Under-Approximation Refinement for Timed Automata

Bachelor's Thesis

Kevin Grimm

Department of Mathematics and Computer Science
Artificial Intelligence

University of Basel

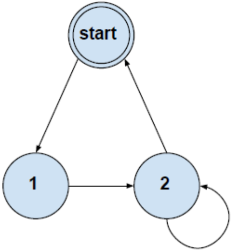
February 13, 2017

Idea

Adapt the under-approximation refinement algorithm to find errors in **real-time systems**.

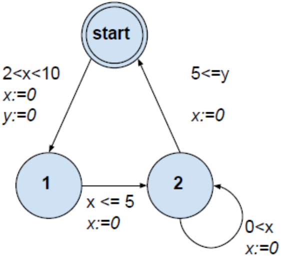
- Real-time systems can be modelled with **timed automata**.

EXAMPLE



- L: locations
- l_0: initial location
- E: edges

EXAMPLE

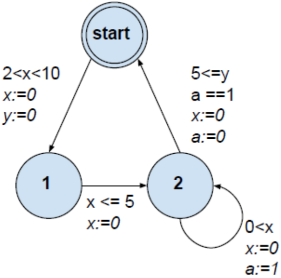


- L: locations
- l_0: initial location
- E: edges

Additionally:

- C: clock variables

EXAMPLE

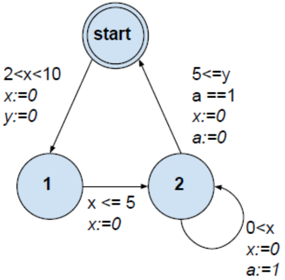


- L: locations
- l_0: initial location
- E: edges

Additionally:
- C: clock variables
- V: integer variables

EXAMPLE

Problem: idling in locations



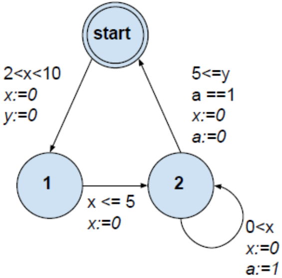
- L: locations
- l_0: initial location
- E: edges

Additionally:

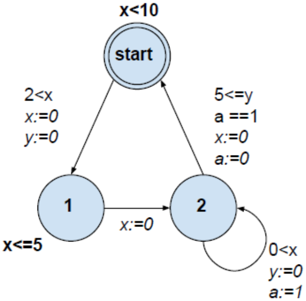
- C: clock variables
- V: integer variables

EXAMPLE

Problem: idling in locations

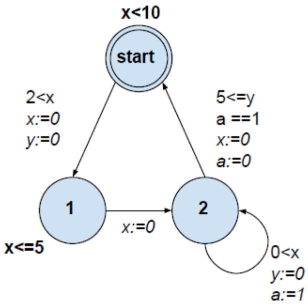


Solution: location invariants



EXAMPLE

6-tuple $\langle L, C, V, E, I, l_0 \rangle$



- L: locations
- l_0: initial location
- E: edges

Additionally:

- C: clock variables
- V: integer variables
- I: assigns location invariants

Idea

Adapt the under-approximation refinement algorithm to find **errors** in real-time systems.

- **State spaces** can be used to model the behaviour of timed automata.
- Errors are states with specific properties.

State Space

Given: $T = \langle L, C, V, E, I, l_0 \rangle$

States are triples $\langle l, v, u \rangle$, where:

- l: locations $\in L$
- v: integer valuation
- u: clock assignment

Two transition types:

- delay transition: $\langle l, v, u \rangle \xrightarrow{d} \langle l, v, u + d \rangle$
- action transition: $\langle l, v, u \rangle \xrightarrow{a} \langle l', v', u' \rangle$

Challenge: clocks are real valued

Zone Graph

Challenge: clocks are real valued
Solution: zones

- zone: conjunction of clock constraints
  Example: $z := (0 \leq x \leq 12 \wedge y == 0)$

- zone graph:
  states are triples $\langle l, v, z \rangle$
  zone transition: $\langle l, v, z \rangle \xrightarrow{a} \langle l', v', z' \rangle$

Transitions

- Zone transition: transition between states of zone graph
- Structural transition: transition induced by edges

CONCURRENT SYSTEMS

Challenge: building the product of timed automata is complex

Solution: running concurrent systems on the fly

- timed automaton is a 7-tuple $\langle L, \Sigma, C, V, E, I, l_0 \rangle$:
  - Σ contains synchronisation labels
- Labelled Edges:
  - Asynchronous transition: void label
  - Synchronous transition: synchronisation label
- States are triples $\langle l, v, z \rangle$ where l is a location valuation

Idea

Adapt the under-approximation refinement algorithm to **find** errors in real-time systems.

