Temporal Answer Set Programming on Finite Traces

Pedro Cabalar, Roland Kaminski, Torsten Schaub, Anna Schuhmann

University of Corunna, Spain  University of Potsdam, Germany

Potassco
Outline

1 Motivation
2 Introduction
3 Language
4 Semantics
5 Compilation
6 Systems
7 Summary
1 Motivation

2 Introduction

3 Language

4 Semantics

5 Compilation

6 Systems

7 Summary
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.
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 problem solving.

- What is ASP good for?
  Solving knowledge-intense combinatorial 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

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
Motivation

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
Motivation

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 . . .
Motivation

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
Motivation

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
Motivation

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).

:- 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).


:- { 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).
```
Robotic intra-logistic
Motivation

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

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

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).


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), moveto(C,C',T), C < C'.
Robotic intra-logistic
Introduction

Outline

1 Motivation
2 Introduction
3 Language
4 Semantics
5 Compilation
6 Systems
7 Summary
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 \textit{LTL}
    - complexity beyond \textit{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
  ~ $LTL_f$ by G. De Giacomo and M. Vardi (2013)
Outline

1 Motivation
2 Introduction
3 Language
4 Semantics
5 Compilation
6 Systems
7 Summary
Regular formulas

Formulas

\[ \varphi ::= a | \bot | \varphi_1 \otimes \varphi_2 \]

where

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

Defined connectives

- \( \top = \neg \bot \)
- \( \neg \varphi = \varphi \rightarrow \bot \)
- \( \varphi \leftrightarrow \psi = (\varphi \rightarrow \psi) \land (\psi \rightarrow \varphi) \)
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 | • for previous |
| S    | for since      |
| T    | for trigger    |

| future | ○ for next |
| U      | for until   |
| R      | for release  |
Temporal formulas

- **Temporal operators**

  \begin{align*}
  \text{past} & : \bullet \quad \text{for previous} \\
  \text{for since} & : S \\
  \text{for trigger} & : T
  \end{align*}

- **Temporal formulas**

  \[ \varphi ::= a \mid \bot \mid \varphi_1 \otimes \varphi_2 \mid \bullet \varphi \mid \varphi_1 S \varphi_2 \mid \varphi_1 T \varphi_2 \mid \circ \varphi \mid \varphi_1 U \varphi_2 \mid \varphi_1 R \varphi_2 \]

- **Defined operators**

  \begin{align*}
  \blacksquare \varphi &= \bot T \varphi & \text{always before} \\
  \lozenge \varphi &= T S \varphi & \text{eventually before} \\
  \mathbb{I} &= \neg \bullet T & \text{initial} \\
  \hat{\bullet} \varphi &= \bullet \varphi \vee \mathbb{I} & \text{weak previous} \\
  \blacksquare \varphi &= \bot R \varphi & \text{always afterward} \\
  \lozenge \varphi &= T U \varphi & \text{eventually afterward} \\
  \mathbb{F} &= \neg \circ T & \text{final} \\
  \hat{\circ} \varphi &= \circ \varphi \vee \mathbb{F} & \text{weak next}
  \end{align*}
Examples

“\textit{If we shoot twice with a gun that was never loaded, it will eventually fail.}”

\[ \square(\text{shoot} \land \Diamond\Diamond\text{shoot} \land \blacksquare\text{unloaded} \rightarrow \Diamond\Diamond\text{fail}) \]

“\textit{Why does shooting a loaded gun fail in unloading it?}”

\[ \square(\Diamond\Diamond \rightarrow \neg\neg(\text{shoot} \land \Diamond\Diamond\text{loaded} \land \text{loaded})) \]
From models to traces

- **Alphabet**: Set $\mathcal{A}$ of atoms
From models to traces

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

