Temporal Answer Set Programming on Finite Traces

Pedro Cabalar, Roland Kaminski, Torsten Schaub, Anna Schuhmann

University of Corunna, Spain University of Potsdam, Germany

Outline

Outline

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this?
  Problems consisting of (many) decisions and constraints

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this?
  Problems consisting of (many) decisions and constraints

  Examples Sudoku, Configuration, Diagnosis, Music composition, Planning, System design, Time tabling, etc.

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this? — And **industrial ones**?
  Problems consisting of (many) decisions and constraints

  Examples Sudoku, Configuration, Diagnosis, Music composition, Planning, System design, Time tabling, etc.

Potassco

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this? — And **industrial ones** ?
  - Debian, Ubuntu: Linux package configuration
  - Exeura: Call routing
  - Fcc: Radio frequency auction
  - Gioia Tauro: Workforce management
  - Nasa: Decision support for Space Shuttle
  - Siemens: Partner units configuration
  - Variantum: Product configuration

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this? — And **industrial ones**?
  - Debian, Ubuntu: Linux package configuration
  - Exeura: Call routing
  - Fcc: Radio frequency auction
  - Gioia Tauro: Workforce management
  - Nasa: Decision support for Space Shuttle
  - Siemens: Partner units configuration
  - Variantum: Product configuration

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative pr
- What is ASP good for?
  Solving knowledge-intense combinato
- What problems are this? — And **indu**
  - Debian, Ubuntu: Linux package cor
  - Exeura: Call routing
  - **Fcc: Radio frequency auction**
  - Gioia Tauro: Workforce management
  - Nasa: Decision support for Space Shuttle
  - Siemens: Partner units configuration
  - Variantum: Product configuration

> Over 13 months in 2016–17 the US Federal Communications Commission conducted an "incentive auction" to repurpose radio spectrum from broadcast television to wireless internet. In the end, the auction yielded $19.8 billion, $10.05 billion of which was paid to 175 broadcasters for voluntarily relinquishing their licenses across 14 UHF channels. Stations that continued broadcasting were assigned potentially new channels to fit as densely as possible into the channels that remained. The government netted more than $7 billion (used to pay down the national debt) after covering costs. A crucial element of the auction design was the construction of a solver, dubbed SATFC, that determined whether sets of stations could be "repacked" in this way; it needed to run every time a station was given a price quote. This

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this?
  Problems consisting of (many) decisions and constraints
- What are ASP's distinguishing features?
  - High level, versatile modeling language
  - High performance solvers

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this?
  Problems consisting of (many) decisions and constraints
- What are ASP's distinguishing features?
  - High level, versatile modeling language
  - High performance solvers
- Any industrial impact?
  - ASP Tech companies: dlv systems and potassco solutions

Answer Set Programming (ASP)

- What is ASP?
  ASP is an approach for declarative problem solving
- What is ASP good for?
  Solving knowledge-intense combinatorial (optimization) problems
- What problems are this?
  Problems consisting of (many) decisions and constraints
- What are ASP's distinguishing features?
  - High level, versatile modeling language
  - High performance solvers
- Any industrial impact?
  - ASP Tech companies: dlv systems and potassco solutions
- Anything not so good for ASP?
  - Number crunching

Some biased moments in time

- '70/'80 Capturing incomplete information

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
    - Axiomatic characterization
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
    - Axiomatic characterization
  - Logic programming Negation as failure
    - Herbrand interpretations
    - Fix-point characterizations
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
    - Axiomatic characterization
  - Logic programming Negation as failure
    - Herbrand interpretations
    - Fix-point characterizations
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
    - Extensions of first-order logic
    - Modalities, fix-points, second-order logic

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
    - Stable models semantics derived from non-monotonic logics
    - Alternating fix-point theory
  - ASP solving
    "Stable models = Well-founded semantics + Branch"

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
    - Stable models semantics derived from non-monotonic logics
    - Alternating fix-point theory
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
    - Modeling — Grounding — Solving
    - Icebreakers: `lparse` and `smodels`

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
  - Growing dissemination — *see last slide* —
  - Constructive logics Equilibrium Logic

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
  - Growing dissemination — *see last slide* —
  - Constructive logics Equilibrium Logic
    - Roots: Logic of Here-and-There (Heyting'30), G3 (Gödel'32)

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
  - Growing dissemination — *see last slide* —
  - Constructive logics Equilibrium Logic
- '10 Integration

Some biased moments in time

- '70/'80 Capturing incomplete information
  - Databases Closed world assumption
  - Logic programming Negation as failure
  - Non-monotonic reasoning
    Auto-epistemic and Default logics, Circumscription
- '90 Amalgamation and computation
  - Logic programming semantics
    Well-founded and stable models semantics
  - ASP solving
    "Stable models = Well-founded semantics + Branch"
- '00 Applications and semantic rediscoveries
  - Growing dissemination — *see last slide* —
  - Constructive logics Equilibrium Logic
- '10 Integration — let's see ...