Use **directed model checking** (similar to planning):

- heuristic function
- search algorithm
- goal state corresponds to error state
- plan corresponds to trace of error state

Under-Approximation Refinement

- Challenge: state explosion problem
- Observation: used transitions often a small subset of all available transitions

	GBFS	UT Search
Ø A	12.74%	10.39%
Ø C	16.25%	11.58%
Ø D	11.17%	8.37%
Ø N	74.41%	71.64%
Ø M	75.63%	72.85%

- Solution: evaluate and limit applicable transitions

Definitions

- Given: $A = \langle L, \Sigma, C, V, E, I, l_0 \rangle$

  Under-approximation of A:
  $UA = \langle L, \Sigma, C, V, E', I, l_0 \rangle$, where E' is a subset of E

- Given: $M = \{A_1, A_2, ..., A_n\}$

  Under-approximation of M:
  $UM = \{UA_1, UA_2, ..., UA_n\}$

APPROACH

Idea:

- Search on under-approximations
- Refine transition set if needed

Components:

- Search algorithm
- Refinement guard
- Refinement strategy

Algorithm

- Search algorithm: GBFS

  - Always expand most promising state
  - Store explored states in closed list
  - Store successors in open list

ALGORITHM

- Refinement guard: plateau, local minimum, empty open list

Refine if:

- successors do not improve the heuristic value
- open list is empty

Algorithm

- Refinement strategy:

1. Relaxed plan of each state in closed list with minimal h
2. Allow transitions of the relaxed plans

3.a new transitions found:
  - reopen States of closed List with new transition

3.b no transitions found:
  - repeat 1. with minimal $h + 1$

3.c no transitions found and closed list completely scanned:
  if open list is not empty: return to search
  else: Allow applicable transitions of states with minimal h

Implementation

- Mcta
- h^U heuristic for relaxed plans
- Possible to use different search heuristics

Setup

- 5 Test-sets: A, C, D, N, M
- Comparison with UT and GBFS
- All tests conducted 3 times

Results for the h^U Heuristic

	runtime in *s*			used memory in *MB*			explored states			trace length		
	GBFS	**UT**	**UA**	**GBFS**	**UT**	**UA**	**GBFS**	**UT**	**UA**	**GBFS**	**UT**	**UA**
A2	**0.0**	**0.0**	**0.0**	**60**	**60**	**60**	25	**20**	**20**	21	**18**	**18**
A3	**0.0**	0.01	**0.0**	**61**	**61**	**61**	82	**27**	46	18	**17**	**17**
A4	**0.01**	0.04	**0.01**	**62**	**62**	**62**	39	**34**	131	28	**22**	23
A5	0.48	0.17	**0.08**	72	**68**	72	4027	**42**	586	47	**27**	29
A6	-	**1.0**	1.89	-	**91**	254	-	**50**	5564	-	**32**	35

Results for the h^U Heuristic

	runtime in *s*			used memory in *MB*			explored states			trace length		
	GBFS	**UT**	**UA**	**GBFS**	**UT**	**UA**	**GBFS**	**UT**	**UA**	**GBFS**	**UT**	**UA**
C1	**0.01**	**0.01**	**0.01**	**61**	**61**	**61**	429	**243**	239	67	**54**	55
C2	**0.01**	0.02	**0.01**	**61**	**61**	**61**	828	**212**	239	83	**54**	55
C3	**0.01**	0.02	**0.01**	**61**	**61**	**61**	1033	**198**	239	79	**54**	55
C4	0.15	**0.02**	0.03	64	**61**	**61**	12k	**174**	1117	112	**55**	64
C5	0.86	**0.03**	0.04	78	**61**	**61**	65k	**147**	1493	176	**61**	75
C6	5.46	**0.03**	0.05	166	**61**	62	453k	**147**	1493	432	**61**	75
C7	45.42	**0.04**	0.05	974	**61**	62	4230k	**143**	1493	924	**61**	75
C8	31.46	0.32	**0.09**	758	**62**	63	3403k	**1466**	2875	2221	**56**	161
C9	-	0.36	**0.2**	-	**62**	66	-	**1575**	6119	-	**69**	169
D1	0.05	0.11	**0.02**	**61**	**61**	**61**	1344	939	**292**	96	**88**	89
D2	3.03	**0.15**	0.65	98	**62**	67	112k	**843**	14k	220	**89**	203
D3	0.91	0.13	**0.06**	75	**62**	**62**	44k	**717**	1228	241	**89**	95
D4	6.36	**0.15**	3.31	138	**62**	94	259k	**615**	83k	410	**89**	262
D5	0.22	11.76	**0.06**	64	125	**62**	4455	87k	**720**	115	**107**	108
D6	**0.52**	-	0.59	**67**	-	**67**	12k	-	**7490**	301	-	**206**
D7	**0.82**	4.8	2.49	**70**	80	85	20k	**18k**	34k	154	**109**	128
D8	1.01	**0.63**	121.53	72	**64**	1186	23k	**1883**	1808k	259	**109**	253
D9	-	**0.59**	-	-	**64**	-	-	**1533**	-	-	**110**	-

