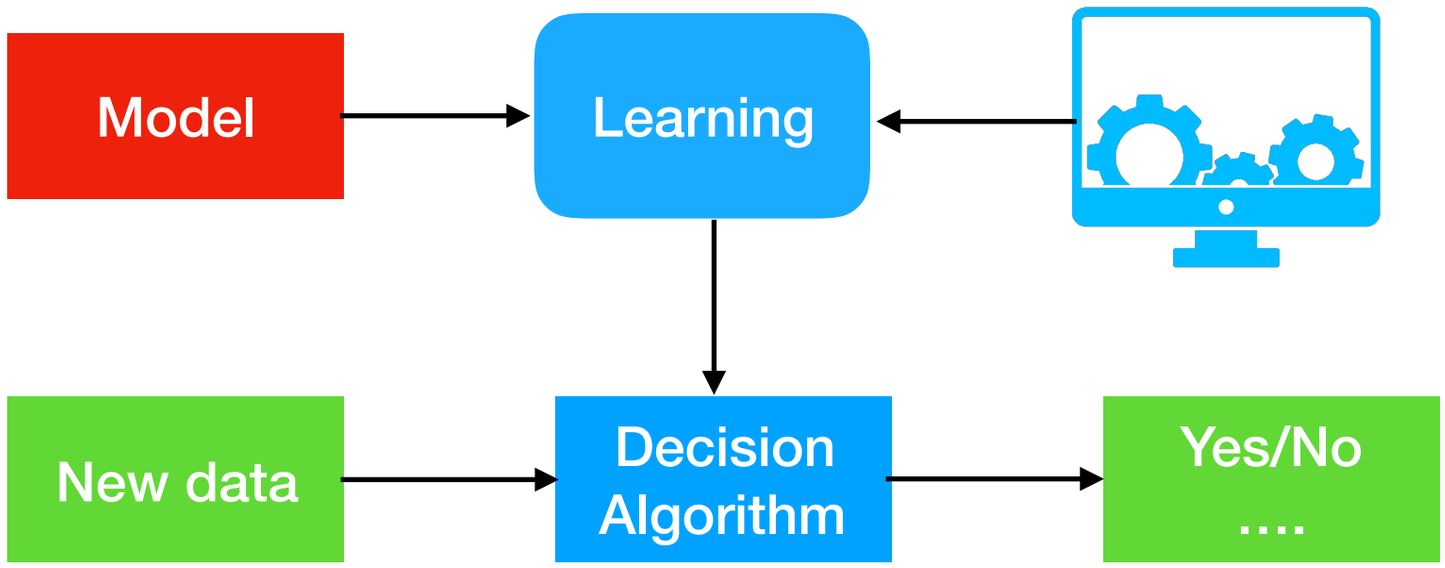
Decision Procedures

- Satisfiability (SAT) - Propositional Logic
- Satisfiability Modulo Theory (SMT) - First Ordered Logic …
- Integer Linear Programming (ILP) - Mathematical Optimization for integer variables
- Constrained Programming (CP) - No objective
- Machine Learning (ML)
- Model-based (Machine) Learning (MML)



Model-based Machine Learning (MML)

- Instead of training data, a model is given



Model-based Machine Learning (MML)

- No need of BIG training data
- Avoid UNSAFE/INSECURE training data
  - Ex) Pingpong —> Tenis
- Need experience of building system models

Stratego

- Strategy learning based on
  - Game theory
  - Model checking
- Generation, optimization, comparison as well as consequence and performance exploration of **strategies** for **stochastic priced timed games** [http://people.cs.aau.dk/~marius/stratego/intro.html]

Terminology

- A **game** is a mathematical model of a system consisting of several players (processes) which have independent objectives, often they are competing and sometimes opposing and even conflicting.

- A **strategy** is a prescription of one player's actions (transitions) for any possible situation (system state) eventually leading to achieving the player's goal.