Robotic intra-logistic

- Robotics systems for logistics and warehouse automation based on hundreds of
  - mobile robots
  - movable shelves

Robotic intra-logistic

- Robotics systems for logistics and warehouse automation based on hundreds of
  - mobile robots
  - movable shelves
- Main tasks: order fulfillment, i.e.
  - routing
  - order picking
  - replenishment

Robotic intra-logistic

- Robotics systems for logistics and warehouse automation based on hundreds of
  - mobile robots
  - movable shelves
- Main tasks: order fulfillment, i.e.
  - routing
  - order picking
  - replenishment
- Many competing industry solutions:
  - Amazon, Dematic, Genzebach, Gray Orange, Swisslog

Robotic intra-logistic in ASP
routing

```
time(1..horizon).

direction((X,Y)) :- X=-1..1, Y=-1..1, |X+Y|=1.
nextto((X,Y),(X',Y'),(X+X',Y+Y')) :- position((X,Y)), direction((X',Y')), position((X+X',Y+Y')).

 { move(R,D,T) : direction(D) } 1 :- isRobot(R), time(T).

position(R,C,T) :- move(R,D,T), position(R,C',T-1),     nextto(C',D,C).
                :- move(R,D,T), position(R,C ,T-1), not nextto(C ,D,_).

position(R,C,T) :- position(R,C,T-1), not move(R,_,T), isRobot(R), time(T).

moveto(C',C,T) :- nextto(C',D,C), position(R,C',T-1), move(R,D,T).
 :- moveto(C',C,T), moveto(C,C',T).

 :- { position(R,C,T) : isRobot(R) }  > 1, position(C), time(T).
```

Robotic intra-logistic in ASP
routing to shelves

```
time(1..horizon).

direction((X,Y)) :- X=-1..1, Y=-1..1, |X+Y|=1.
nextto((X,Y),(X',Y'),(X+X',Y+Y')) :- position((X,Y)), direction((X',Y')), position((X+X',Y+Y')).

 { move(R,D,T) : direction(D) } 1 :- isRobot(R), time(T).

position(R,C,T) :- move(R,D,T), position(R,C',T-1),     nextto(C',D,C).
                :- move(R,D,T), position(R,C ,T-1), not nextto(C ,D,_).

position(R,C,T) :- position(R,C,T-1), not move(R,_,T), isRobot(R), time(T).

moveto(C',C,T) :- nextto(C',D,C), position(R,C',T-1), move(R,D,T).
 :- moveto(C',C,T), moveto(C,C',T).

 :- { position(R,C,T) : isRobot(R) }  > 1, position(C), time(T).


processed(O,A) :- ordered(O,A), shelved(S,A), position(S,C,0), position(R,C,horizon), isRobot(R).

processed(O) :- isOrder(O), processed(O,A) : ordered(O,A).

:- not processed(O), isOrder(O).
```

Potassco

Robotic intra-logistic

Robotic intra-logistic in ASP

routing + transport + delivery

```
time(1..horizon).

direction((X,Y)) :- X=-1..1, Y=-1..1, |X+Y|=1.
nextto((X,Y),(X',Y'),(X+X',Y+Y')) :- position((X,Y)), direction((X',Y')), position((X+X',Y+Y')).

 {    move(R,D,T) : direction(D) ;
    pickup(R,S,T) : isShelf(S)   ;
   putdown(R,S,T) : isShelf(S)   } 1 :- isRobot(R), time(T).

waits(R,T) :- not pickup(R,_,T), not putdown(R,_,T), not move(R,_,T), isRobot(R), time(T).

position(R,C,T) :- move(R,D,T),    position(R,C',T-1),     nextto(C',D,C).
                :- move(R,D,T),    position(R,C ,T-1), not nextto(C, D,_).

 carries(R,S,T) :- pickup(R,S,T),  position(R,C,T-1), position(S,C ,T-1).
                :- pickup(R,S,T),   carries(R,_,T-1).
                :- pickup(R,S,T),   carries(_,S,T-1).
                :- pickup(R,S,T),  position(R,C,T-1), position(S,C',T-1), C != C'.

                :- putdown(R,S,T), not carries(R,S,T-1).

serves(R,S,P,T) :- position(R,C,T), carries(R,S,T),  position(P,C), isStation(P).

position(R,C,T) :- position(R,C,T-1), not    move(R,_,T), isRobot(R), time(T).
 carries(R,S,T) :-  carries(R,S,T-1), not putdown(R,_,T),             time(T).

position(S,C,T) :- position(R,C,T  ),     carries(R,S,T).
position(S,C,T) :- position(S,C,T-1), not carries(_,S,T), isShelf(S), time(T).

moveto(C',C,T) :- nextto(C',D,C), position(R,C',T-1), move(R,D,T).
:- moveto(C',C,T), moveto(C,C',T), C < C'.

 :- { position(R,C,T) : isRobot(R) }  > 1, position(C), time(T).
 :- { position(S,C,T) : isShelf(S) }  > 1, position(C), time(T).
```

Robotic intra-logistic

Outline

Motivation

- Formal accounts of dynamic systems
  - temporal logics
  - calculi for action and change

Motivation

- Formal accounts of dynamic systems
  - temporal logics
  - calculi for action and change
- Answer Set Programming (ASP)
  - Temporal equilibrium logic
    - language of *LTL*
    - complexity beyond *LTL*
    - infinite traces
  - Action languages
    - static and dynamic laws
    - same complexity as ASP
    - finite traces

Motivation

- Formal accounts of dynamic systems
  - temporal logics
  - calculi for action and change
- Answer Set Programming (ASP)
  - Temporal equilibrium logic
    - language of *LTL*
    - complexity beyond *LTL*
    - infinite traces
  - Action languages
    - static and dynamic laws
    - same complexity as ASP
    - finite traces
- Useful for representing and reasoning dynamic knowledge?

Motivation

- Formal accounts of dynamic systems
  - temporal logics
  - calculi for action and change
- Answer Set Programming (ASP)
  - Temporal equilibrium logic
    - language of *LTL*
    - complexity beyond *LTL*
    - infinite traces
  - Action languages
    - static and dynamic laws
    - same complexity as ASP
    - finite traces
- Useful for representing and reasoning dynamic knowledge?

Motivation

- Formal accounts of dynamic systems
  - temporal logics
  - calculi for action and change
- Answer Set Programming (ASP)
  - Temporal equilibrium logic
    - language of *LTL*
    - complexity beyond *LTL*
    - infinite traces
  - Action languages
    - static and dynamic laws
    - same complexity as ASP
    - finite traces
- Useful for representing and reasoning dynamic knowledge?