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 $H$
  - $H \leq H'$ if $H_i \subseteq H'_i$ for $i = 0..\lambda$
Semantics

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 $H$
  - $H \leq H'$ 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 $\langle H, T \rangle$ of length $\lambda$ over alphabet $A$ satisfies a temporal formula $\varphi$ at time point $k = 0..\lambda$, $k \neq \omega$, written $\langle H, T \rangle, k \models \varphi$, if the following conditions hold:

1. $\langle H, T \rangle, k \not\models \bot$
2. $\langle H, T \rangle, k \models a$ iff $a \in H_k$, for any atom $a \in \mathcal{A}$
3. $\langle H, T \rangle, k \models \varphi \land \psi$ iff $\langle H, T \rangle, k \models \varphi$ and $\langle H, T \rangle, k \models \psi$
4. $\langle H, T \rangle, k \models \varphi \lor \psi$ iff $\langle H, T \rangle, k \models \varphi$ or $\langle H, T \rangle, k \models \psi$
5. $\langle H, T \rangle, k \models \varphi \rightarrow \psi$ iff $\langle H', T \rangle, k \not\models \varphi$ or $\langle H', T \rangle, k \models \psi$, for all $H' \in \{H, T\}$
Satisfaction of temporal formulas

\[ \bullet \varphi \]

\[ \varphi S \psi \]

\[ \varphi T \psi \]

\[ \varnothing \varphi \]

\[ \varphi U \psi \]

\[ \varphi R \psi \]

\[ \varnothing, \varphi \]
Satisfaction of temporal formulas

6. $\langle H, T \rangle, k \models \lozenge \varphi$ iff $k > 0$ and $\langle H, T \rangle, k-1 \models \varphi$

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

8. $\langle H, T \rangle, k \models \varphi T \psi$ iff for all $j = 0..k$, we have $\langle H, T \rangle, j \models \psi$ or $\langle H, T \rangle, i \models \varphi$ for some $i = j+1..k$
Satisfaction of temporal formulas

6. $\langle H, T \rangle, k \models \Box \varphi$ iff $k > 0$ and $\langle H, T \rangle, k-1 \models \varphi$

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

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

9. $\langle H, T \rangle, k \models \Diamond \varphi$ iff $k < \lambda$ and $\langle H, T \rangle, k+1 \models \varphi$

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

11. $\langle H, T \rangle, k \models \varphi R \psi$ iff for all $j = k..\lambda$, we have $\langle H, T \rangle, j \models \psi$ or $\langle H, T \rangle, i \models \varphi$ for some $i = k..j-1$.  □
Satisfaction of temporal formulas

6. \( \langle H, T \rangle, k \models \lozenge \varphi \) iff \( k > 0 \) and \( \langle H, T \rangle, k-1 \models \varphi \)

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

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

9. \( \langle H, T \rangle, k \models \lozenge \varphi \) iff \( k < \lambda \) and \( \langle H, T \rangle, k+1 \models \varphi \)

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

11. \( \langle H, T \rangle, k \models \varphi \mathcal{R} \psi \) iff for all \( j = k..\lambda \), we have \( \langle H, T \rangle, j \models \psi \) or \( \langle H, T \rangle, i \models \varphi \) for some \( i = k..j-1 \).
Satisfaction of (defined) temporal formulas

12. \( \langle H, T \rangle, k \models T \)

13. \( \langle H, T \rangle, k \models \Box \varphi \) iff \( \langle H, T \rangle, i \models \varphi \) for all \( i = 0..k \)

14. \( \langle H, T \rangle, k \models \Diamond \varphi \) iff \( \langle H, T \rangle, i \models \varphi \) for some \( i = 0..k \)

15. \( \langle H, T \rangle, k \models \bot \) iff \( k \equiv 0 \)

16. \( \langle H, T \rangle, k \models \Diamond \varphi \) iff \( k \equiv 0 \) or \( \langle H, T \rangle, k-1 \models \varphi \)

17. \( \langle H, T \rangle, k \models \Box \varphi \) iff \( \langle H, T \rangle, i \models \varphi \) for any \( i = k..\lambda \)

18. \( \langle H, T \rangle, k \models \Diamond \varphi \) iff \( \langle H, T \rangle, i \models \varphi \) for some \( i = k..\lambda \)

19. \( \langle H, T \rangle, k \models \Diamond \varphi \) iff \( k \equiv \lambda \)