  - **Deterministic strategy** specifies a single action per state and

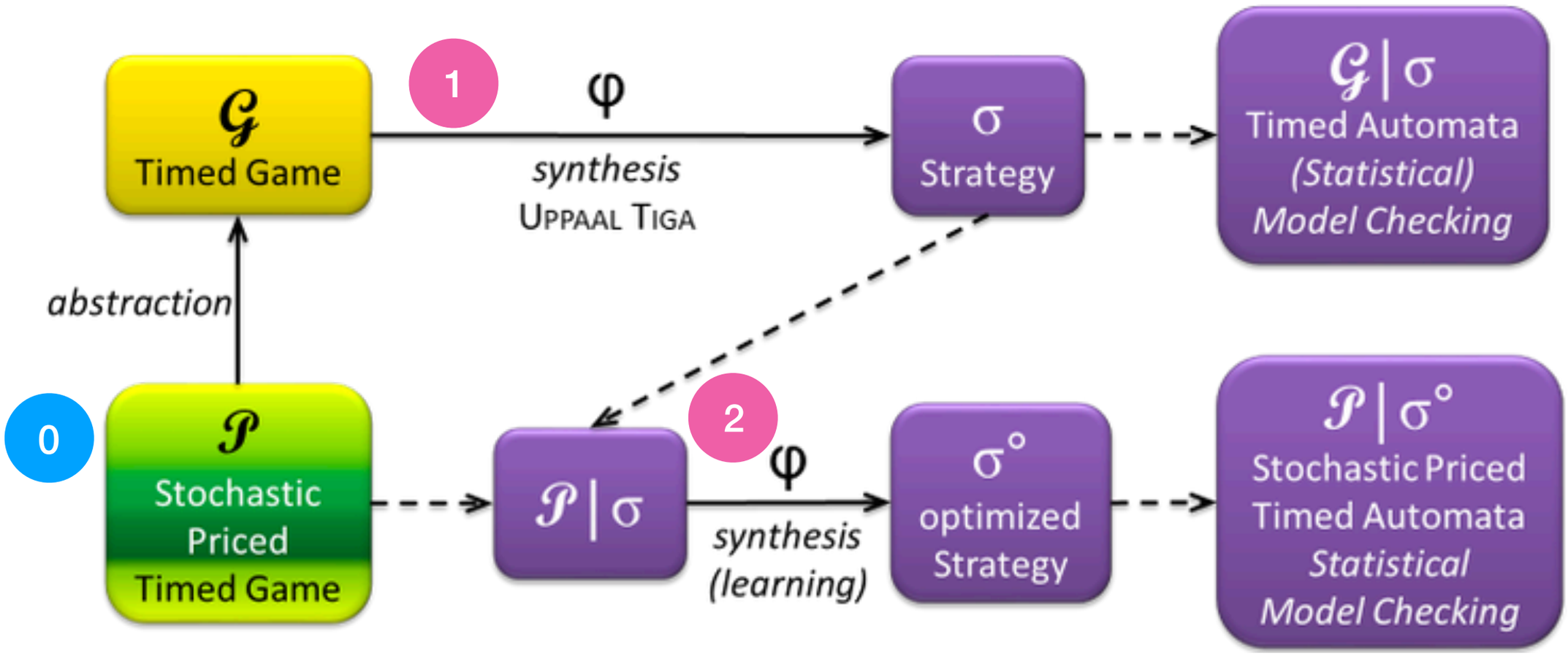
  - **Non-deterministic strategy** may specify several alternatives.

  - **Fully permissive strategy** specifies all alternative actions leading to the goal.

