On Unary Fragments of *MTL* and *TPTL* over Timed Words.

Khushraj Madnani,S.N.Krishna, Paritosh.K.Pandya

ICTAC,2014 Bucharest, Romania

September 19, 2014

Motivation

Motivation

- *MTL* and *TPTL* is extensively studied in the literature.

Motivation

- *MTL* and *TPTL* is extensively studied in the literature.
- These Logics are extension of *LTL* that allow timing constraints to be specified along with the temporal ordering.

Motivation

- *MTL* and *TPTL* is extensively studied in the literature.
- These Logics are extension of *LTL* that allow timing constraints to be specified along with the temporal ordering.
- They exhibit considerable diversity in expressiveness and decidability properties based on restriction on modalities and type of timing constraints.

Motivation

- *MTL* and *TPTL* is extensively studied in the literature.
- These Logics are extension of *LTL* that allow timing constraints to be specified along with the temporal ordering.
- They exhibit considerable diversity in expressiveness and decidability properties based on restriction on modalities and type of timing constraints.
- In general satisfiability checking of full *MTL* and *TPTL* is undecidable.

Motivation

- *MTL* and *TPTL* is extensively studied in the literature.
- These Logics are extension of *LTL* that allow timing constraints to be specified along with the temporal ordering.
- They exhibit considerable diversity in expressiveness and decidability properties based on restriction on modalities and type of timing constraints.
- In general satisfiability checking of full *MTL* and *TPTL* is undecidable.
- Unlike *LTL*, they are more expressive with *past* operators.

Motivation

- *MTL* and *TPTL* is extensively studied in the literature.
- These Logics are extension of *LTL* that allow timing constraints to be specified along with the temporal ordering.
- They exhibit considerable diversity in expressiveness and decidability properties based on restriction on modalities and type of timing constraints.
- In general satisfiability checking of full *MTL* and *TPTL* is undecidable.
- Unlike *LTL*, they are more expressive with *past* operators.
- Thus it becomes interesting to study satisfiability checking and expressiveness for different fragments of these logics.

Presentation Flow

- Preliminaries
- Satisfiability Checking
  - $MTL[\Diamond_I, \diamondminus_I]$
  - $2 - TPTL[\Diamond_I]$
  - $MTL[\Diamond_I]$
- Expressiveness
- Conclusion
- Future Work

Preliminaries: Timed Word

Preliminaries: Timed Word

- Timed word are the models over which *MTL* Formula is being evaluated.

- Timed word are the models over which *MTL* Formula is being evaluated.
- A Finite Timed Word is defined as finite sequence of symbols along with their corresponding timestamp.

Preliminaries: Timed Word

- Timed word are the models over which *MTL* Formula is being evaluated.
- A Finite Timed Word is defined as finite sequence of symbols along with their corresponding timestamp.
- For example $\rho = ((\sigma_1, t_1), (\sigma_2, t_2), (\sigma_3, t_3), \ldots, (\sigma_n, t_n))$ where $\sigma_i \in \Sigma$(set of alphabets) ,$t_i \in R$ is a time-stamp, n is finite and $\{1, \ldots, n\}$ is called domain of ρ .

Preliminaries: Timed Word

- Timed word are the models over which *MTL* Formula is being evaluated.
- A Finite Timed Word is defined as finite sequence of symbols along with their corresponding timestamp.
- For example $\rho = ((\sigma_1, t_1), (\sigma_2, t_2), (\sigma_3, t_3), \ldots, (\sigma_n, t_n))$ where $\sigma_i \in \Sigma$(set of alphabets) ,$t_i \in R$ is a time-stamp, n is finite and $\{1, \ldots, n\}$ is called domain of ρ .
- A strictly monotonic timed word is a timed word where timestamps are strictly increasing that is $i_1 > i_2$ implies $t_{i_1} > t_{i_2}$.In general,$i_1 > i_2$ implies $t_{i_1} \geq t_{i_2}$

- *MTL* Syntax

- *MTL* Syntax

$\phi ::= x \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg \phi \mid \phi \ \mathsf{S}_I \ \phi \mid \phi \ \mathsf{U}_I \ \phi \mid \mathsf{O}\phi \mid \bar{\mathsf{O}}\phi$

where I is interval of the form $\langle x, y \rangle$, $x \in \mathcal{N} \cup \{0\}$, $y, x \in \mathcal{N} \cup \{0, \infty\}$ and $\langle ... \rangle \in \{[...], (...), [...), (...]\}$

Metric Temporal Logic

Metric Temporal Logic

- *MTL* Semantics

Metric Temporal Logic

- *MTL* Semantics

$$\rho, i \models \bigcirc\phi \iff \exists \rho, i+1 \models \phi$$

Metric Temporal Logic

- *MTL* Semantics

$$\rho, i \models O\phi \iff \exists \rho, i+1 \models \phi$$

$$\rho, i \models \bar{O}\phi \iff \exists \rho, i-1 \models \phi$$

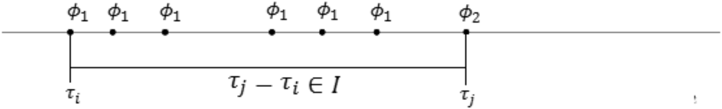
Metric Temporal Logic

- *MTL* Semantics

$$\rho, i \models \mathsf{O}\phi \iff \exists \rho, i+1 \models \phi$$

$$\rho, i \models \bar{\mathsf{O}}\phi \iff \exists \rho, i-1 \models \phi$$

$$\rho, i \models \phi_1 \mathsf{U}_I \phi_2 \iff \exists j \geq i \ \rho, j \models$$
$$\phi_2 \ and \ \tau_j - \tau_i \in I \ and \ \forall \ i \leq k < j \ \rho, k \models \phi_1$$



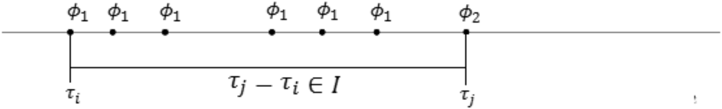
Metric Temporal Logic

- *MTL* Semantics

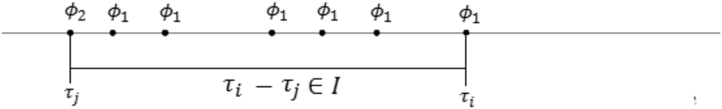
$$\rho, i \models \mathsf{O}\phi \iff \exists \rho, i+1 \models \phi$$

$$\rho, i \models \bar{\mathsf{O}}\phi \iff \exists \rho, i-1 \models \phi$$

$$\rho, i \models \phi_1 \, \mathsf{U}_I \phi_2 \iff \exists j \geq i \ \rho, j \models$$
$$\phi_2 \ \textit{and} \ \tau_j - \tau_i \in I \ \textit{and} \ \forall \ i \leq k < j \ \rho, k \models \phi_1$$



$$\rho, i \models \phi_1 \, \mathsf{S}_I \phi_2 \iff \exists j \leq i \ \rho, j \models$$
$$\phi_2 \ \textit{and} \ \tau_i - \tau_j \in I \ \textit{and} \ \forall \ i \geq k > j \ \rho, k \models \phi_1$$

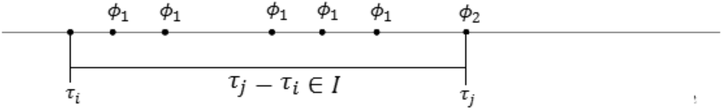


MTL contd.

MTL contd.

- Strict Operators

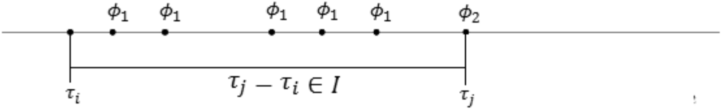
  $\rho, i \models \phi_1 \, \mathsf{U}_I \phi_2 \iff \exists j > i \; \rho, j \models \phi_2$ *and* $\tau_j - \tau_i \in I$ *and* $\forall \; i < k < j \; \rho, k \models \phi_1$



MTL contd.

- Strict Operators

  $\rho, i \models \phi_1 \mathsf{U}_I \phi_2 \iff \exists j > i \; \rho, j \models \phi_2 \; \textit{and} \; \tau_j - \tau_i \in I \; \textit{and} \; \forall \; i < k < j \; \rho, k \models \phi_1$

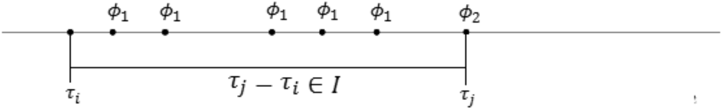


- It has been proved that strict versions of Until and Since are strictly more expressive than weak versions.