20. \( \langle H, T \rangle, k \models \Diamond \varphi \) iff \( k \equiv \lambda \) or \( \langle H, T \rangle, k+1 \models \varphi \)
Emerging temporal logics

- Temporal logic of here-and-there ($THT$)
Emerging temporal logics

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

- Finale
  - $\Diamond F$ enforces finite traces
  - $\neg \Diamond F$ enforces infinite traces
Emerging temporal logics

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

**Finale**
- \( \Diamond F \) enforces finite traces
- \( \neg \Diamond F \) enforces infinite traces

**Excluded middle** (*EM*)
- \( \square (a \lor \neg a) \) for each atom \( a \in A \)

Note: All variants of *THT* are monotonic!
Emerging temporal logics

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

- **Finale**
  - \(\Diamond F\) enforces finite traces
  - \(\neg \Diamond F\) enforces infinite traces

- **Excluded middle** \((EM)\)
  - \(\Box (a \lor \neg a)\) for each atom \(a \in A\)

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

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

- **Finale**
  - $\Diamond F$ enforces finite traces
  - $\neg \Diamond F$ enforces infinite traces

- **Excluded middle** (*EM*)
  - $\Box (a \lor \neg a)$ for each atom $a \in A$

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

Note: All variants of *THT* are monotonic!
Emerging temporal logics

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

- **Finale**
  - \(\Diamond F\) enforces finite traces
  - \(\neg \Diamond F\) enforces infinite traces

- **Excluded middle** \((EM)\)
  - \(\Box (a \lor \neg a)\) for each atom \(a \in A\)

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

- **Note** All variants of \(THT\) are monotonic!
Semantics

Temporal equilibrium logic (TEL)

A total $HT$-trace $\langle T, T \rangle$ is an equilibrium model of a temporal formula $\varphi$, if

1. $\langle T, T \rangle, 0 \models \varphi$,
2. $\langle H, T \rangle, 0 \nvdash \varphi$ for all $H < T$
A total $HT$-trace $\langle T, T \rangle$ is an equilibrium model of a temporal formula $\varphi$, if

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

$T$ is called a temporal stable model of $\varphi$
Semantics

Temporal equilibrium logic ($TEL$)

A total $HT$-trace $\langle T, T \rangle$ is an equilibrium model of a temporal formula $\varphi$, if

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

$T$ is called a temporal stable model of $\varphi$

Examples

- $\square(\neg a \rightarrow \Diamond a)$ yields
  - $(\emptyset \{a\})^\omega$ in $TEL_\omega$ and $(\emptyset \{a\})^+$ in $TEL_f$
Temporal equilibrium logic (TEL)

- A total $HT$-trace $\langle T, T \rangle$ is an equilibrium model of a temporal formula $\varphi$, if

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

- $T$ is called a temporal stable model of $\varphi$

Examples

- $\Box (\neg a \rightarrow \diamond a)$ yields
  - $(\emptyset \{ a \})^\omega$ in $TEL_\omega$ and $(\emptyset \{ a \})^+$ in $TEL_f$
- $\Box (\neg \Box a \rightarrow a) \land \Box (\Box a \rightarrow a)$ yields
  - no model in $TEL_\omega$ but $(\{ a \})^+$ in $TEL_f$
Temporal equilibrium logic \((TEL)\)

- A total \(HT\)-trace \(\langle T, T \rangle\) is an equilibrium model of a temporal formula \(\varphi\), if
  - \(\langle T, T \rangle, 0 \models \varphi\),
  - \(\langle H, T \rangle, 0 \not\models \varphi\) for all \(H < T\)

- \(T\) is called a temporal stable model of \(\varphi\)

- Examples
  - \(\Box (\neg a \rightarrow \bigcirc a)\) yields
    - \((\emptyset \{a\})^\omega\) in \(TEL_\omega\) and \((\emptyset \{a\})^+\) in \(TEL_f\)
  - \(\Box (\neg \bigcirc a \rightarrow a) \land \Box (\bigcirc a \rightarrow a)\) yields
    - no model in \(TEL_\omega\) but \((\{a\})^+\) in \(TEL_f\)
  - \(\Box \Diamond a\) yields
    - no model in \(TEL_\omega\) but \((\emptyset^* \{a\})\) in \(TEL_f\)
Normalform

- **Temporal literals** \( \{ a, \neg a, \bullet a, \neg \bullet a \mid a \in A \} \)