Motivation

- Formal accounts of dynamic systems
  - temporal logics
  - calculi for action and change
- Answer Set Programming (ASP)
  - Temporal equilibrium logic
    - language of *LTL*
    - complexity beyond *LTL*
    - infinite traces
  - Action languages
    - static and dynamic laws
    - same complexity as ASP
    - finite traces
- Proposal Temporal equilibrium logic over finite traces

Motivation

- Formal accounts of dynamic systems
  - temporal logics
  - calculi for action and change
- Answer Set Programming (ASP)
  - Temporal equilibrium logic
    - language of LTL
    - complexity beyond LTL
    - infinite traces
  - Action languages
    - static and dynamic laws
    - same complexity as ASP
    - finite traces
- Proposal Temporal equilibrium logic over finite traces

  $\sim$ LTL_f by G. De Giacomo and M. Vardi (2013)

Potassco

Outline

Potassco

Regular formulas

- Formulas

$$\varphi ::= a \mid \bot \mid \varphi_1 \otimes \varphi_2$$

  where
  - a is an atom
  - $\otimes$ is a binary Boolean connective among $\rightarrow, \wedge, \vee$

- Defined connectives
  - $\top = \neg\bot$
  - $\neg\varphi = \varphi \rightarrow \bot$
  - $\varphi \leftrightarrow \psi = (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$

Regular formulas*

- Formulas

$$\varphi ::= a \mid \bot \mid \varphi_1 \otimes \varphi_2$$

where

  - a is an atom
  - $\otimes$ is a binary Boolean connective among $\rightarrow, \wedge, \vee$

- Defined connectives
  - $\top = \neg\bot$
  - $\neg\varphi = \varphi \rightarrow \bot$
  - $\varphi \leftrightarrow \psi = (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$

* in the logic of Here-and-There (Heyting'32; Gödel'32)

Temporal formulas

- Temporal operators

past	$\bullet$	for *previous*	*future*	$\circ$	for *next*
	$\mathbf{S}$	for *since*		$\mathbb{U}$	for *until*
	$\mathbf{T}$	for *trigger*		$\mathbb{R}$	for *release*

Temporal formulas

- Temporal operators

past	$\bullet$	for *previous*	*future*	$\circ$	for *next*
	$\mathbf{S}$	for *since*		$\mathbb{U}$	for *until*
	$\mathbf{T}$	for *trigger*		$\mathbb{R}$	for *release*

- Temporal formulas

$$\varphi ::= a \mid \bot \mid \varphi_1 \otimes \varphi_2 \mid \bullet\varphi \mid \varphi_1 \,\mathbf{S}\, \varphi_2 \mid \varphi_1 \,\mathbf{T}\, \varphi_2 \mid \circ\varphi \mid \varphi_1 \,\mathbb{U}\, \varphi_2 \mid \varphi_1 \,\mathbb{R}\, \varphi_2$$

- Defined operators

$\blacksquare\varphi = \bot \,\mathbf{T}\, \varphi$	*always before*	$\Box\varphi = \bot \,\mathbb{R}\, \varphi$	*always afterward*
$\blacklozenge\varphi = \top \,\mathbf{S}\, \varphi$	*eventually before*	$\Diamond\varphi = \top \,\mathbb{U}\, \varphi$	*eventually afterward*
$\mathbf{I} = \neg\bullet\top$	*initial*	$\mathbb{F} = \neg\circ\top$	*final*
$\widehat{\bullet}\varphi = \bullet\varphi \vee \mathbf{I}$	*weak previous*	$\widehat{\circ}\varphi = \circ\varphi \vee \mathbb{F}$	*weak next*

Examples

- *"If we shoot twice with a gun that was never loaded, it will eventually fail."*

$$\Box(\mathit{shoot} \wedge \bullet\blacklozenge\mathit{shoot} \wedge \blacksquare\mathit{unloaded} \rightarrow \Diamond\mathit{fail})$$

- *"Why does shooting a loaded gun fail in unloading it?"*

$$\Box(\mathbb{F} \rightarrow \neg\neg(\mathit{shoot} \wedge \bullet\mathit{loaded} \wedge \mathit{loaded}))$$

Outline

From models to traces

- Alphabet Set $\mathcal{A}$ of atoms

From models to traces

- Alphabet Set $\mathcal{A}$ of atoms
- Model A set $H \subseteq \mathcal{A}$ of atoms

From models to traces

- Alphabet Set $\mathcal{A}$ of atoms
- Model A set $H \subseteq \mathcal{A}$ of atoms
- *HT*-Model A pair $\langle H, T \rangle$ of set of atoms st $H \subseteq T \subseteq \mathcal{A}$

From models to traces

- Alphabet Set $\mathcal{A}$ of atoms
- Trace A sequence $\langle H_i \rangle_{i=0}^{\lambda}$ of sets $H_i \subseteq \mathcal{A}$

From models to traces

- Alphabet Set $\mathcal{A}$ of atoms
- Trace A sequence $\langle H_i \rangle_{i=0}^{\lambda}$ of sets $H_i \subseteq \mathcal{A}$
  - finite if $\lambda < \omega$
  - infinite if $\lambda = \omega$

From models to traces

- Alphabet Set $\mathcal{A}$ of atoms
- Trace A sequence $\langle H_i \rangle_{i=0}^{\lambda}$ of sets $H_i \subseteq \mathcal{A}$
  - finite if $\lambda < \omega$
  - infinite if $\lambda = \omega$
- Notation
  - We often abbreviate $\langle H_i \rangle_{i=0}^{\lambda}$ by $\mathbf{H}$
  - $\mathbf{H} \leq \mathbf{H}'$ if $H_i \subseteq H'_i$ for $i = 0..\lambda$