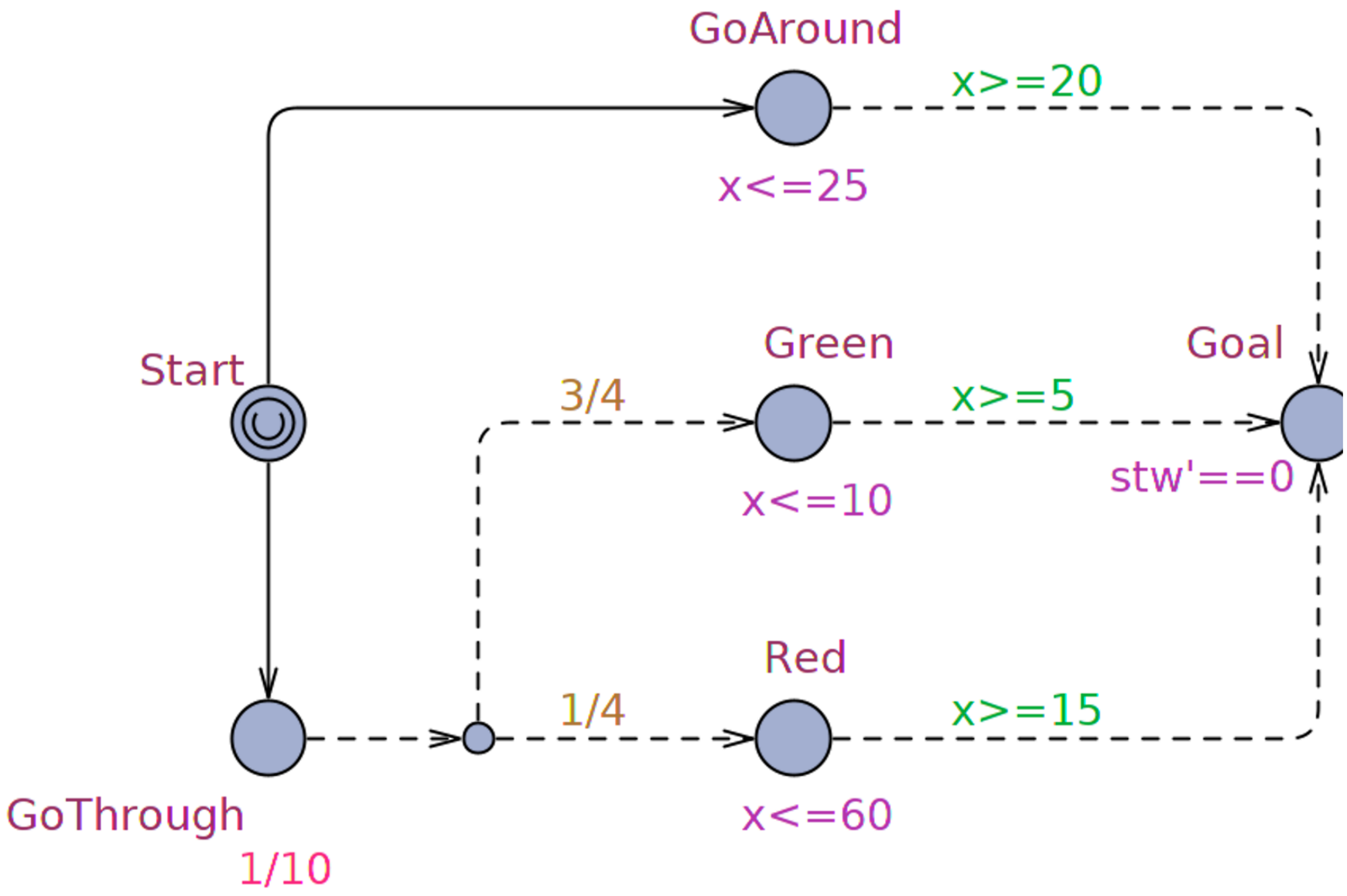
Terminology

- Uppaal **timed game** is a two-player game specified by a network of timed automata with two types of edges:
  - Solid and dashed, corresponding to the player- and opponent-chosen transitions (Uppaal)
- A **stochastic timed game** is a probabilistic extension to a timed game.
- A **stochastic priced timed game** is a further extension with continuous price expressions which allow estimating the distribution of cost in a timed game.

Overview

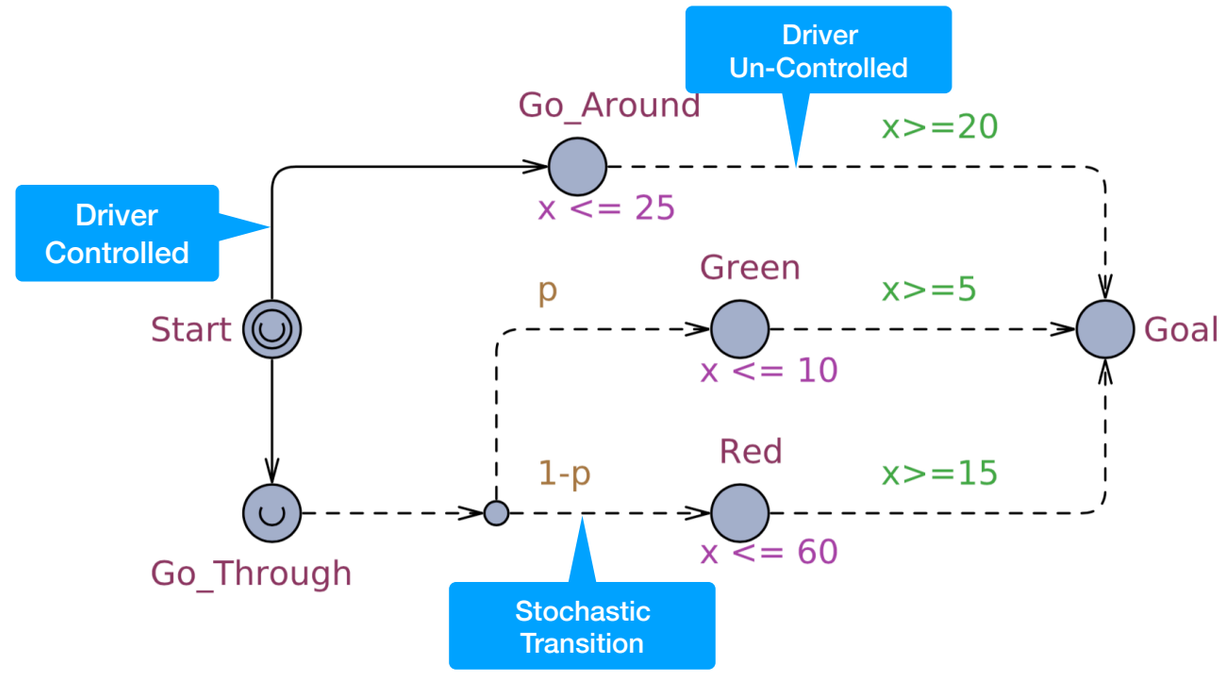

1
φ
synthesis
UPPAAL TIGA
Timed Game
σ
Strategy
Timed Automata
(Statistical)
Model Checking
abstraction
0
Stochastic
Priced
Timed Game
2
φ
synthesis
(learning)
optimized
Strategy
Stochastic Priced
Timed Automata
Statistical
Model Checking

Example


GoAround
x>=20
x<=25
Start
Green
Goal
3/4
x>=5
x<=10
stw'==0
Red
1/4
x>=15
x<=60
GoThrough
1/10

0. Construction (Constraint) Model

- Controlled transition
- Uncontrolled transition



Formal Analysis

- A<> (carpassing.Goal && x<=25)
- Pr[<=100] (<> carpassing.Goal)
- E[<=100; 500] (max: stw)

1. Check Strategy (Objective)

- **control: A<> carpassing.Goal && x <= 25**
  - Is it possible to choose a passage to pass the city within 25 minutes no matter which way is taken?
  - Answer: possible (satisfied).
- **strategy GoFast = control: A<> carpassing.Goal && x <= 25**
  - Is it possible to choose a passage to pass the city within 25 minutes no matter which way is taken?
  - Answer: possible (satisfied) and the tool computes fully permissive strategy named "GoFast".