MTL contd.

- Strict Operators

$$\rho, i \models \phi_1 \, \mathsf{U}_I \phi_2 \iff \exists j > i \ \rho, j \models \phi_2 \ and \ \tau_j - \tau_i \in I \ and \ \forall \ i < k < j \ \rho, k \models \phi_1$$



- It has been proved that strict versions of Until and Since are strictly more expressive than weak versions.
- Unary Operators

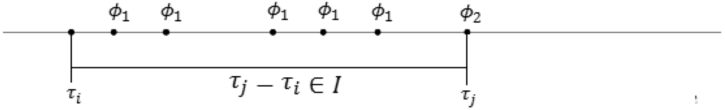
$$\Diamond_I \phi = \top \, \mathsf{U}_I \phi; \ \Box_I \phi = \neg \Diamond_I(\neg \phi); \ \diamondminus_I \phi = \top \ \mathsf{S}_I \ \phi$$

Strict versions of these operators can be defined similarly.

MTL contd.

- Strict Operators

$$\rho, i \models \phi_1 \, \mathsf{U}_I \phi_2 \iff \exists j > i \;\; \rho, j \models \phi_2 \; \textit{and} \; \tau_j - \tau_i \in I \; \textit{and} \; \forall \; i < k < j \;\; \rho, k \models \phi_1$$



- It has been proved that strict versions of Until and Since are strictly more expressive than weak versions.
- Unary Operators

$$\Diamond_I \phi = \top \, \mathsf{U}_I \phi; \; \Box_I \phi = \neg \Diamond_I (\neg \phi); \; \diamondminus_I \phi = \top \;\; \mathsf{S}_I \; \phi$$

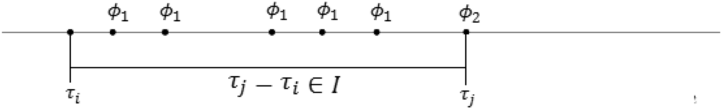
Strict versions of these operators can be defined similarly.

- Subclasses:

MTL contd.

- Strict Operators

$$\rho, i \models \phi_1 \, \mathsf{U}_I \phi_2 \iff \exists j > i \ \rho, j \models \phi_2 \ \textit{and} \ \tau_j - \tau_i \in I \ \textit{and} \ \forall \ i < k < j \ \rho, k \models \phi_1$$



- It has been proved that strict versions of Until and Since are strictly more expressive than weak versions.
- Unary Operators

$$\Diamond_I \phi = \top \, \mathsf{U}_I \phi; \ \Box_I \phi = \neg \Diamond_I (\neg \phi); \ \Diamondminus_I \phi = \top \ \mathsf{S}_I \ \phi$$

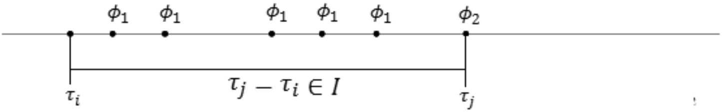
Strict versions of these operators can be defined similarly.

- Subclasses:
- By restricting set of allowed intervals. e.g. *MITL*.

MTL contd.

- Strict Operators

  $$\rho, i \models \phi_1 \, \mathsf{U}_I \phi_2 \iff \exists j > i \ \rho, j \models \phi_2 \ \textit{and} \ \tau_j - \tau_i \in I \ \textit{and} \ \forall \ i < k < j \ \rho, k \models \phi_1$$



- It has been proved that strict versions of Until and Since are strictly more expressive than weak versions.
- Unary Operators

  $$\Diamond_I \phi = \top \, \mathsf{U}_I \phi; \ \Box_I \phi = \neg \Diamond_I (\neg \phi); \ \diamondminus_I \phi = \top \ \ \mathsf{S}_I \ \phi$$

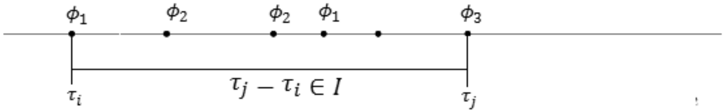
  Strict versions of these operators can be defined similarly.
- Subclasses:
- By restricting set of allowed intervals. e.g. *MITL*.
- By restricting set of operators. e.g. *MTL*$[\mathsf{U}_I]$.

TPTL Syntax

$$\phi ::= x \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg \phi \mid \phi \,\mathsf{S}\, \phi \mid \phi \,\mathsf{U}\, \phi \mid \mathsf{O}\phi \mid \bar{\mathsf{O}}\phi \mid y.\varphi \mid y \in I$$

where y is a clock (freeze) variable. Note, all the strict and unary operators can be defined similarly as before.

- Note that the truth of the formula is defined at a point i in a timed word ρ with valuation of the freeze variables ν. Thus the model is ρ, i, ν.
- All the unary and strict modal operators can be defined similarly.
- Following is the model of the formula $x.\Diamond(\phi_1 \wedge \Diamond(\phi_2 \wedge \Diamond(\phi_3 \wedge x \in I)))$



Deterministic Two Counter Machine

Deterministic Two Counter Machine

- A Two counter machine can be defined as 4 tuple $M = (P, C, D, I)$, where
  - P is a program counter whose value is bounded $value(P) \in 0, 1, \ldots, n$ where $n \in N$.
  - C, D takes value from Z.
  - I is a finite set of instructions $I = \{I_0, I_1, \ldots, I_n\}.I_n$ being the halt instruction.

Deterministic Two Counter Machine

- A Two counter machine can be defined as 4 tuple $M = (P, C, D, I)$, where
  - P is a program counter whose value is bounded $value(P) \in 0, 1, \ldots, n$ where $n \in N$.
  - C, D takes value from Z.
  - I is a finite set of instructions $I = \{I_0, I_1, \ldots, I_n\}.I_n$ being the halt instruction.
- Instructions are of 3 types:

Deterministic Two Counter Machine

- A Two counter machine can be defined as 4 tuple $M = (P, C, D, I)$, where
  - P is a program counter whose value is bounded $value(P) \in 0, 1, \ldots, n$ where $n \in N$.
  - C, D takes value from Z.
  - I is a finite set of instructions $I = \{I_0, I_1, \ldots, I_n\}.I_n$ being the halt instruction.
- Instructions are of 3 types:
  - $DEC(X)$

- A Two counter machine can be defined as 4 tuple $M = (P, C, D, I)$, where
  - P is a program counter whose value is bounded $value(P) \in 0, 1, \ldots, n$ where $n \in N$.
  - C, D takes value from Z.
  - I is a finite set of instructions $I = \{I_0, I_1, \ldots, I_n\}.I_n$ being the halt instruction.
- Instructions are of 3 types:
  - $DEC(X)$
  - $INC(X)$

Deterministic Two Counter Machine

- A Two counter machine can be defined as 4 tuple $M = (P, C, D, I)$, where
  - P is a program counter whose value is bounded $value(P) \in 0, 1, \ldots, n$ where $n \in N$.
  - C, D takes value from Z.
  - I is a finite set of instructions $I = \{I_0, I_1, \ldots, I_n\}$. I_n being the halt instruction.
- Instructions are of 3 types:
  - $DEC(X)$
  - $INC(X)$
  - IF $X == 0$ then $JMP(Y)$ else continue

Deterministic Two Counter Machine

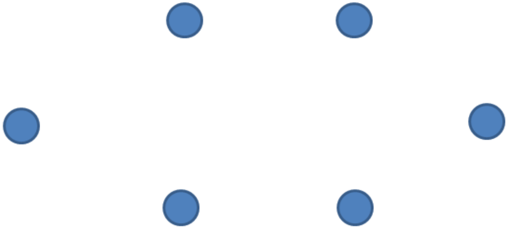
- A Two counter machine can be defined as 4 tuple $M = (P, C, D, I)$, where
  - P is a program counter whose value is bounded $value(P) \in 0, 1, \ldots, n$ where $n \in N$.
  - C, D takes value from Z.
  - I is a finite set of instructions $I = \{I_0, I_1, \ldots, I_n\}.I_n$ being the halt instruction.
- Instructions are of 3 types:
  - $DEC(X)$
  - $INC(X)$
  - IF $X == 0$ then $JMP(Y)$ else continue
- Value of Program counter and both the counters, C and D, $< i, c, d >$ defines the state of the machine

Deterministic Two Counter Machine

- A Two counter machine can be defined as 4 tuple $M = (P, C, D, I)$, where
  - P is a program counter whose value is bounded $value(P) \in 0, 1, \ldots, n$ where $n \in N$.
  - C, D takes value from Z.
  - I is a finite set of instructions $I = \{I_0, I_1, \ldots, I_n\}.I_n$ being the halt instruction.
- Instructions are of 3 types:
  - $DEC(X)$
  - $INC(X)$
  - IF $X == 0$ then $JMP(Y)$ else continue
- Value of Program counter and both the counters, C and D, $< i, c, d >$ defines the state of the machine
- Whether a unique run of a DTCM is halting or not is undecidable.