From models to traces

- Alphabet Set $\mathcal{A}$ of atoms
- Trace A sequence $\langle H_i \rangle_{i=0}^{\lambda}$ of sets $H_i \subseteq \mathcal{A}$
  - finite if $\lambda < \omega$
  - infinite if $\lambda = \omega$
- Notation
  - We often abbreviate $\langle H_i \rangle_{i=0}^{\lambda}$ by $\mathbf{H}$
  - $\mathbf{H} \leq \mathbf{H}'$ if $H_i \subseteq H'_i$ for $i = 0..\lambda$

- *HT*-Trace A sequence $\langle H_i, T_i \rangle_{i=0}^{\lambda}$ of pairs st $H_i \subseteq T_i \subseteq \mathcal{A}$ for $i = 0..\lambda$

From models to traces

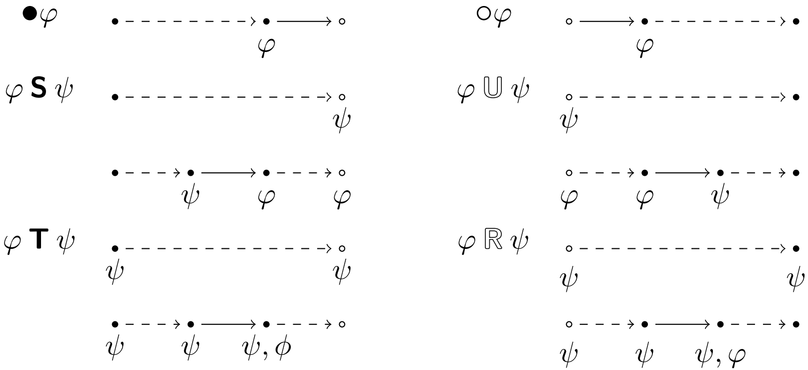
- Alphabet Set $\mathcal{A}$ of atoms
- Trace A sequence $\langle H_i \rangle_{i=0}^{\lambda}$ of sets $H_i \subseteq \mathcal{A}$
  - finite if $\lambda < \omega$
  - infinite if $\lambda = \omega$
- Notation
  - We often abbreviate $\langle H_i \rangle_{i=0}^{\lambda}$ by $\mathbf{H}$
  - $\mathbf{H} \leq \mathbf{H}'$ if $H_i \subseteq H_i'$ for $i = 0..\lambda$
- *HT*-Trace A sequence $\langle H_i, T_i \rangle_{i=0}^{\lambda}$ of pairs st $H_i \subseteq T_i \subseteq \mathcal{A}$ for $i = 0..\lambda$
- Notation We abbreviate $\langle H_i, T_i \rangle_{i=0}^{\lambda}$ by $\langle \mathbf{H}, \mathbf{T} \rangle$
- Note $\mathbf{H} \leq \mathbf{T}$

Satisfaction of regular formulas

An *HT*-trace $\langle \mathbf{H}, \mathbf{T} \rangle$ of length λ over alphabet $\mathcal{A}$ *satisfies* a temporal formula φ at time point $k = 0..\lambda$, $k \neq \omega$, written $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \varphi$, if the following conditions hold:

1. $\langle \mathbf{H}, \mathbf{T} \rangle, k \not\models \bot$
2. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models a$ iff $a \in H_k$, for any atom $a \in \mathcal{A}$
3. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \varphi \wedge \psi$ iff $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \varphi$ and $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \psi$
4. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \varphi \vee \psi$ iff $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \varphi$ or $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \psi$
5. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \varphi \rightarrow \psi$ iff $\langle \mathbf{H}', \mathbf{T} \rangle, k \not\models \varphi$ or $\langle \mathbf{H}', \mathbf{T} \rangle, k \models \psi$,
   for all $\mathbf{H}' \in \{\mathbf{H}, \mathbf{T}\}$

Satisfaction of temporal formulas



Satisfaction of temporal formulas

6. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \bullet\varphi$ iff $k > 0$ and $\langle \mathbf{H}, \mathbf{T} \rangle, k{-}1 \models \varphi$
7. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \varphi \, \mathbf{S} \, \psi$ iff for some $j = 0..k$, we have $\langle \mathbf{H}, \mathbf{T} \rangle, j \models \psi$ and $\langle \mathbf{H}, \mathbf{T} \rangle, i \models \varphi$ for all $i = j{+}1..k$
8. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \varphi \, \mathbf{T} \, \psi$ iff for all $j = 0..k$, we have $\langle \mathbf{H}, \mathbf{T} \rangle, j \models \psi$ or $\langle \mathbf{H}, \mathbf{T} \rangle, i \models \varphi$ for some $i = j{+}1..k$

Satisfaction of temporal formulas

6 $\langle \mathbf{H}, \mathbf{T}\rangle, k \models \bullet\varphi$ iff $k > 0$ and $\langle \mathbf{H}, \mathbf{T}\rangle, k{-}1 \models \varphi$

7 $\langle \mathbf{H}, \mathbf{T}\rangle, k \models \varphi \,\mathbf{S}\, \psi$ iff for some $j = 0..k$, we have $\langle \mathbf{H}, \mathbf{T}\rangle, j \models \psi$ and $\langle \mathbf{H}, \mathbf{T}\rangle, i \models \varphi$ for all $i = j{+}1..k$

8 $\langle \mathbf{H}, \mathbf{T}\rangle, k \models \varphi \,\mathbf{T}\, \psi$ iff for all $j = 0..k$, we have $\langle \mathbf{H}, \mathbf{T}\rangle, j \models \psi$ or $\langle \mathbf{H}, \mathbf{T}\rangle, i \models \varphi$ for some $i = j{+}1..k$