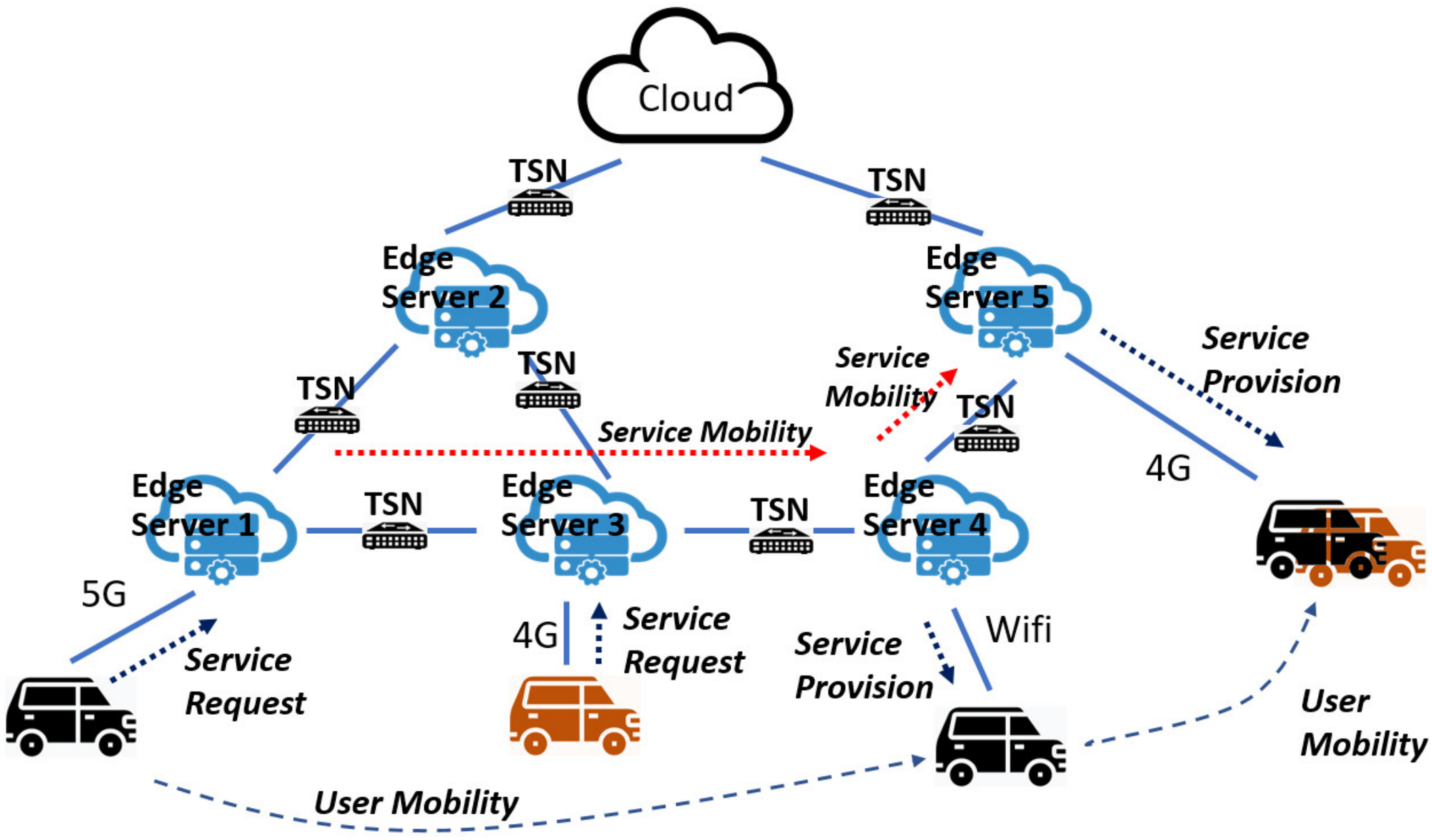
2. Learning

- **A<> (carpassing.Goal && x<=25) under GoFast**
  - Is it possible to pass the city within 25 minutes under decisions of GoFast strategy?
    - Answer: yes (satisfied).
- **E[<=100; 500] (max: stw) under GoFast**
  - Estimate the time (needed to pass the city, where the clock time is stopped).

Use Case:

Cloud-Based Real-Time SW Update

Cloud-based
Real-Time SW Update



Problem Description (Informal)

- For vehicles using connectivity to infrastructure through edges on paths and roads,

- Assign SW app updates to edges such that an objective, such as minimized traveling time for a given update, is satisfied.