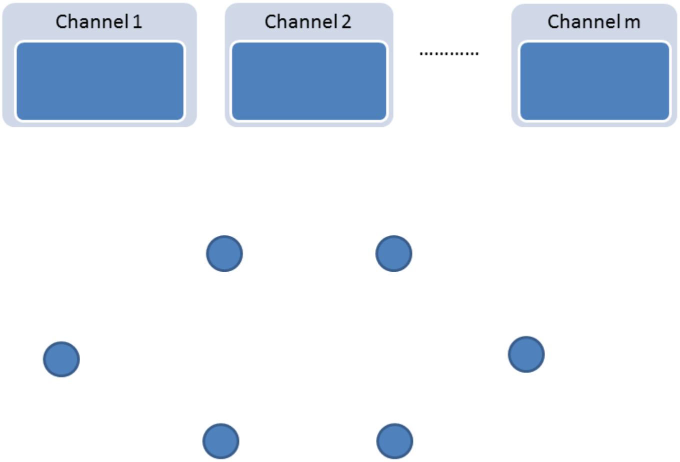
Channel Machines

5-Tuple $\{S,$



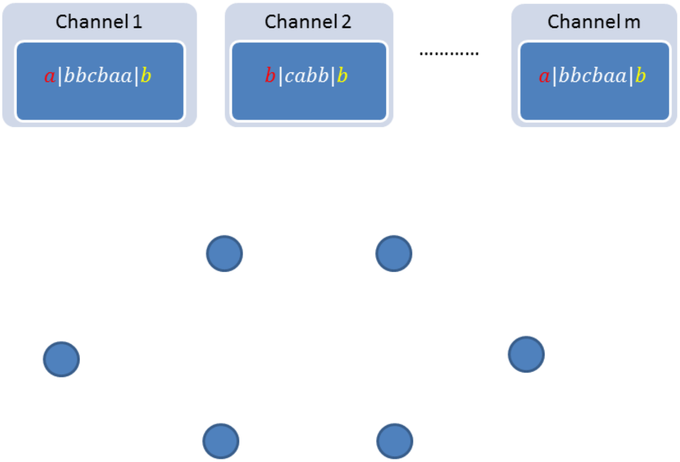
Channel Machines

5-Tuple $\{S, C$



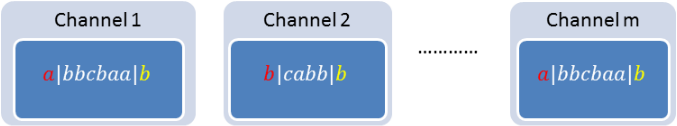
Channel Machines

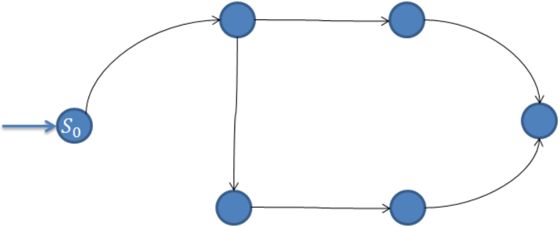
5-Tuple $\{S, C, M$



Channel Machines

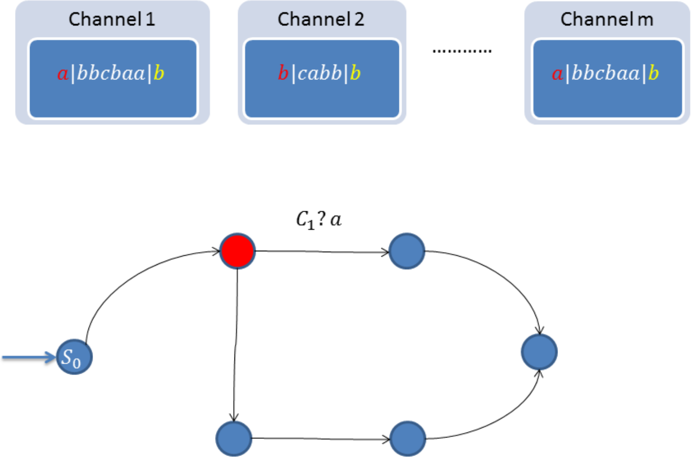
5-Tuple $\{S, C, M, T, s_0\}$





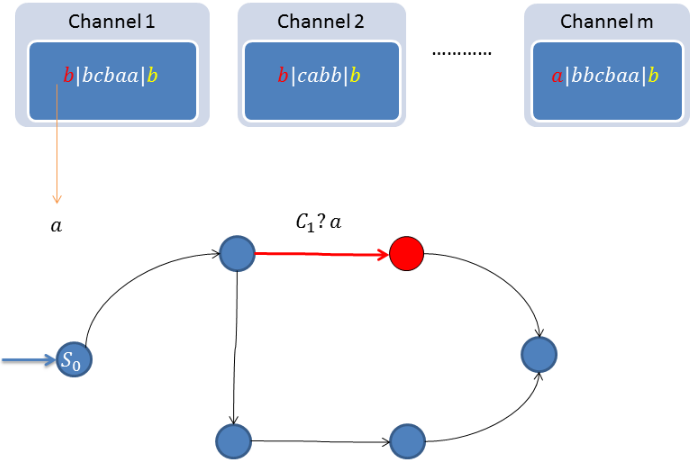
Channel Machines

5-Tuple $\{S, C, M, T, s_0\}$ Sample run $C_1?a$

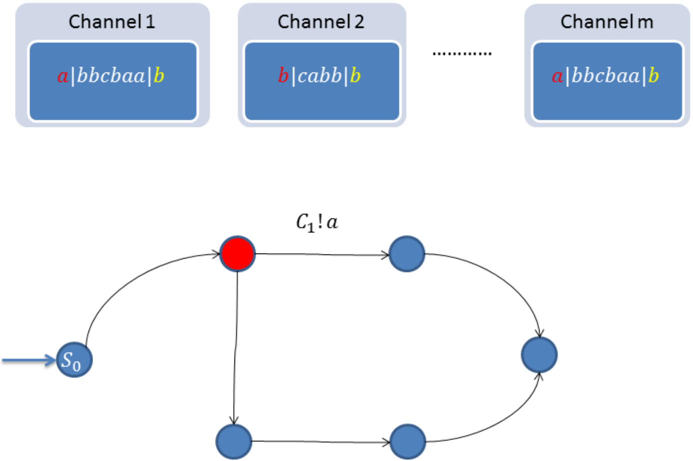


Channel Machines

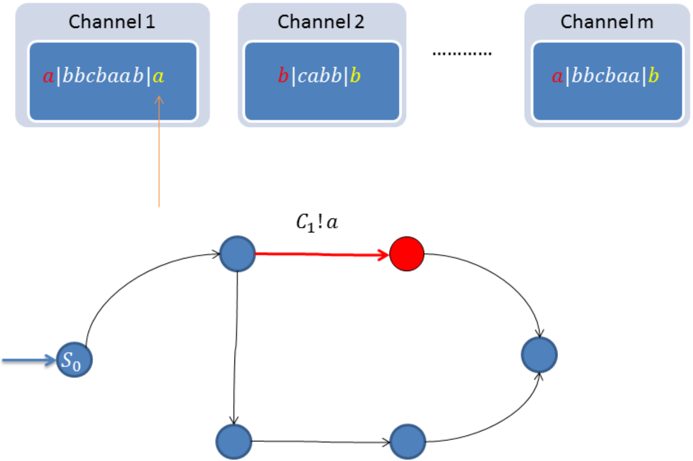
5-Tuple $\{S, C, M, T, s_0\}$ Sample run $C_1?a$