9 $\langle \mathbf{H}, \mathbf{T}\rangle, k \models \circ\varphi$ iff $k < \lambda$ and $\langle \mathbf{H}, \mathbf{T}\rangle, k{+}1 \models \varphi$

10 $\langle \mathbf{H}, \mathbf{T}\rangle, k \models \varphi \,\mathbb{U}\, \psi$ iff for some $j = k..\lambda$, we have $\langle \mathbf{H}, \mathbf{T}\rangle, j \models \psi$ and $\langle \mathbf{H}, \mathbf{T}\rangle, i \models \varphi$ for all $i = k..j{-}1$

11 $\langle \mathbf{H}, \mathbf{T}\rangle, k \models \varphi \,\mathbb{R}\, \psi$ iff for all $j = k..\lambda$, we have $\langle \mathbf{H}, \mathbf{T}\rangle, j \models \psi$ or $\langle \mathbf{H}, \mathbf{T}\rangle, i \models \varphi$ for some $i = k..j{-}1$. $\square$

Satisfaction of temporal formulas

6. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \bullet\varphi$ iff $k > 0$ and $\langle \mathbf{H}, \mathbf{T} \rangle, k{-}1 \models \varphi$
7. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \varphi \, \mathbf{S} \, \psi$ iff for some $j = 0..k$, we have $\langle \mathbf{H}, \mathbf{T} \rangle, j \models \psi$ and $\langle \mathbf{H}, \mathbf{T} \rangle, i \models \varphi$ for all $i = j{+}1..k$
8. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \varphi \, \mathbf{T} \, \psi$ iff for all $j = 0..k$, we have $\langle \mathbf{H}, \mathbf{T} \rangle, j \models \psi$ or $\langle \mathbf{H}, \mathbf{T} \rangle, i \models \varphi$ for some $i = j{+}1..k$

9. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \circ\varphi$ iff $k < \lambda$ and $\langle \mathbf{H}, \mathbf{T} \rangle, k{+}1 \models \varphi$
10. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \varphi \, \mathbb{U} \, \psi$ iff for some $j = k..\lambda$, we have $\langle \mathbf{H}, \mathbf{T} \rangle, j \models \psi$ and $\langle \mathbf{H}, \mathbf{T} \rangle, i \models \varphi$ for all $i = k..j{-}1$
11. $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \varphi \, \mathbb{R} \, \psi$ iff for all $j = k..\lambda$, we have $\langle \mathbf{H}, \mathbf{T} \rangle, j \models \psi$ or $\langle \mathbf{H}, \mathbf{T} \rangle, i \models \varphi$ for some $i = k..j{-}1$. $\square$

Satisfaction of (defined) temporal formulas

12 $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \top$

13 $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \blacksquare\varphi$ iff $\langle \mathbf{H}, \mathbf{T} \rangle, i \models \varphi$ for all $i = 0..k$

14 $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \blacklozenge\varphi$ iff $\langle \mathbf{H}, \mathbf{T} \rangle, i \models \varphi$ for some $i = 0..k$

15 $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \mathbf{I}$ iff $k = 0$

16 $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \widehat{\bullet}\varphi$ iff $k = 0$ or $\langle \mathbf{H}, \mathbf{T} \rangle, k{-}1 \models \varphi$

17 $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \Box\varphi$ iff $\langle \mathbf{H}, \mathbf{T} \rangle, i \models \varphi$ for any $i = k..\lambda$

18 $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \Diamond\varphi$ iff $\langle \mathbf{H}, \mathbf{T} \rangle, i \models \varphi$ for some $i = k..\lambda$

19 $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \mathbb{F}$ iff $k = \lambda$

20 $\langle \mathbf{H}, \mathbf{T} \rangle, k \models \widehat{\circ}\varphi$ iff $k = \lambda$ or $\langle \mathbf{H}, \mathbf{T} \rangle, k{+}1 \models \varphi$

Emerging temporal logics

- Temporal logic of here-and-there (THT)

Emerging temporal logics

- Temporal logic of here-and-there (*THT*)
- Finale
  - $\Diamond\mathbb{F}$ enforces finite traces
  - $\neg\Diamond\mathbb{F}$ enforces infinite traces

Emerging temporal logics

- Temporal logic of here-and-there (THT)
- Finale
  - $\Diamond\mathbb{F}$ enforces finite traces
  - $\neg\Diamond\mathbb{F}$ enforces infinite traces
- Excluded middle (EM)
  - $\Box(a \vee \neg a)$ for each atom $a \in \mathcal{A}$

Emerging temporal logics

- Temporal logic of here-and-there (THT)

- Finale
  - $\Diamond\mathbb{F}$ enforces finite traces
  - $\neg\Diamond\mathbb{F}$ enforces infinite traces
- Excluded middle (EM)
  - $\square(a \vee \neg a)$ for each atom $a \in \mathcal{A}$

- Temporal logics stronger than THT
  - $THT_\omega = THT + \{\neg\Diamond\mathbb{F}\}$
  - $THT_f = THT + \{\Diamond\mathbb{F}\}$
  - $LTL = THT + \{(EM)\}$
  - $LTL_\omega = THT_\omega + \{(EM)\}$
  - $LTL_f = THT_f + \{(EM)\}$

Emerging temporal logics

- Temporal logic of here-and-there (THT)
- Finale
  - $\Diamond\mathbb{F}$ enforces finite traces
  - $\neg\Diamond\mathbb{F}$ enforces infinite traces
- Excluded middle (EM)
  - $\Box(a \vee \neg a)$ for each atom $a \in \mathcal{A}$
- Temporal logics stronger than THT
  - $THT_\omega = THT + \{\neg\Diamond\mathbb{F}\}$
  - $THT_f = THT + \{\Diamond\mathbb{F}\}$
  - $LTL = THT + \{(EM)\}$
  - $LTL_\omega = THT_\omega + \{(EM)\}$
  - $LTL_f = THT_f + \{(EM)\}$