Our Approach

- Model-based learning
    - Set up a model of the system and properties of the system to hold,
    - Find a strategy, i.e., a solution, that meets properties,
    - Verify that the system under the control guided by the strategy always holds given properties

Assumptions

- An edge is defined by a range to cover, where a vehicle completes SW update at a specific edge,
- A vehicle has a different driving speed over edges, but determining driving speed of a vehicle over an edge is out of this work scope,
- An SW update is completed at one edge (one-to-one mapping)

Model of System

Let *m* be a memory quantity that an SW update of a vehicle requires, m available quantity of memory on an edge, and c the computation time of a vehicle for processing each unit of memory.

A route is defined by a sequence of paths,

A path is defined by $p = (e, tt_{best}, tt_{wrost})$, where e is an edge on the path, and tt_{best} and tt_{wrost} are, respectively, the best-case traveling time and the worst-case traveling time to cross the edge.

ut_i is a time that elapses to complete all the updates of apps for a vehicle v_i.

Model of System

The underlying system comprises a set of edges and vehicles traveling over edges,

E is a set of edges which relay between a cloud server and vehicles and formally defined by

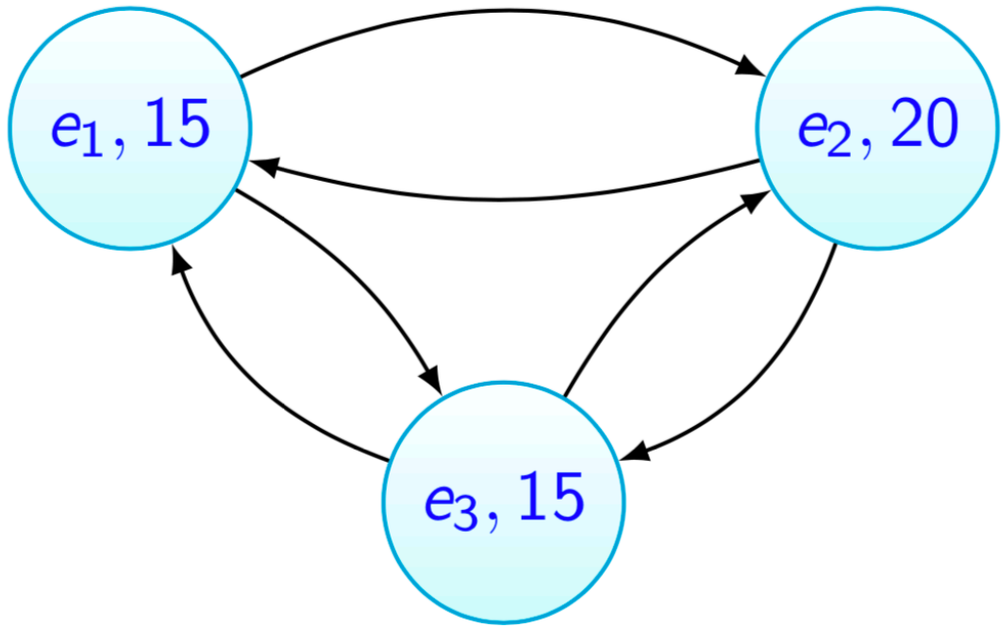
$E = \{e_k \mid e_k = (m_k, 2^E)\}$ where 2^E is a subset of E which can be reached from e_k.

V is a set of vehicles and defined by

$$V = \{v_i \mid v_i = (r_i, u_i, c_i)\}$$

where r_i is a route, $u_i = \{m_j\}$ where m_j is a required memory capacity for the SW update of j, and c_i a processing time for update.

Running Case



$$E = \{e_1 = (15, \{e_2, e_3\}), e_2 = (20, \{e_1, e_3\}), e_3 = (15, \{e_1, e_2\})\}$$

$$\begin{aligned} V = \{v_1 &= ((e_1, 10, 15) \rightarrow (e_2, 14, 15) \rightarrow (e_1, 10, 15), \{3, 4\}, 2), \\ v_2 &= ((e_2, 24, 35) \rightarrow (e_1, 10, 15) \rightarrow (e_3, 21, 25), \{1, 4, 2\}, 1), \\ v_3 &= ((e_1, 10, 15) \rightarrow (e_2, 25, 36) \rightarrow (e_3, 21, 26), \{5, 6\}, 2), \} \end{aligned}$$

Constraints

Let $upd_{i,j}$ be a predicate that indicates whether the update i of app for a vehicle i is over.