5-Tuple $\{S, C, M, T, s_0\}$ Sample run $C_1!a$

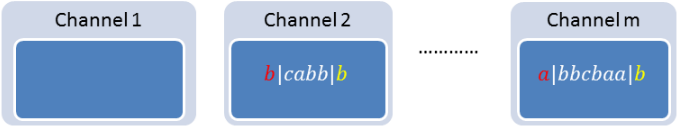


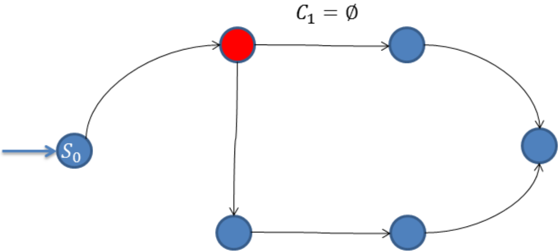
5-Tuple $\{S, C, M, T, s_0\}$ Sample run $C_1!a$



Channel Machines

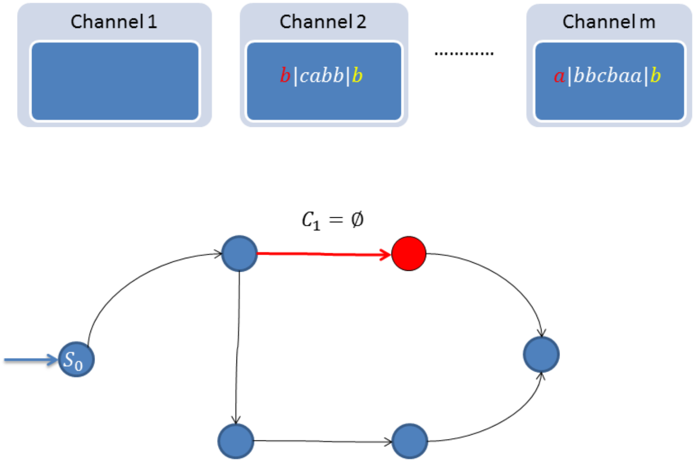
5-Tuple $\{S, C, M, T, s_0\}$ Sample run $C_1 = \emptyset$





Channel Machines

5-Tuple $\{S, C, M, T, s_0\}$ Sample run $C_1 = \emptyset$

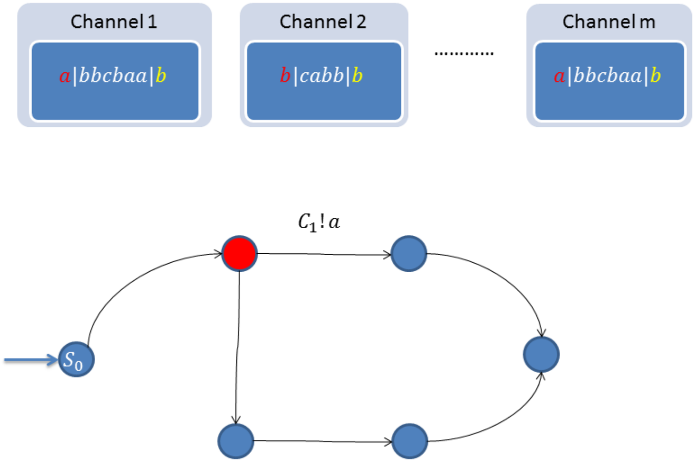


Channel Machines with Insertion Errors - *ICMET*.

Channel Machines with Insertion Errors - *ICMET*. Example Run.

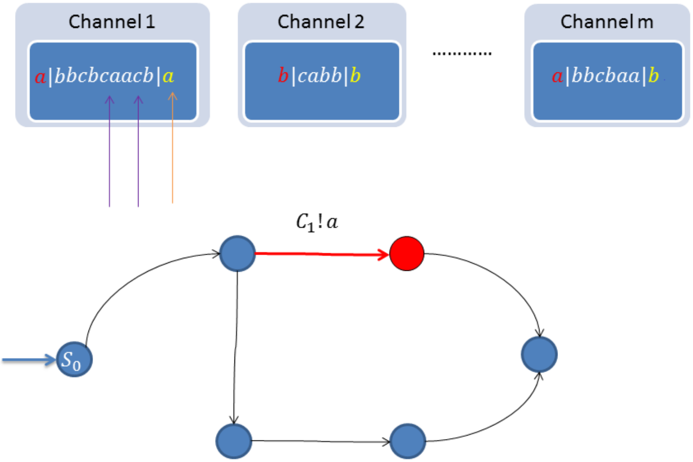
Insertion Error Channel Machines with Emptiness Testing

Channel Machines with Insertion Errors - *ICMET*. Example Run.



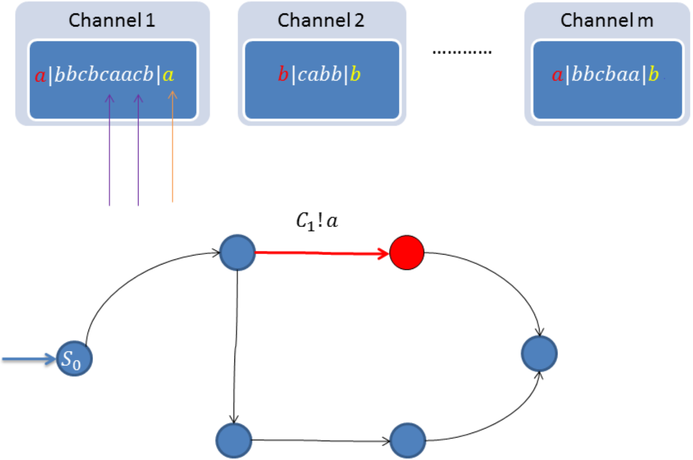
Insertion Error Channel Machines with Emptiness Testing

Channel Machines with Insertion Errors - *ICMET*. Example Run.



Insertion Error Channel Machines with Emptiness Testing

To verify whether a state is reachable from the initial state in ICMET is decidable with Non Primitive Recursive complexity.



EF Games

Description:

EF Games

Description:

- 2 Player - (Spoiler and Duplicator), played on pair of words for n rounds.

Description:

- 2 Player - (Spoiler and Duplicator), played on pair of words for n rounds.
- Configuration in a game is defined as pair of points (i_0, i_1) where $i_0 \in dom(\rho_0)$ and $i_1 \in dom(\rho_1)$

EF Games

Description:

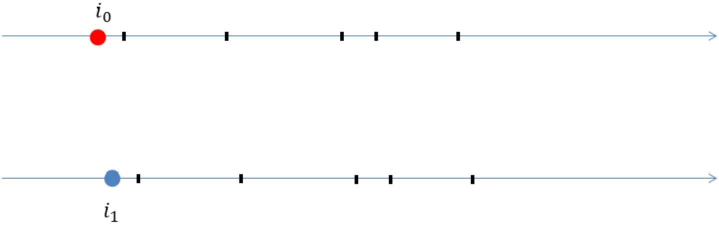
- 2 Player - (Spoiler and Duplicator), played on pair of words for n rounds.
- Configuration in a game is defined as pair of points (i_0, i_1) where $i_0 \in dom(\rho_0)$ and $i_1 \in dom(\rho_1)$
- 0-round. Spoiler chooses to be at one of the words, say, ρ_0. Duplicator has to be at ρ_1. The starting configuration is $(1, 1)$. If $(\rho_0, 1) \neq (\rho_1, 1)$ Spoiler wins the zero round game. Else Continue.

Description:

- 2 Player - (Spoiler and Duplicator), played on pair of words for n rounds.
- Configuration in a game is defined as pair of points (i_0, i_1) where $i_0 \in dom(\rho_0)$ and $i_1 \in dom(\rho_1)$
- 0-round. Spoiler chooses to be at one of the words, say, ρ_0. Duplicator has to be at ρ_1. The starting configuration is $(1, 1)$. If $(\rho_0, 1) \neq (\rho_1, 1)$ Spoiler wins the zero round game. Else Continue.
- Until Move is played in 2 parts:

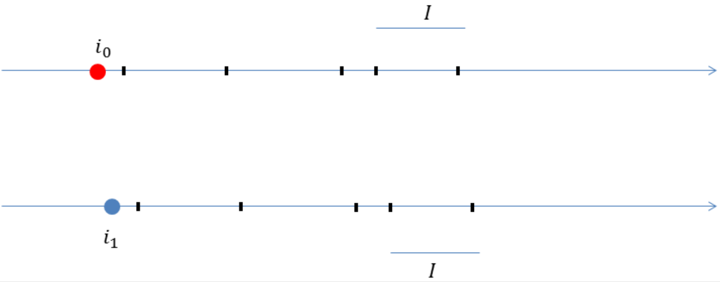
EF Games - U Move

$\Diamond_I - part$:

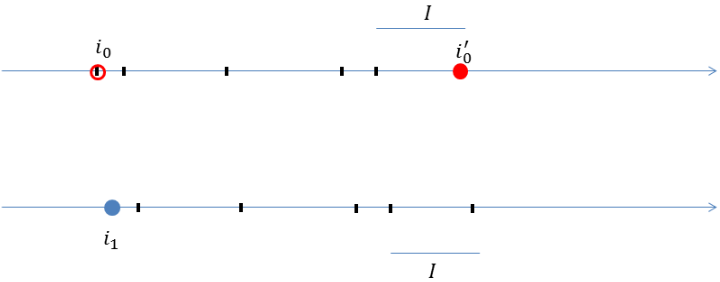


EF Games - U Move

$\Diamond_I - part$

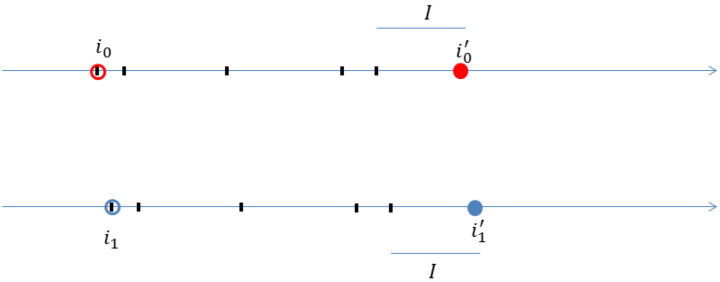


$\Diamond_I - part$

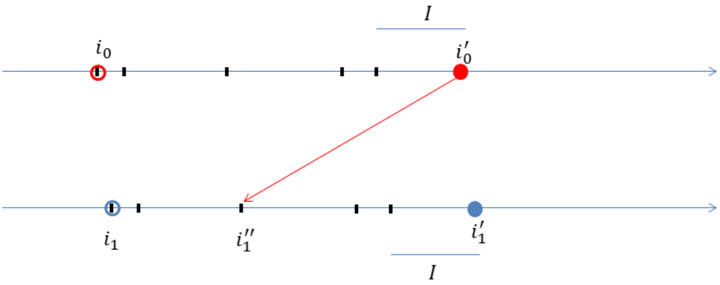


EF Games - U Move

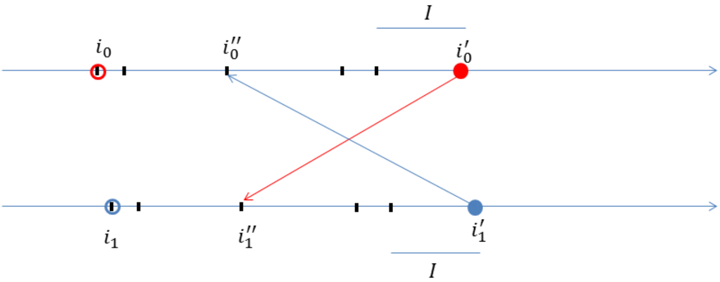
$\Diamond_I - part$



U $-$ *part*



U $-$ *part*



EF Games

- **Game equivalence:** $(\rho_0, i_0) \approx_k (\rho_1, i_1)$ iff for every k-round $MTL[U_I, S_I]$ EF-game over the words ρ_0, ρ_1 starting from the configuration (i_0, i_1), the Duplicator always has a winning strategy.

EF Games

- **Game equivalence:** $(\rho_0, i_0) \approx_k (\rho_1, i_1)$ iff for every k-round $MTL[U_I, S_I]$ EF-game over the words ρ_0, ρ_1 starting from the configuration (i_0, i_1), the Duplicator always has a winning strategy.
- **Formula equivalence:** $(\rho_0, i_0) \equiv_k (\rho_1, i_1)$ iff for every $MTL[U_I, S_I]$ formula ϕ of modal depth $\leq k$, $\rho_0, i_0 \models \phi \iff \rho_1, i_1 \models \phi$

EF Games

- **Game equivalence:** $(\rho_0, i_0) \approx_k (\rho_1, i_1)$ iff for every k-round $MTL[U_I, S_I]$ EF-game over the words ρ_0, ρ_1 starting from the configuration (i_0, i_1), the Duplicator always has a winning strategy.
- **Formula equivalence:** $(\rho_0, i_0) \equiv_k (\rho_1, i_1)$ iff for every $MTL[U_I, S_I]$ formula ϕ of modal depth $\leq k$, $\rho_0, i_0 \models \phi \iff \rho_1, i_1 \models \phi$
- **EF Theorem of *MTL***: Game equivalence $\equiv$ Formula equivalence.

As tool for comparing expressiveness.

EF Theorem as Tool

As tool for comparing expressiveness.

- Choose a formula φ in logic L_1 and a number n.

EF Theorem as Tool

As tool for comparing expressiveness.

- Choose a formula φ in logic L_1 and a number n.
- Let spoiler choose a pair of words ρ_0, ρ_1 such that $\rho_0 \models \varphi$ and $\rho_1 \models \neg\varphi$.

As tool for comparing expressiveness.

- Choose a formula φ in logic L_1 and a number n.
- Let spoiler choose a pair of words ρ_0, ρ_1 such that $\rho_0 \models \varphi$ and $\rho_1 \models \neg\varphi$.
- Play n round L_2 EF Game. If the duplicator wins then L_2 doesn't have an equivalent formula φ.

Related Work

Related Work

- Satisfiability problem for $MTL[\mathsf{U}_I, \mathsf{S}_I]$ is undecidable. [Alur *et al.*]

Related Work

- Satisfiability problem for $MTL[\mathrm{U}_I, \mathrm{S}_I]$ is undecidable. [Alur *et al.*]
- Satisfiability Checking for *MITL* is decidable with *EXPSPACE* complexity. [Alur *et al.*]

Related Work

- Satisfiability problem for $MTL[\mathrm{U}_I, \mathrm{S}_I]$ is undecidable. [Alur *et al.*]
- Satisfiability Checking for *MITL* is decidable with *EXPSPACE* complexity. [Alur *et al.*]
- Satisfiability Checking for $MTL[\mathrm{U}_I]$ is decidable.[Ouaknine *et al.*]

Related Work

- Satisfiability problem for $MTL[\mathsf{U}_I, \mathsf{S}_I]$ is undecidable. [Alur *et al*.]
- Satisfiability Checking for *MITL* is decidable with *EXPSPACE* complexity. [Alur *et al*.]
- Satisfiability Checking for $MTL[\mathsf{U}_I]$ is decidable.[Ouaknine *et al*.]
- *TPTL* is strictly more expressive than *MTL*.[*Bouyer et al*.]

Related Work

- Satisfiability problem for $MTL[\mathsf{U}_I, \mathsf{S}_I]$ is undecidable. [Alur *et al*.]
- Satisfiability Checking for *MITL* is decidable with *EXPSPACE* complexity. [Alur *et al*.]
- Satisfiability Checking for $MTL[\mathsf{U}_I]$ is decidable.[Ouaknine *et al*.]
- *TPTL* is strictly more expressive than *MTL*.[*Bouyer et al*.]
- Satisfiability Cheking for $MITL[\Diamond_b, \diamondminus_b]$ is *NEXPTIME*-complete. [Pandya *et al*.]

Related Work

- Satisfiability problem for $MTL[\mathsf{U}_I, \mathsf{S}_I]$ is undecidable. [Alur *et al.*]
- Satisfiability Checking for *MITL* is decidable with *EXPSPACE* complexity. [Alur *et al.*]
- Satisfiability Checking for $MTL[\mathsf{U}_I]$ is decidable.[Ouaknine *et al.*]
- *TPTL* is strictly more expressive than *MTL*.[*Bouyer et al.*]
- Satisfiability Cheking for $MITL[\Diamond_b, \diamondminus_b]$ is *NEXPTIME*-complete. [Pandya *et al.*]
- For $MITL[\Diamond_\infty, \diamondminus_\infty]$ is *NP*-complete.[Pandya *et al.*]

Our Result

Our Result

- Satisfiability Checking for $MTL[\Diamond_I, \Diamond\!\!\!-_I]$ is undecidable.

Our Result

- Satisfiability Checking for $MTL[\Diamond_I, \diamondminus_I]$ is undecidable.
- For $MTL[\Diamond_I]$ is non primitive recursive.

Our Result

- Satisfiability Checking for $MTL[\Diamond_I, \diamondminus_I]$ is undecidable.
- For $MTL[\Diamond_I]$ is non primitive recursive.
- For 2 clock $TPTL[F]$ is undecidable.