Emerging temporal logics

- Temporal logic of here-and-there (THT)

- Finale
  - $\Diamond\mathbb{F}$ enforces finite traces
  - $\neg\Diamond\mathbb{F}$ enforces infinite traces
- Excluded middle (EM)
  - $\square(a \vee \neg a)$ for each atom $a \in \mathcal{A}$

- Temporal logics stronger than THT
  - $THT_\omega = THT + \{\neg\Diamond\mathbb{F}\}$
  - $THT_f = THT + \{\Diamond\mathbb{F}\}$
  - $LTL = THT + \{(EM)\}$
  - $LTL_\omega = THT_\omega + \{(EM)\}$
  - $LTL_f = THT_f + \{(EM)\}$

- Note All variants of THT are monotonic !

Temporal equilibrium logic (*TEL*)

- A total *HT*-trace $\langle \mathbf{T}, \mathbf{T} \rangle$ is an equilibrium model of a temporal formula φ, if

  1. $\langle \mathbf{T}, \mathbf{T} \rangle, 0 \models \varphi$,
  2. $\langle \mathbf{H}, \mathbf{T} \rangle, 0 \not\models \varphi$ for all $\mathbf{H} < \mathbf{T}$

Temporal equilibrium logic (*TEL*)

- A total *HT*-trace $\langle \mathbf{T}, \mathbf{T} \rangle$ is an equilibrium model of a temporal formula φ, if

  1. $\langle \mathbf{T}, \mathbf{T} \rangle, 0 \models \varphi$,
  2. $\langle \mathbf{H}, \mathbf{T} \rangle, 0 \not\models \varphi$ for all $\mathbf{H} < \mathbf{T}$

- $\mathbf{T}$ is called a temporal stable model of φ

Temporal equilibrium logic (*TEL*)

- A total *HT*-trace $\langle \mathbf{T}, \mathbf{T} \rangle$ is an equilibrium model of a temporal formula φ, if
  1. $\langle \mathbf{T}, \mathbf{T} \rangle, 0 \models \varphi$,
  2. $\langle \mathbf{H}, \mathbf{T} \rangle, 0 \not\models \varphi$ for all $\mathbf{H} < \mathbf{T}$
- $\mathbf{T}$ is called a temporal stable model of φ
- Examples
  - $\square(\neg a \rightarrow \circ a)$ yields
    - $(\emptyset\ \{a\})^\omega$ in TEL_ω and $(\emptyset\ \{a\})^+$ in TEL_f

Temporal equilibrium logic (*TEL*)

- A total *HT*-trace $\langle \mathbf{T}, \mathbf{T} \rangle$ is an equilibrium model of a temporal formula φ, if
  1. $\langle \mathbf{T}, \mathbf{T} \rangle, 0 \models \varphi$,
  2. $\langle \mathbf{H}, \mathbf{T} \rangle, 0 \not\models \varphi$ for all $\mathbf{H} < \mathbf{T}$
- $\mathbf{T}$ is called a temporal stable model of φ
- Examples
  - $\Box(\neg a \rightarrow \circ a)$ yields
    - $(\emptyset\ \{a\})^\omega$ in TEL_ω and $(\emptyset\ \{a\})^+$ in TEL_f
  - $\Box(\neg \circ a \rightarrow a) \wedge \Box(\circ a \rightarrow a)$ yields
    - no model in TEL_ω but $(\{a\})^+$ in TEL_f

Temporal equilibrium logic (TEL)

- A total HT-trace $\langle \mathbf{T}, \mathbf{T} \rangle$ is an equilibrium model of a temporal formula φ, if
  1. $\langle \mathbf{T}, \mathbf{T} \rangle, 0 \models \varphi$,
  2. $\langle \mathbf{H}, \mathbf{T} \rangle, 0 \not\models \varphi$ for all $\mathbf{H} < \mathbf{T}$
- $\mathbf{T}$ is called a temporal stable model of φ
- Examples
  - $\Box(\neg a \rightarrow \circ a)$ yields
    - $(\emptyset\ \{a\})^\omega$ in TEL_ω and $(\emptyset\ \{a\})^+$ in TEL_f
  - $\Box(\neg \circ a \rightarrow a) \wedge \Box(\circ a \rightarrow a)$ yields
    - no model in TEL_ω but $(\{a\})^+$ in TEL_f
  - $\Box \Diamond a$ yields
    - no model in TEL_ω but $(\emptyset^* \{a\})$ in TEL_f

Outline

Normalform

- Temporal literals $\{a, \neg a, \bullet a, \neg\bullet a \mid a \in \mathcal{A}\}$

- Temporal rules
  - initial rule $B \to A$
  - dynamic rule $\widehat{\circ}\,\square(B \to A)$
  - final rule $\square(\,\mathbb{F} \to (B \to A)\,)$

  where $B = b_1 \wedge \cdots \wedge b_n$ and $A = a_1 \vee \cdots \vee a_m$
  and b_i and a_j are temporal literals for dynamic rules,
  and regular literals for initial and final rules

- Temporal logic program is a set of temporal rules

Normalform

- Temporal literals $\{a, \neg a, \bullet a, \neg\bullet a \mid a \in \mathcal{A}\}$
- Temporal rules
  - initial rule $B \to A$
  - dynamic rule $\widehat{\circ}\,\square(B \to A)$
  - final rule $\square(\,\mathbb{F} \to (B \to A)\,)$

  where $B = b_1 \wedge \cdots \wedge b_n$ and $A = a_1 \vee \cdots \vee a_m$
  and b_i and a_j are temporal literals for dynamic rules,
  and regular literals for initial and final rules
- Temporal logic program is a set of temporal rules
- Theorem Every temporal formula φ can be converted into a temporal logic program THT_f-equivalent to φ