The constraints that we have: Given an edge *k*,

$$\sum_{i \in V} \sum_{j \in u_i} upd_{i,j,k} \times m_j \leq \overline{m_k}$$

Given an update j and a vehicle *i*, $upd_{i,j,k}=1$

$$\sum_{k \in E} upd_{i,j,k} = 1$$

Objectives

Given E and V, a scheduling strategy that assigns an update of a vehicle to an edge is computed such that

$$\min \sum_{i \in V} ut_i$$

Edge and Vehicle Specifications

Listing 1: Specification of edges and vehcles in Uppaal models

```
// Edge's capacities
memsz_t curmemsz[eid_t] = {15, 20, 15};

//Vehicle's routes and duration for each edge
const paths_t v1path = {{1,10,15}, {2,14,15}, {1,10,15}, nullpath } ;
const paths_t v2path = {{2,24,35}, {1,10,15}, {3,21,25}, nullpath } ;
const paths_t v3path = {{1,10,15}, {2,25,36}, {3,21,26}, nullpath } ;

//Vehicle update requirements
upd_t v1upds[updid_t] = {{3, 0},{4, 0}, noneedupd};
upd_t v2upds[updid_t] = {{1, 0},{4, 0}, {2, 0}};
upd_t v3upds[updid_t] = {{5, 0},{6, 0}, noneedupd};

Tony = Veh(1, v1upds, 2, v1path, curmemsz);
Nany = Veh(2, v2upds, 1, v2path, curmemsz);
Jony = Veh(3, v3upds, 2, v3path, curmemsz);
```

Vehicle Model

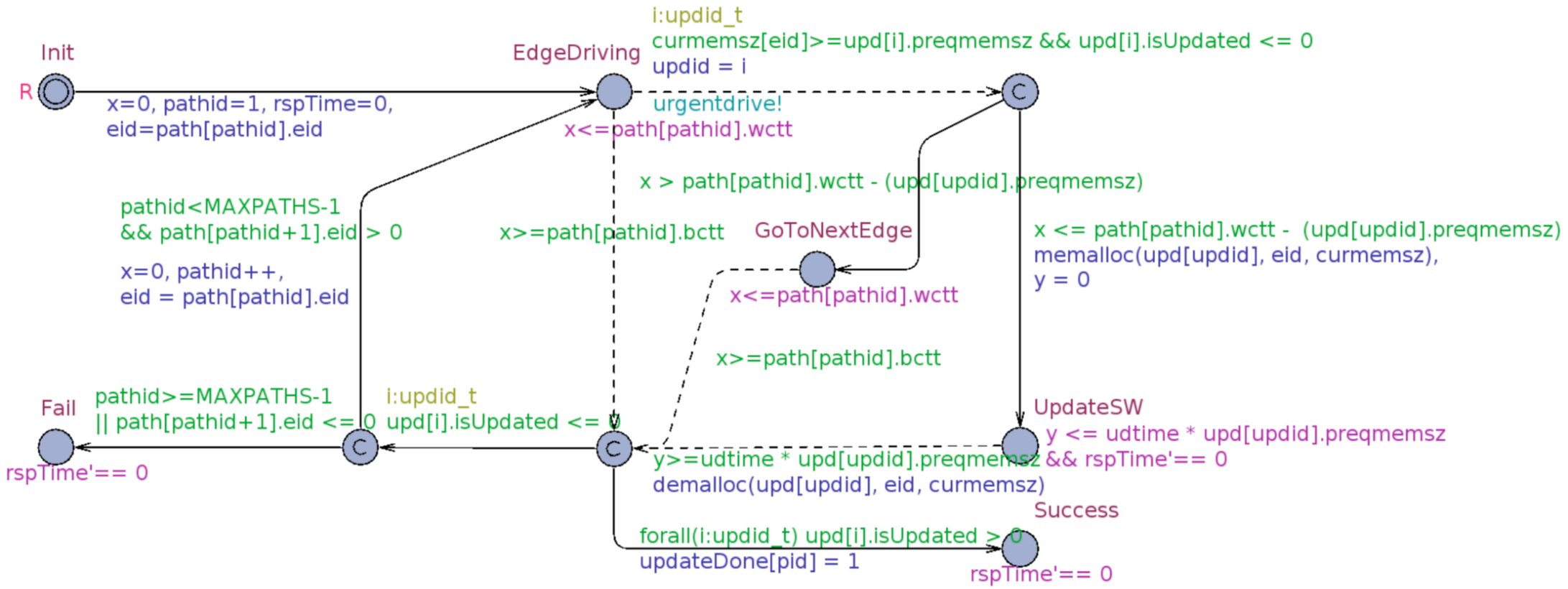


Figure: Vehicle model in stochastic timed automata

Queries (Symbolic)

Check if all updates of 3 vehicle finishes within 100 time units, (ϕ)

A <> updateDone[1] > 0 && updateDone[2] > 0 && updateDone[3] > 0 && time <= 100

Search for a strategy (σ), named $_{\text{BoundedUT}}$, that all updates for 3 vehicles are over within 100 time units.

strategy BoundedUT = control : A <> updateDone[1] > 0 && updateDone[2] > 0 && updateDone[3] > 0 && time <= 100

Check if all updates of 3 vehicle finishes within 100 time units under the strategy $_{\text{BoundedUT}}$, $(P \mid \sigma \models \phi')$

A <> updateDone[1] > 0 && updateDone[2] > 0 && updateDone[3] > 0 && time <= 100 under BoundedUT

Queries (Statistical)