Our Result

- Satisfiability Checking for $MTL[\Diamond_I, \overleftarrow{\Diamond}_I]$ is undecidable.
- For $MTL[\Diamond_I]$ is non primitive recursive.
- For 2 clock $TPTL[F]$ is undecidable.
- Expressiveness Picture of Unary MTL.

Presentation Flow

- Preliminaries
- Satisfiability Checking
  - $MTL[\Diamond_I, \ominus_I]$
  - $2 - TPTL[\Diamond_I]$
  - $MTL[\Diamond_I]$
- Expressiveness
- Conclusion
- Future Work

Segment of Timed Word Showing Encoding of Increment Operation

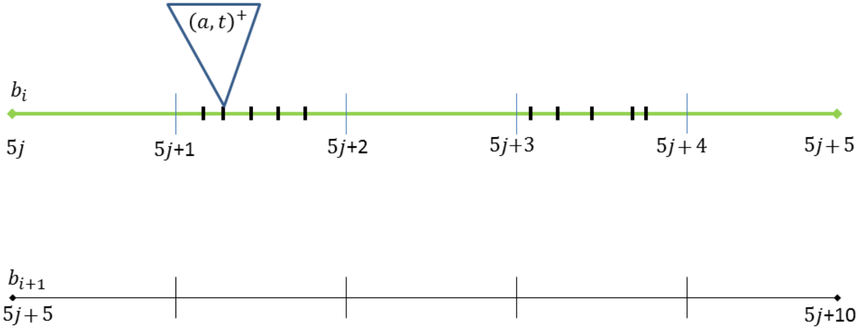


Figure: Run showing increment d

Segment of Timed Word Showing Encoding of Increment Operation

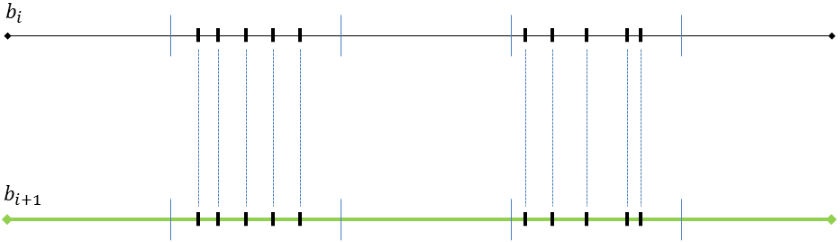


Figure: Run showing increment d

Segment of Timed Word Showing Encoding of Increment Operation

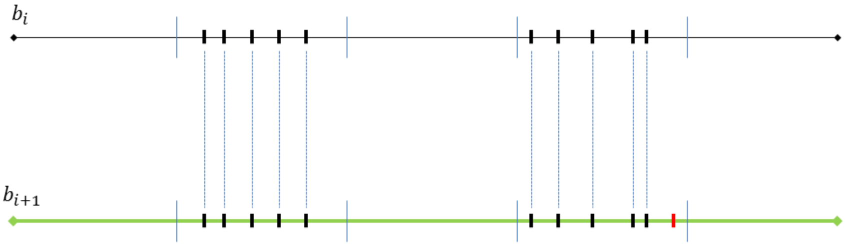


Figure: Run showing increment *d*

$MTL[\Diamond_I, \Diamond\!\!\!-_I]$ Satisfiability

- We will discuss formula for only increment operation.

- We will discuss formula for only increment operation.
- $\phi^{inc_c} \;=\; \Box[b_i \Rightarrow (COPY_C \wedge \Diamond_{[5,5]}(b_{i+1}) \wedge INC_D)]$

$MTL[\Diamond_I, \diamondminus_I]$ Satisfiability

- We will discuss formula for only increment operation.
- $\phi^{inc_c} \;=\; \Box[b_i \Rightarrow (COPY_C \wedge \Diamond_{[5,5]}(b_{i+1}) \wedge INC_D)]$
- $COPY_C$:

$$COPY_C = \Box_{[1,2]}((a \Rightarrow \Diamond_{[5,5]} a)) \wedge \Box_{[6,7]}((a \Rightarrow \diamondminus_{[5,5]} a)).$$

$MTL[\Diamond_I, \overleftarrow{\Diamond}_I]$ Satisfiability

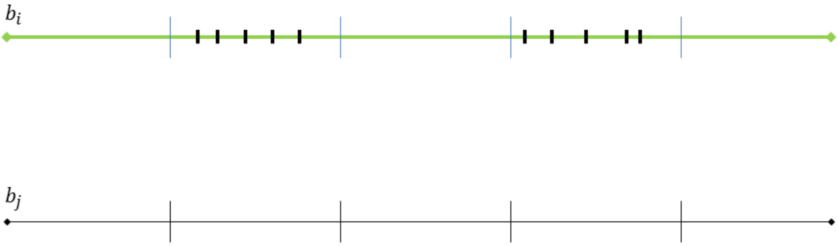
- We will discuss formula for only increment operation.
- $\phi^{inc_c} \;=\; \Box[b_i \Rightarrow (COPY_C \wedge \Diamond_{[5,5]}(b_{i+1}) \wedge INC_D)]$
- $COPY_C$:

$$COPY_C = \Box_{[1,2]}((a \Rightarrow \Diamond_{[5,5]} a)) \wedge \Box_{[6,7]}((a \Rightarrow \overleftarrow{\Diamond}_{[5,5]} a)).$$

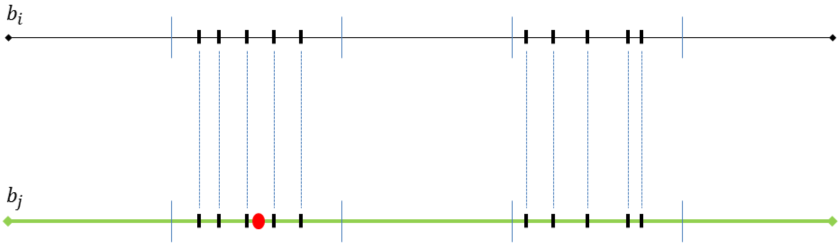
- INC_D:

$$INC_D = \Box_{[3,4]}(a \Rightarrow \Diamond_{[5,5]} a) \wedge \Box_{[8,9]}(((a \wedge \Diamond_{(0,1)}(a)) \Rightarrow \overleftarrow{\Diamond}_{[5,5]}(a)) \wedge (a \wedge \neg\Diamond_{[0,1]}(a)) \Rightarrow \neg\overleftarrow{\Diamond}_{[5,6)} a).$$

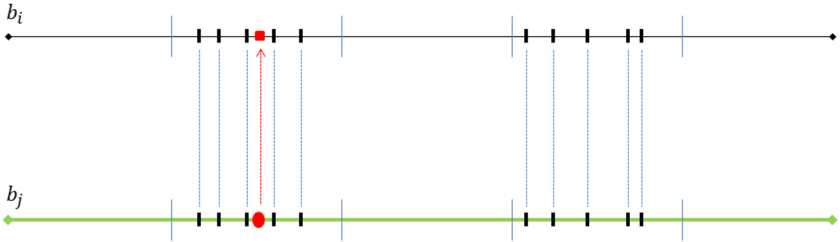
Correctness Idea



Correctness Idea



Correctness Idea



Presentation Flow

- Preliminaries
- Satisfiability Checking
  - $MTL[\Diamond_I, \Diamond\!\!\!-_I]$
  - $2 - TPTL[\Diamond_I]$
  - $MTL[\Diamond_I]$
- Expressiveness
- Conclusion
- Future Work

1. In the previous logic past was helping in ensuring the precise copying of all the non-last a's. Here we try to specify similar restriction using two clocks.

$$COPY_1 = \Box x.[(a \wedge \Diamond(a \wedge x \in (0,1))) \Rightarrow \Diamond(a \wedge x \in [5,5])].$$

$$COPY_2 = \Box x.[(a \wedge \Diamond(a \wedge x \in (0,1))) \Rightarrow (\Diamond y.(a \wedge x \in (0,1) \wedge \neg\Diamond(a \wedge x \in (5,\infty) \wedge y \in (0,5))))].$$

This is the most important constraint in the encoding which results in undecidability.

$2 - TPTL[\Diamond_I]$ Satisfiability

$2-TPTL[\Diamond_I]$ Satisfiability

- We will discuss formula for only increment operation.