Example

$$\widehat{\circ}\,\square(\bullet loaded \wedge \neg unloaded \rightarrow loaded)$$

$$\widehat{\circ}\,\square(shoot \wedge \bullet loaded \wedge loaded \rightarrow goal)$$

$$\square(\mathbb{F} \rightarrow (\neg goal \rightarrow \bot))$$

Bounded translation

- Temporal literals at time point k

$$\tau_k(a) = a_k \qquad \tau_k(\neg a) = \neg a_k$$
$$\tau_k(\bullet a) = a_{k-1} \qquad \tau_k(\neg \bullet a) = \neg a_{k-1}$$

- Temporal rules focusing on $B \to A$ at time point k

$$\tau_k(r) = \tau_k(a_1) \vee \cdots \vee \tau_k(a_m) \leftarrow \tau_k(b_1) \wedge \cdots \wedge \tau_k(b_n)$$

- Temporal logic program P bounded by finite length λ

$$\begin{aligned}\tau_\lambda(P) = \quad & \{\tau_0(r) \mid r \in I(P)\,\} \\ & \cup \{\tau_k(r) \mid r \in D(P), k = 1..\lambda\,\} \\ & \cup \{\tau_\lambda(r) \mid r \in F(P)\,\}\end{aligned}$$

Incremental translation

- Issue build $\tau_\lambda(P)$ from $\tau_{\lambda-1}(P)$
- Method module theory accounting for composition of logic programs

Incremental translation

- **Issue** build $\tau_\lambda(P)$ from $\tau_{\lambda-1}(P)$
- **Method** module theory accounting for composition of logic programs

- **Translation** as before, except for
  - translate final rules at time point k as

$$\tau^*_k(r) = \tau_k(a_1) \vee \cdots \vee \tau_k(a_m) \leftarrow \tau_k(b_1) \wedge \cdots \wedge \tau_k(b_n) \wedge \neg q_{k+1}$$

  - add q_k to each logic program at time point k

Incremental translation

- Issue build $\tau_\lambda(P)$ from $\tau_{\lambda-1}(P)$
- Method module theory accounting for composition of logic programs
- Translation as before, except for
  - translate final rules at time point k as

$$\tau_k^*(r) = \tau_k(a_1) \vee \cdots \vee \tau_k(a_m) \leftarrow \tau_k(b_1) \wedge \cdots \wedge \tau_k(b_n) \wedge \neg q_{k+1}$$

  - add q_k to each logic program at time point k

Outline

tel

- *tel*
  - is a preprocessor
  - implements the bounded translation
- *tel* is solver independent

tel

- *tel*
  - is a preprocessor
  - implements the bounded translation
- *tel* is solver independent

- Example

$$\{ \rightarrow a, \quad \widehat{\circ}\,\square(\bullet a \rightarrow b), \quad \square(\mathbb{F} \rightarrow (\neg b \rightarrow \bot)) \}$$

is represented as

```
a.
#next^ #always+ ( (#previous a) -> b).
#always+ ( #final -> (~ b -> #false)).
```

telingo

- *telingo*
  - extends the full modeling language of *clingo* by temporal operators
  - implements the incremental translation
- *telingo* is an extension of *clingo*

telingo

- *telingo*
  - extends the full modeling language of *clingo* by temporal operators
  - implements the incremental translation
- *telingo* is an extension of *clingo*

- Primes allow for expressing (iterated) next and previous operators
  - $\bullet p(a)$ and $\circ q(b)$ can be expressed by `'p(a)` and `q'(b)`

telingo

- *telingo*
  - extends the full modeling language of *clingo* by temporal operators
  - implements the incremental translation
- *telingo* is an extension of *clingo*

- Primes allow for expressing (iterated) next and previous operators
  - $\bullet p(a)$ and $\circ q(b)$ can be expressed by `'p(a)` and `q'(b)`
- Example *"A robot cannot lift a box unless its capacity exceeds the box's weight plus that of all held objects"*

```
:- lift(R,B), robot(R), box(B,W),
   #sum {    C : capacity(R,C);
          -V,O : 'holding(R,O,V) } < W.
```

telingo's temporal logic programs

- initial rule $B \rightarrow A$
- dynamic rule $\widehat{\circ}\,\Box(B \rightarrow A)$
- final rule $\Box(\,\mathbb{F} \rightarrow (B \rightarrow A)\,)$

telingo's temporal logic programs

- initial rule $B \to A$
- dynamic rule $\widehat{\circ}\,\square(B \to A)$
- final rule $\square(\,\mathbb{F} \to (B \to A)\,)$
- always rule $\square(B \to A)$

telingo's temporal logic programs

- initial rule $B \rightarrow A$ `#program initial.`
- dynamic rule $\widehat{\circ}\,\square(B \rightarrow A)$ `#program dynamic.`
- final rule $\square(\,\mathbb{F} \rightarrow (B \rightarrow A)\,)$ `#program final.`
- always rule $\square(B \rightarrow A)$ `#program always.`

telingo's temporal logic programs

- initial rule $B \rightarrow A$ `#program initial.`
- dynamic rule $\widehat{\circ}\,\square(B \rightarrow A)$ `#program dynamic.`
- final rule $\square(\,\mathbb{F} \rightarrow (B \rightarrow A)\,)$ `#program final.`
- always rule $\square(B \rightarrow A)$ `#program always.`
- Example $\{\; \rightarrow a,\; \widehat{\circ}\,\square(\bullet a \rightarrow b),\; \square(\mathbb{F} \rightarrow (\neg b \rightarrow \bot))\;\}$

telingo's temporal logic programs

- initial rule $B \to A$ `#program initial.`
- dynamic rule $\widehat{\circ}\,\square(B \to A)$ `#program dynamic.`
- final rule $\square(\,\mathbb{F} \to (B \to A)\,)$ `#program final.`
- always rule $\square(B \to A)$ `#program always.`