Search for a strategy that all updates for 3 vehicles are over such that the update time for the vehicles are minimized,

strategy GoFast = minE(time)[<= 100] : <> updateDone[1] > 0 && updateDone[2] > 0 && updateDone[3] > 0

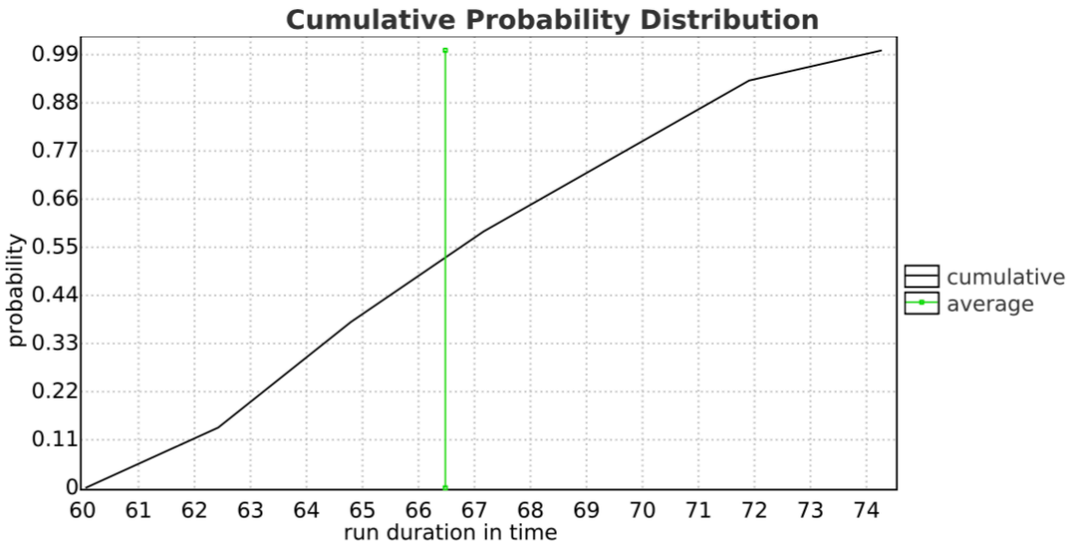
Check the possibility that all update of 3 vehicles are over

Pr[<= 100](<> updateDone[1] > 0 && updateDone[2] > 0 && updateDone[3] > 0)

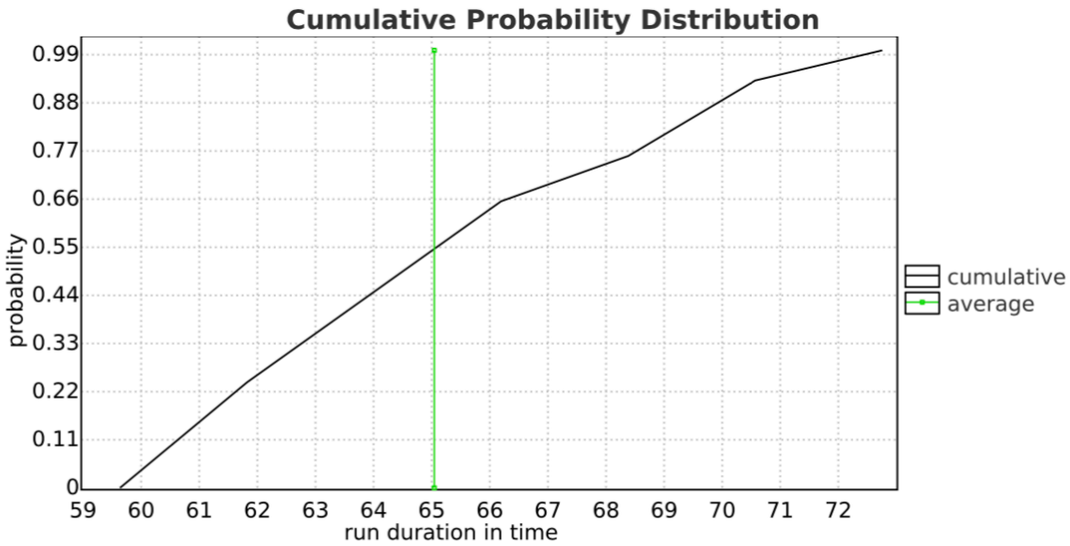
Check the possibility that all update of 3 vehicles are over under the minimum response time

Pr[<= 100](<> updateDone[1] > 0 && updateDone[2] > 0 &&updateDone[3] > 0) under GoFast

Cumulative Probability Distributions



Parameters: α=0.05, ε=0.05, bucket width=2.3704, bucket count=6
Runs: 29 in total, 29 (100%) displayed, 0 (0%) remaining
Span of displayed sample: [60.053, 74.275]
Mean from displayed sample: 66.479 ± 1.37 (95% CI)