RESULTS FOR THE h^U HEURISTIC

	runtime in *s*			used memory in *MB*			explored states			trace length		
	GBFS	**UT**	**UA**	**GBFS**	**UT**	**UA**	**GBFS**	**UT**	**UA**	**GBFS**	**UT**	**UA**
M1	0.02	0.06	**0.01**	**61**	**61**	**61**	7668	4366	**1529**	71	73	**66**
M2	0.06	0.05	**0.02**	63	**61**	**61**	18k	**2018**	3852	119	**81**	105
M3	0.06	0.39	**0.02**	63	64	**61**	19k	17k	**4794**	124	163	**93**
M4	0.14	0.5	**0.05**	68	66	**63**	46k	15k	**12k**	160	**91**	148
N1	0.05	0.09	**0.02**	62	62	**61**	9117	5191	**1880**	99	80	**68**
N2	0.14	0.09	**0.06**	66	**62**	63	23k	**3260**	8106	154	136	**103**
N3	0.27	0.41	**0.06**	69	65	**63**	43k	19k	**7117**	147	149	**87**
N4	1.08	0.46	**0.19**	88	**67**	**67**	152k	**15k**	25k	314	377	**185**

Plateau Guard

- Problem: many useless refinements

	# of refinements
Ø A	1076
Ø C	938
Ø D	97934
Ø M	3493
Ø N	7273

- Idea: ignore small plateaus and local minima
- Proposed solution: plateau guard

Plateau Guard

- Problem: many useless refinements
- Idea: ignore small plateaus and local minima
- Proposed solution: plateau guard

  - Counter for encountered plateaus and local minima
  - Refine only if counter reaches a set value
  - Reset counter to 0 after refinement

Plateau Guard

	# of refinements												
Guard	**0**	**5**	**10**	**25**	**50**	**100**	**150**	**500**	**1k**	**1.5k**	**2k**	**2.5k**	**5k**
ØA	1076	182	100	43	23	13	9	4	3	2	2	2	2
ØC	938	151	83	26	16	11	10	9	9	9	9	9	10
ØD	97934	16392	8801	3183	77	30	15	14	13	14	14	14	14
ØM	3493	643	339	146	67	43	27	14	12	12	12	11	11
ØN	7273	1250	674	283	137	65	48	20	15	13	12	11	11

Plateau Guard

	runtime in *s*												
Guard	**0**	**5**	**10**	**25**	**50**	**100**	**150**	**500**	**1k**	**1.5k**	**2k**	**2.5k**	**5k**
ØA	0.4	0.35	0.37	0.39	0.35	0.36	0.37	0.4	0.37	0.33	0.28	**0.25**	0.47
ØC	0.05	0.05	0.05	**0.04**	**0.04**	0.05	0.06	0.1	0.1	0.14	0.13	0.14	0.18
ØD	16.09	16.47	15.62	13.81	0.52	0.42	**0.21**	0.32	0.34	0.54	0.46	0.78	0.6
ØM	0.03	0.03	0.03	0.03	**0.02**	0.03	0.03	0.04	0.04	0.04	0.05	0.05	0.06
ØN	0.08	0.07	0.08	0.08	0.07	**0.06**	0.07	0.08	0.09	0.1	0.12	0.14	0.19
D8	121.53	124.97	120.03	106.4	**0.06**	0.37	0.13	0.3	0.46	0.72	0.73	1.98	0.9
D9	-	-	-	-	-	-	-	-	**2.18**	3.63	18.61	5.69	9.46

Conclusion

- Successful adaptation and implementation
- Fast and memory efficient algorithm
- Similar performance to UT
- Improvements possible with minor changes

Future Work

- Test different refinement guards
- Test different refinement strategies
- Evaluation on more diverse test-sets

Thank you for your attention