- **Temporal rules**
  - **initial rule** \( B \rightarrow A \)
  - **dynamic rule** \( \hat{o} \square (B \rightarrow A) \)
  - **final rule** \( \square (\Diamond \rightarrow (B \rightarrow A)) \)

where \( B = b_1 \land \cdots \land b_n \) and \( A = a_1 \lor \cdots \lor a_m \)
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.
Compilation

Normalform

- **Temporal literals** \( \{ a, \neg a, \bullet a, \neg \bullet a \mid a \in A \} \)

- **Temporal rules**
  - initial rule \( B \rightarrow A \)
  - dynamic rule \( \hat{\circ} \Box (B \rightarrow A) \)
  - final rule \( \Box (\mathcal{F} \rightarrow (B \rightarrow A)) \)

where \( B = b_1 \land \cdots \land b_n \) and \( A = a_1 \lor \cdots \lor a_m \)
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 \( \varphi \) can be converted into a
  temporal logic program \( THT_f \)-equivalent to \( \varphi \)
Example

\[ \Diamond \Box(\bullet \text{loaded} \land \neg \text{unloaded} \rightarrow \text{loaded}) \]
\[ \Diamond \Box(\text{shoot} \land \bullet \text{loaded} \land \text{loaded} \rightarrow \text{goal}) \]
\[ \Box(\mathcal{F} \rightarrow (\neg \text{goal} \rightarrow \bot)) \]
Bounded translation

- **Temporal literals** at time point $k$

  \[
  \tau_k(a) = a_k \quad \tau_k(\neg a) = \neg a_k \]

  \[
  \tau_k(\bullet a) = a_{k-1} \quad \tau_k(\neg \bullet a) = \neg a_{k-1}
  \]

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

  \[
  \tau_k(r) = \tau_k(a_1) \lor \cdots \lor \tau_k(a_m) \leftarrow \tau_k(b_1) \land \cdots \land \tau_k(b_n)
  \]

- **Temporal logic program** $P$ bounded by finite length $\lambda$

  \[
  \tau_\lambda(P) = \{ \tau_0(r) \mid r \in I(P) \} \cup \{ \tau_k(r) \mid r \in D(P), k = 1..\lambda \} \cup \{ \tau_\lambda(r) \mid r \in F(P) \}
  \]
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
    
    $$\tau_k^*(r) = \tau_k(a_1) \lor \cdots \lor \tau_k(a_m) \leftarrow \tau_k(b_1) \land \cdots \land \tau_k(b_n) \land \neg q_{k+1}$$
  - 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
    
    $$
    \tau_k^*(r) = \tau_k(a_1) \lor \cdots \lor \tau_k(a_m) \leftarrow \tau_k(b_1) \land \cdots \land \tau_k(b_n) \land \neg q_{k+1}
    $$
  - add $q_k$ to each logic program at time point $k$
$tel$

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

Example:

\begin{align*}
\text{\texttt{a}} \land \Diamond (\texttt{a} \rightarrow \texttt{b}) \\
\Diamond (\texttt{F} \rightarrow (\neg \texttt{b} \rightarrow \bot))
\end{align*}
tel

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

Example

\[
\{ \rightarrow a, \quad \widehat{\lozenge} \Box (\bullet a \rightarrow b), \quad \Box (\mathcal{F} \rightarrow (\neg b \rightarrow \bot)) \}
\]

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 $q'(b)$

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**
  - 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”

\[
:- \text{lift}(R,B), \text{robot}(R), \text{box}(B,W), \\
\#\text{sum} \{ C : \text{capacity}(R,C); \\
- V, O : '\text{holding}(R,O,V) \} < W.
\]
telingo’s temporal logic programs

- **initial rule**  
  \[ B \rightarrow A \]

- **dynamic rule**  
  \[ \hat{\circ} \Box (B \rightarrow A) \]

- **final rule**  
  \[ \Box (F \rightarrow (B \rightarrow A)) \]
telingo’s temporal logic programs

- initial rule \( B \rightarrow A \)
- dynamic rule \( \Diamond \Box (B \rightarrow A) \)
- final rule \( \Box (\Diamond \rightarrow (B \rightarrow A)) \)
- always rule \( \Box (B \rightarrow A) \)
telingo’s temporal logic programs

- initial rule: \( B \rightarrow A \) #program initial.
- dynamic rule: \( \Diamond \Box (B \rightarrow A) \) #program dynamic.
- final rule: \( \Box (\mathcal{F} \rightarrow (B \rightarrow A)) \) #program final.
- always rule: \( \Box (B \rightarrow A) \) #program always.
**teingo’s temporal logic programs**

- **initial rule** \( B \rightarrow A \)  
  #program initial.
- **dynamic rule** \( \hat{\circ} \lozenge (B \rightarrow A) \)  
  #program dynamic.