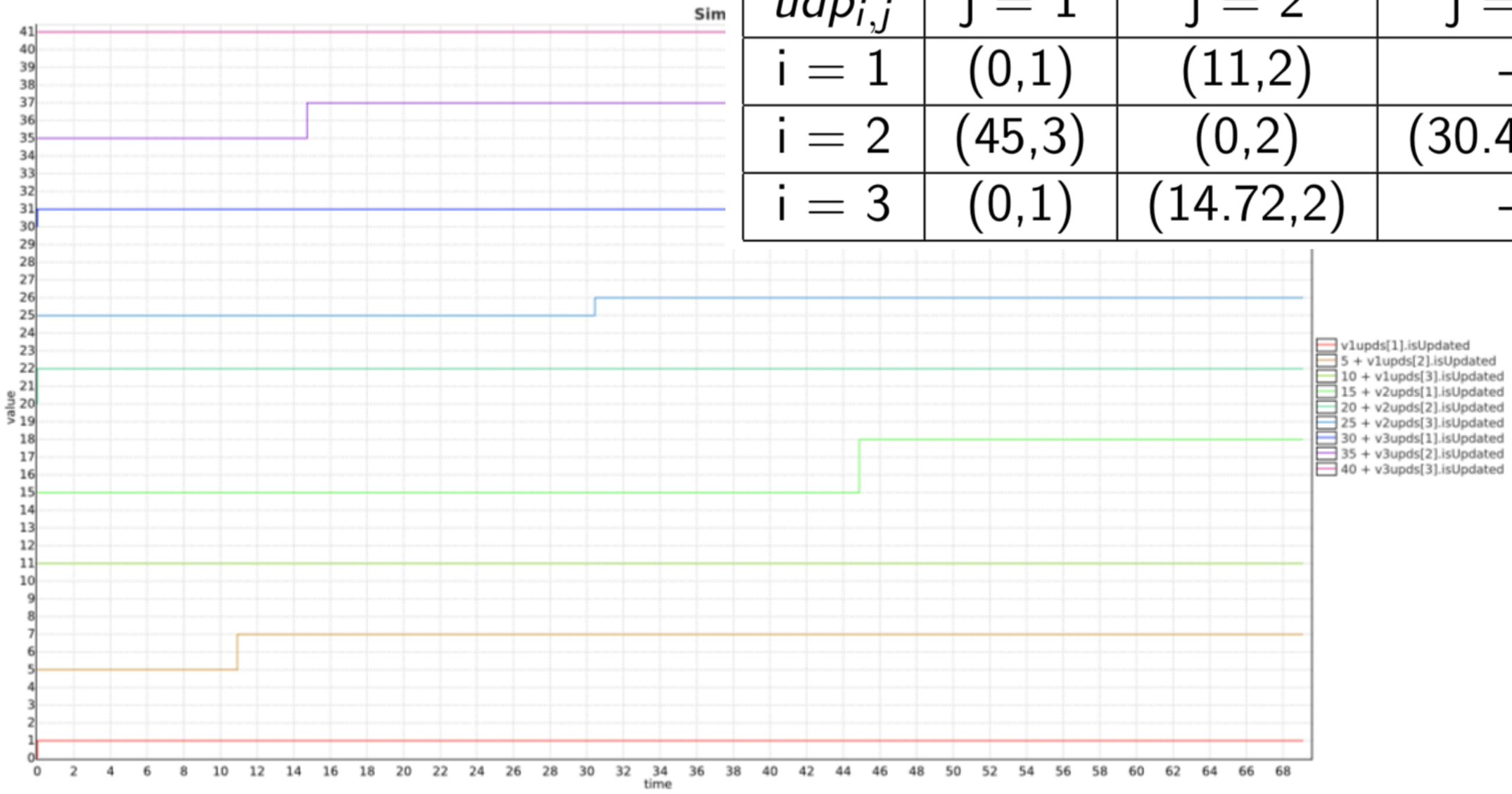
Parameters: α=0.05, ε=0.05, bucket width=2.1867, bucket count=6
Runs: 29 in total, 29 (100%) displayed, 0 (0%) remaining
Span of displayed sample: [59.636, 72.756]
Mean from displayed sample: 65.045 ± 1.44 (95% CI)

The model not under strategy finished in average 66.479 time units in between [60.053,74.275], but the model not under strategy finished in average 65.045 time units in between [59.636, 72.756]

Maximum SW Update Time

- The maximum SW update time:
    - Simulation-based estimation : 65.1763
        - E[<= 1000;1000] (max : sysrsp time)
    - Simulation-based estimation under the strategy control: 65.026
        - E[<= 1000;1000] (max : sysrsp time) under GoFast

SW Update Strategy with Mim-Update-Time Strategy



$udp_{i,j}$	j = 1	j = 2	j = 3
i = 1	(0,1)	(11,2)	-
i = 2	(45,3)	(0,2)	(30.45,1)
i = 3	(0,1)	(14.72,2)	-

Conclusions

- Machine learning is a next generation of software programming

- Decision procedure is ALSO one of machine learning techniques,

  - Instead of data, Stratego generates a constrained behavior trace (data) from a model, which is guaranteed by controller

  - Create a control algorithm that guarantees user-defined prosperities.