- We will discuss formula for only increment operation.
- $\phi^{inc_c} \ = \ \Box[b_i \Rightarrow (COPY_C \wedge \Diamond(b_{i+1} \wedge x \in [5,5]) \wedge INC_D)]$

- We will discuss formula for only increment operation.
- $\phi^{inc_c} \;=\; \Box[b_i \Rightarrow (COPY_C \wedge \Diamond(b_{i+1} \wedge x \in [5,5]) \wedge INC_D)]$
- $COPY_C$:

$$COPY_C = \Box(x \in [1,2] \Rightarrow y.(a \wedge \neg\Diamond(a \wedge y \in (0,1))) \Rightarrow \Diamond y.(a \wedge x \in [6,7] \wedge \neg\Diamond(a \wedge y \in (0,1))))$$

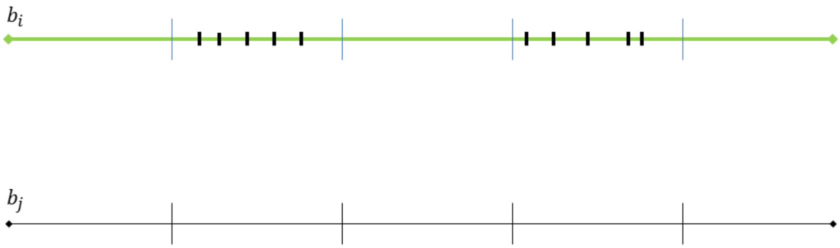
- We will discuss formula for only increment operation.
- $\phi^{inc_c} \;=\; \Box[b_i \Rightarrow (COPY_C \wedge \Diamond(b_{i+1} \wedge x \in [5,5]) \wedge INC_D)]$
- $COPY_C$:

$$COPY_C = \Box(x \in [1,2] \Rightarrow y.(a \wedge \neg\Diamond(a \wedge y \in (0,1))) \Rightarrow \Diamond y.(a \wedge x \in [6,7] \wedge \neg\Diamond(a \wedge y \in (0,1))))$$

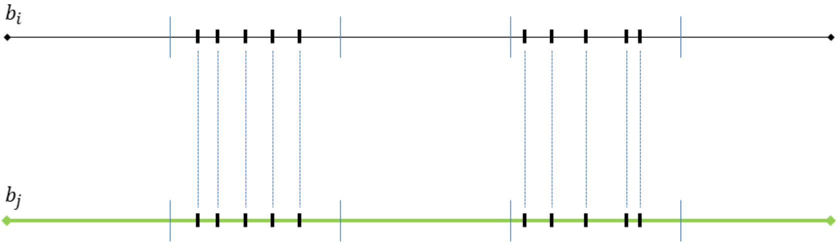
- INC_D:

$$INC_D = \Box(x \in [3,4] \Rightarrow y.(a \wedge \neg\Diamond(a \wedge y \in (0,1))) \Rightarrow \Diamond y.(a \wedge x \in [8,9] \wedge \Diamond(a \wedge y \in (0,1))\neg\Diamond(a \wedge \Diamond(a \wedge y \in (0,1)))))$$

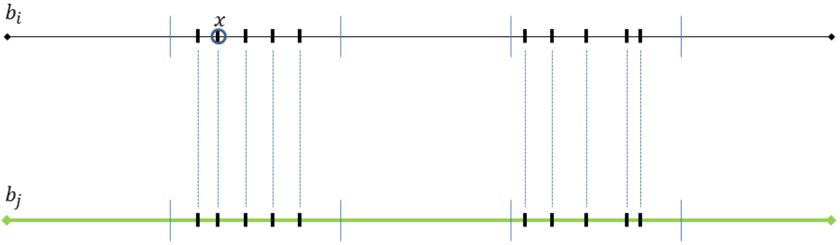
Correctness Idea



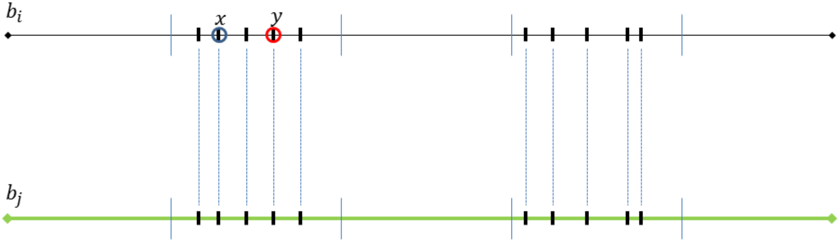
Correctness Idea



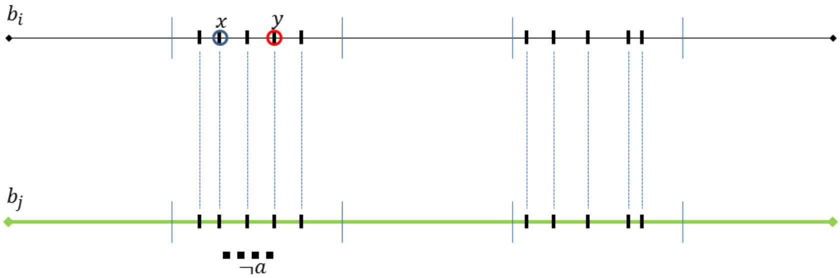
Correctness Idea



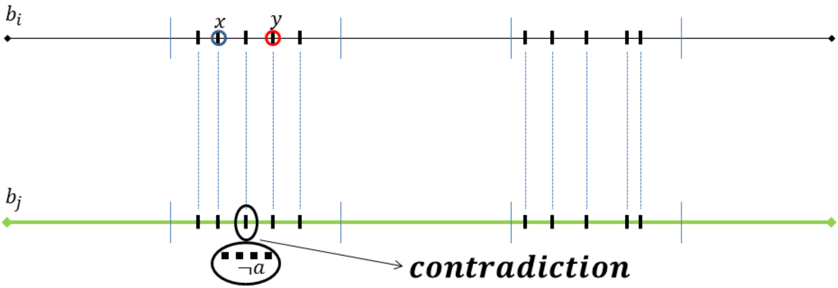
Correctness Idea



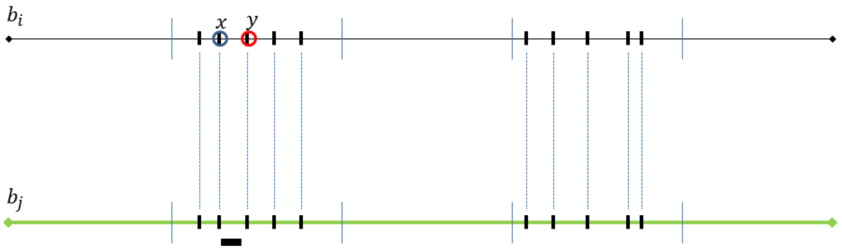
Correctness Idea



Correctness Idea



Correctness Idea



Presentation Flow

- Preliminaries
- Satisfiability Checking
  - $MTL[\Diamond_I, \ominus_I]$
  - $2 - TPTL[\Diamond_I]$
  - $MTL[\Diamond_I]$
- Expressiveness
- Conclusion
- Future Work

Encoding Configuration

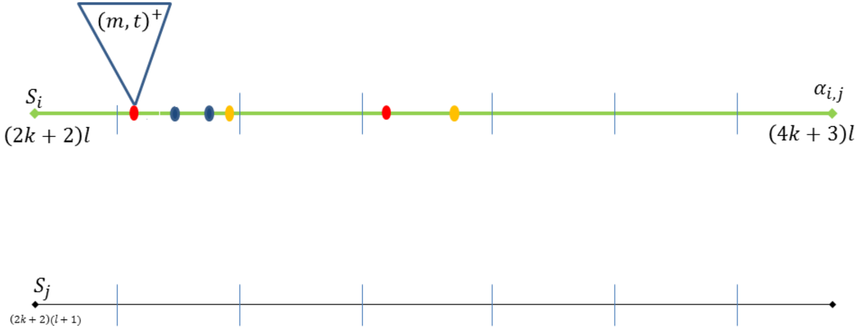


Figure: Encoding Configuration

Complexity of Satisfiability Checking of $MTL[\Diamond_I]$

- We reduce the reachability problem of ICMET to the above problem.

- We reduce the reachability problem of ICMET to the above problem.
- The instructions of the form are Write($c!m$) Read($c = \phi$) are easily expressible using $MTL[F_I]$ formula.

Complexity of Satisfiability Checking of $MTL[\Diamond_I]$

- We reduce the reachability problem of ICMET to the above problem.
- The instructions of the form are Write($c!m$) Read($c = \phi$) are easily expressible using $MTL[F_I]$ formula.
- We discuss Read($c?m$). The challenge is to assert that a particular point resembles head of the channel.