- **final rule** \( \Box (\not F \rightarrow (B \rightarrow A)) \)  
  #program final.
- **always rule** \( \Box (B \rightarrow A) \)  
  #program always.

**Example**  
\{ \rightarrow a, \hat{\circ} \lozenge (\bullet a \rightarrow b), \Box (\not F \rightarrow (\not b \rightarrow \bot)) \}
**telingo’s temporal logic programs**

- initial rule  
  \[ B \rightarrow A \]  
  #program initial.

- dynamic rule  
  \[ \diamond \Box (B \rightarrow A) \]  
  #program dynamic.

- final rule  
  \[ \Box (F \rightarrow (B \rightarrow A)) \]  
  #program final.

- always rule  
  \[ \Box (B \rightarrow A) \]  
  #program always.

**Example**  
\[ \{ \rightarrow a, \diamond \Box (\bullet a \rightarrow b), \Box (F \rightarrow (\neg b \rightarrow \bot)) \} \]

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 I
- &final F
telingo’s temporal formulas

- $\&\text{initial}$  $\top$
- $\&\text{final}$  $\bot$
- $\&\text{tel} \{ \varphi \}$ for temporal formula $\varphi$
telingo’s temporal formulas

- \&initial  \text{I}
- \&final  \text{F}
- \&tel \{ \varphi \} for temporal formula \varphi

Temporal operators

<table>
<thead>
<tr>
<th>past</th>
<th>future</th>
</tr>
</thead>
<tbody>
<tr>
<td>\bullet previous</td>
<td>\circ next</td>
</tr>
<tr>
<td>S</td>
<td>since</td>
</tr>
<tr>
<td>T</td>
<td>trigger</td>
</tr>
<tr>
<td>\Diamond</td>
<td>eventually before</td>
</tr>
<tr>
<td>■</td>
<td>always before</td>
</tr>
<tr>
<td>\Hat</td>
<td>weak previous</td>
</tr>
</tbody>
</table>

Torsten Schaub (KRR@UP) Temporal Answer Set Programming on Finite Traces
### telingo’s temporal formulas

- \&initial \( I \)
- \&final \( F \)
- \&tel \( \{ \varphi \} \) for temporal formula \( \varphi \)

#### Temporal operators

<table>
<thead>
<tr>
<th>past</th>
<th>future</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;=</td>
</tr>
<tr>
<td>&lt;=?</td>
<td>&gt;=?</td>
</tr>
<tr>
<td>&lt;=*</td>
<td>&gt;=*</td>
</tr>
<tr>
<td>&lt;=:</td>
<td>&gt;=:</td>
</tr>
</tbody>
</table>

- previous
- next
- since
- until
- trigger
- release
- eventually before
- eventually afterward
- always before
- always afterward
- weak previous
- weak next
**telingo’s temporal formulas**

- &initial \( I \)
- &final \( F \)
- &tel \( \{ \varphi \} \) for temporal formula \( \varphi \)

**Temporal operators**

<table>
<thead>
<tr>
<th>past</th>
<th>future</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>&lt;?</td>
<td>&gt;?</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;*</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;?</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;*</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&gt;:</td>
</tr>
</tbody>
</table>

- **Boolean operators** & | | ~
**telingo’s temporal formulas**

- \&initial \(I\)
- \&final \(F\)
- \&tel \(\{ \varphi \}\) for temporal formula \(\varphi\)

**Example**

\[
\text{shoot} \land \blacksquare \text{unloaded} \land \Diamond \Diamond \text{shoot} \rightarrow \bot
\]

can be expressed as

\[
:- \text{shoot} , \&\text{tel} \{ \langle\langle \text{unloaded} \rangle \& \langle\langle? \text{shoot}\rangle \}\}
\]
or

\[
:- \&\text{tel} \{ \text{shoot} \& \langle\langle \text{unloaded} \rangle \& \langle\langle? \text{shoot}\rangle \}\}
\]
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 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

1 Motivation
2 Introduction
3 Language
4 Semantics
5 Compilation
6 Systems
7 Summary
**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/telingo)
  - [https://github.com/potassco/tel](https://github.com/potassco/tel)

---

**What's next?** Linear dynamic ASP (cf. forthcoming KR’18 paper) extension of TEL\(_f\) offers Golog-style control
**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/telingo)
  - [https://github.com/potassco/tel](https://github.com/potassco/tel)

- **What’s next?** Linear dynamic ASP (cf. forthcoming KR’18 paper)
  - extension of TEL\(_f\)
  - offers Golog-style control