- Example $\{ \to a,\ \widehat{\circ}\,\square(\bullet a \to b),\ \square(\mathbb{F} \to (\neg b \to \bot)) \}$
  can alternatively be represented as

```
#program initial.
a.

#program dynamic.
b :- 'a.

#program final.
  :- not b.
```

```
#program always.
a :- &initial.


b :- 'a.


  :- not b, &final.
```

telingo's temporal formulas

- `&initial` $\mathbf{I}$
- `&final` $\mathbb{F}$

telingo's temporal formulas

- `&initial` $\mathbf{I}$
- `&final` $\mathbb{F}$
- `&tel {` φ `}` for temporal formula φ

telingo's temporal formulas

- `&initial` $\mathbf{I}$
- `&final` $\mathbb{F}$
- `&tel { `φ` }` for temporal formula φ
  - Temporal operators

past	$\bullet$	*previous*	*future*	$\circ$	*next*
	$\mathbf{S}$	*since*		$\mathbb{U}$	*until*
	$\mathbf{T}$	*trigger*		$\mathbb{R}$	*release*
	$\blacklozenge$	*eventually before*		$\Diamond$	*eventually afterward*
	$\blacksquare$	*always before*		$\Box$	*always afterward*
	$\widehat{\bullet}$	*weak previous*		$\widehat{\circ}$	*weak next*

telingo's temporal formulas

- `&initial` $\mathbf{I}$
- `&final` $\mathbb{F}$
- `&tel { φ }` for temporal formula φ
  - Temporal operators

past	`<`	*previous*	*future*	`>`	*next*
	`<?`	*since*		`>?`	*until*
	`<*`	*trigger*		`>*`	*release*
	`<?`	*eventually before*		`>?`	*eventually afterward*
	`<*`	*always before*		`>*`	*always afterward*
	`<:`	*weak previous*		`>:`	*weak next*

telingo's temporal formulas

- `&initial` $\mathbf{I}$
- `&final` $\mathbb{F}$

- `&tel { `φ` }` for temporal formula φ
  - Temporal operators

past	`<`	*previous*	*future*	`>`	*next*
	`<?`	*since*		`>?`	*until*
	`<*`	*trigger*		`>*`	*release*
	`<?`	*eventually before*		`>?`	*eventually afterward*
	`<*`	*always before*		`>*`	*always afterward*
	`<:`	*weak previous*		`>:`	*weak next*

  - Boolean operators `&` `|` `~`

telingo's temporal formulas

- `&initial` $\mathbf{I}$
- `&final` $\mathbb{F}$
- `&tel { φ }` for temporal formula φ
- Example

$$\mathit{shoot} \wedge \blacksquare \mathit{unloaded} \wedge \bullet\blacklozenge \mathit{shoot} \rightarrow \bot$$

can be expressed as

```
:- shoot , &tel { <* unloaded & < <? shoot }.
```

or

```
:- &tel { shoot & <* unloaded & < <? shoot }.
```

Wolf, sheep, and cabbage

```
#program always.

item(w;s;c).
opp(l,r). opp(r,l).
eats(w,s). eats(s,c).

#program initial.

at(b,l).
at(X,l) :- item(X).                          % everything at the left bank

#program dynamic.

at(X,A) :- 'at(X,B), m(X), opp(A,B).         % effect axiom for moving item X

at(b,A) :- 'at(b,B), opp(A,B).               % boat is always moving

at(X,A) :- 'at(X,A), not at(X,B), opp(A,B).  % inertia

0 { m(X) : item(X) } 1.                      % choose moving at most one item

#program always.

:- m(X), 'at(b,A), 'at(X,B), opp(A,B).       % we cannot move item X if at the opposite bank

:- eats(X,Y), at(X,A), at(Y,A), opp(A,B), at(b,B).  % we cannot leave them alone

#program final.

:- at(X,l).

#show m/1.
```

telingo's solution

```
$ telingo version 1.0
Reading from wolf.tel
Solving...
Solving...
Solving...
Solving...
Solving...
Solving...
Solving...
Solving...
Answer: 1
 State 0:
 State 1:  m(s)
 State 2:
 State 3:  m(w)
 State 4:  m(s)
 State 5:  m(c)
 State 6:
 State 7:  m(s)
Answer: 2
 State 0:
 State 1:  m(s)
 State 2:
 State 3:  m(c)
 State 4:  m(s)
 State 5:  m(w)
 State 6:
 State 7:  m(s)
SATISFIABLE

Models       : 2
Calls        : 8
Time         : 0.156s (Solving: 0.00s)
CPU Time     : 0.028s
```

Outline

Summary

- *TEL_f*
  - combines *HT* and *LTL* on finite traces
  - reducible to a normal form close to logic programs
  - naturally accounts for dynamic KRR
  - advocates past temporal operators
  - offers embeddings for action languages
  - readily implementable via ASP
- ASP-based systems for *TEL_f*
  - `https://github.com/potassco/telingo`
  - `https://github.com/potassco/tel`

Summary

- *TEL_f*
  - combines *HT* and *LTL* on finite traces
  - reducible to a normal form close to logic programs
  - naturally accounts for dynamic KRR
  - advocates past temporal operators
  - offers embeddings for action languages
  - readily implementable via ASP
- ASP-based systems for *TEL_f*
  - `https://github.com/potassco/telingo`
  - `https://github.com/potassco/tel`
- What's next? Linear dynamic ASP (cf. forthcoming KR'18 paper)
  - extension of *TEL_f*
  - offers Golog-style control