Complexity of Satisfiability Checking of $MTL[\Diamond_I]$

- We reduce the reachability problem of ICMET to the above problem.
- The instructions of the form are Write($c!m$) Read($c = \phi$) are easily expressible using $MTL[F_I]$ formula.
- We discuss Read($c?m$). The challenge is to assert that a particular point resembles head of the channel.
- For this we introduce a special symbol b which acts as a separator between head of the channel(first symbol in the unit integral interval) and the rest of the contents of channel.

Figure: Run showing $C_1?m$

Segment of Timed Word showing Read($C_1?m$)

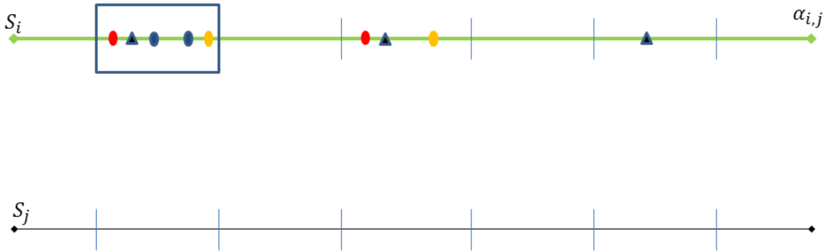


Figure: Run showing $C_1?m$

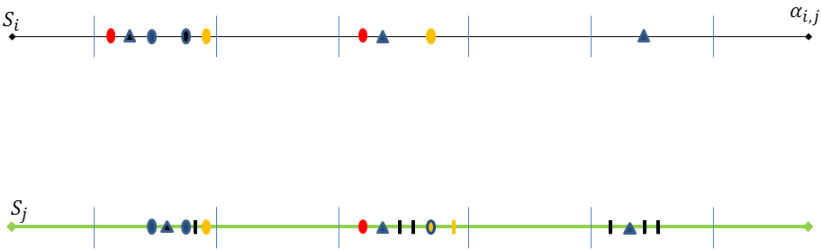


Figure: Run showing $C_1?m$

Contd.

- b as separator

Contd.

- b as separator
  - There is one and only one b in a particular channel:

$$\phi_{b_1} = \Box[S \Rightarrow (\bigwedge_{i=1}^{k} \Diamond_{(2i-1,2i)}(b))]$$

$$\phi_{b_2} = \Box(b \Rightarrow \neg\Diamond_{(0,1)} b)$$

Contd.

- b as separator
  - There is one and only one b in a particular channel:

    $$\phi_{b_1} = \Box[S \Rightarrow (\bigwedge_{i=1}^{k} \Diamond_{(2i-1,2i)}(b))]$$

    $$\phi_{b_2} = \Box(b \Rightarrow \neg\Diamond_{(0,1)} b)$$

  - There is one and only one m before b (if c non-empty):

    $$\phi_{b_3} = \Box[\neg\{M \wedge \Diamond_{(0,1)}(M \wedge \Diamond_{(0,1)}(b)\}]$$

    $$\phi_{b_4} = \Box[S \Rightarrow \{\bigwedge_{j=1}^{k}(\Diamond_{(2j-1,2j)}(M) \Rightarrow \Diamond_{(2j-1,2j)}(M \wedge \Diamond_{(0,1)} b))\}]$$

- If transition is of the form $c_i = \emptyset$

$$\phi_{c_i=\emptyset} = S \wedge \Box_{(2i-1,2i)}(\neg action) \wedge \Box_{(0,2k+2)}(\bigwedge_{m \in M}(m \Rightarrow \Diamond_{[2k+2,2k+2]}(m))$$

- If transition is of the form $c_i = \emptyset$

$$\phi_{c_i=\emptyset} = S \wedge \Box_{(2i-1,2i)}(\neg action) \wedge \Box_{(0,2k+2)}(\bigwedge_{m \in M}(m \Rightarrow \Diamond_{[2k+2,2k+2]}(m))$$

- If transition is of the form $c_i?m_x$ where $m_x \in M$

$$\phi_{c_i?m_x} = S \wedge \bigwedge_{j \neq i, j=1}^{k} \Box_{[2j-1,2j]}\{\bigwedge_{m \in M} m \Rightarrow \Diamond_{[2k+2,2k+2]}(m)\} \wedge \Diamond_{(2i-1,2i)}\{m_x \wedge \Diamond_{(0,1)}(b)\} \wedge \Box_{[2i-1,2i]}\{\bigwedge_{m \in M}(m \wedge \neg\Diamond_{(0,1)} b) \Rightarrow \Diamond_{[2k+2,2k+2]}(m)\}$$

Presentation Flow

- Preliminaries
- Satisfiability Checking
  - $MTL[\Diamond_I, \Diamondminus_I]$
  - $2 - TPTL[\Diamond_I]$
  - $MTL[\Diamond_I]$
- Expressiveness
- Conclusion
- Future Work

Expressiveness

Theorem: $MTL[\Diamond_I, \mathrm{O}]$ is incomparable to $MTL[\mathrm{U}_I]$ $\varphi \equiv a \,\mathrm{U}\, b$

$$A(0)A'(\delta)A'(2\delta)\ldots A'(1)$$

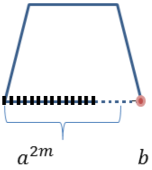


Figure: $\rho_0 \vdash \varphi$

$$A'(0)A'(\delta)A'(2\delta)\ldots A'(1)$$

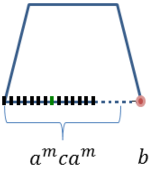


Figure: $\rho_1 \nvdash \varphi$

Presentation Flow

- Preliminaries
- Satisfiability Checking
    - $MTL[\Diamond_I, \diamondminus_I]$
    - $2 - TPTL[\Diamond_I]$
    - $MTL[\Diamond_I]$
- Expressiveness
- Conclusion
- Future Work

Conclusion

- Not much is gained in terms of satisfiability checking by restricting to unary operators and non strict operators.
- Unlike binary, strict and non strict unary operators collide with strict monotonic restriction.

Conclusion

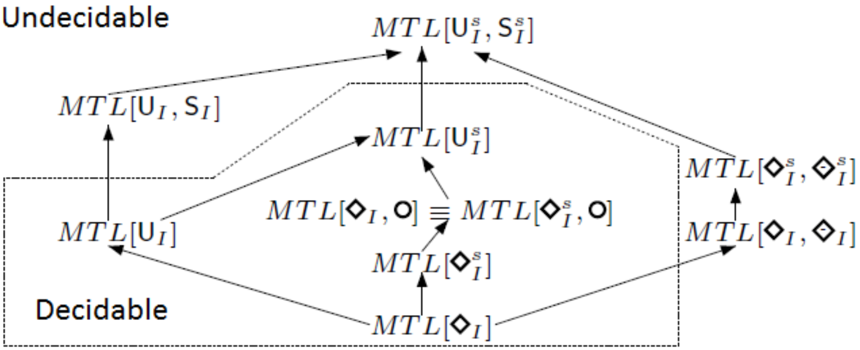


Figure: Expressiveness Hierarchy. Classes within Polygon have decidable satisfiability checking

Conclusion

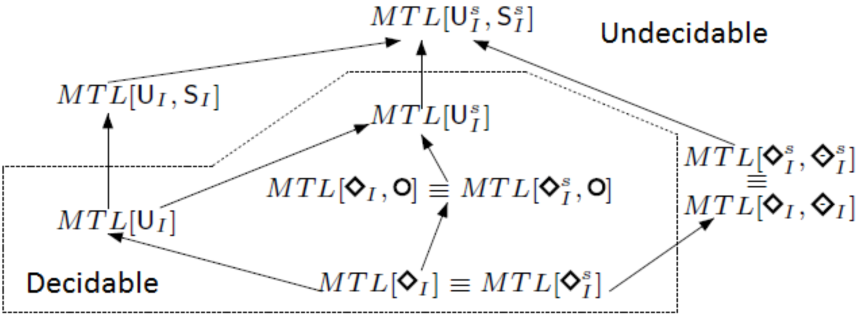


Figure: Expressiveness Hierarchy with Strict Monotonicity. Classes within Polygon have decidable satisfiability checking

Future Work

- Explore membership checking algorithms for these fragments.
- There is not much known about decidable fragments of *TPTL* with more than one clocks other than positive fragment.It would be interesting to explore those fragments too.
- To check how these unary fragments behave with different semantic restriction.
- And to come up with more restrictions which gives us low complexity satisfiability checking.
- Explore satisfiability checking on infinite words too.

Thank You