UNIVERSITÄT GREIFSWALD
Wissen lockt. Seit 1456

Non-classical Logics in the TPTP World

Alexander Steen, University of Greifswald
TPTPTP 2023, Saclay, France

jww. G. Sutcliffe, T. Scholl, C. Benzmüller, D. Fuenmayor

Goal of this talk

Reasoning in non-classical logics (NCLs)

Relevant for many fields:

- Artificial intelligence
  - Knowledge representation
  - Multi-agent systems
  - ...
- Computer science
  - Software verification
  - Hardware verification
  - Cyber-physical systems
  - ...
- Philosophy
  - Formal ethics
  - Metaphysics
  - ...
- many more

What about automated reasoning here?

Goal of this talk

Reasoning in non-classical logics (NCLs)

Relevant for many fields:

- Artificial intelligence
  - Knowledge representation
  - Multi-agent systems
  - ...
- Computer science
  - Software verification
  - Hardware verification
  - Cyber-physical systems
  - ...
- Philosophy
  - Formal ethics
  - Metaphysics
  - ...
- many more

What about automated reasoning here?

NCL automation exists!

Modal logics formats (a specific NCL)

QMLTP: [Raths and Otten, 2011]

```
qmf(con,conjecture,
(( ! [X] : (#box : ( f(X) ) ) ) => (#box : ( ! [X] : ( f(X) ) )))).
```

LWB: [Heuerding et al., 2005]

```
1: ~((p100 & (~p101)) & (((((p101 –> p100) & box(p102 –> p101)) ...
```

MOLTAP: [van Laarhoven, 2009]

```
K_1 M_1 M_3 K_3 K_2 K_3 a –> M_1 M_3 a
```

TANCS2000: [Massacci and Donini, 2000]

```
inputformula(persat2,axiom,
    (v1 | v2 | (box r1 : v3) | (box r1 : v4))
    ).
```

NCL automation exists!

Modal logics formats (a specific NCL)

QMLTP: [Raths and Otten, 2011]

```
qmf(con,conjecture,
(( ! [X] : (#box : ( f(X) ) ) ) => (#box : ( ! [X] : ( f(X) ) )))).
```

LWB: [Heuerding et al., 2005]

```
1: ~((p100 & (~p101)) & (((((p101 -> p100) & box(p102 -> p101)) ...
```

MOLTAP: [van Laarhoven, 2009]

```
K_1 M_1 M_3 K_3 K_2 K_3 a -> M_1 M_3 a
```

TANCS2000: [Massacci and Donini, 2000]

```
inputformula(persat2,axiom,
    (v1 | v2 | (box r1 : v3) | (box r1 : v4))
    ).
```

NCL automation exists!

Modal logics formats (a specific NCL)

QMLTP: [Raths and Otten, 2011]

```
qmf(con,conjecture,
(( ! [X] : (#box : ( f(X) ) ) ) => (#box : ( ! [X] : ( f(X) ) )))).
```

LWB: [Heuerding et al., 2005]

```
1: ~((p100 & (~p101)) & (((((p101 --> p100) & box(p102 --> p101)) ...
```

MOLTAP: [van Laarhoven, 2009]

```
K_1 M_1 M_3 K_3 K_2 K_3 a --> M_1 M_3 a
```

TANCS2000: [Massacci and Donini, 2000]

```
inputformula(persat2,axiom,
    (v1 | v2 | (box r1 : v3) | (box r1 : v4))
    ).
```

An anonymous (potential) user

Heterogeneous landscape for NCL ATPs

NCL automation exists!

Modal logics formats (a specific NCL)

QMLTP: [Raths and Otten, 2011]

```
qmf(con,conjecture,
(( ! [X] : (#box : ( f(X) ) ) ) => (#box : ( ! [X] : ( f(X) ) )))).
```

LWB: [Heuerding et al., 2005]

```
1: ~((p100 & (~p101)) & (((((p101 -> p100) & box(p102 -> p101)) ...
```

MOLTAP: [van Laarhoven, 2009]

```
K_1 M_1 M_3 K_3 K_2 K_3 a -> M_1 M_3 a
```

TANCS2000: [Massacci and Donini, 2000]

```
inputformula(persat2,axiom,
    (v1 | v2 | (box r1 : v3) | (box r1 : v4))
    ).
```

An anonymous (potential) user

To be fair: QMLTP is the de-facto standard for quantified modal logics

and LWB for propositional modal logic (as far as I know)

Other NCLs: Even more (system-specific) formats

- difficult to evaluate/compare within a specific logic
- syntax transformation needed
  - this might change difficulty of problems

There are standards though:

- ILTP/QMLTP syntax (only for intuitionistic/modal logic)
- DFG syntax (has some capacity for expressing different logics)
- Knowledge Interchange Format (KIF) (only first-order)
- Common Logic (CL) (first- and higher-order, XML based)
- OMDoc (representation of mathematical knowledge)
- ...

All of them: Fixing a particular semantics/logic

Some motivation and a disclaimer

Other NCLs: Even more (system-specific) formats

- difficult to evaluate/compare within a specific logic
- syntax transformation needed
  - this might change difficulty of problems

There are standards though:

- ILTP/QMLTP syntax (only for intuitionistic/modal logic)
- DFG syntax (has some capacity for expressing different logics)
- Knowledge Interchange Format (KIF) (only first-order)
- Common Logic (CL) (first- and higher-order, XML based)
- OMDoc (representation of mathematical knowledge)
- ...

All of them: Fixing a particular semantics/logic

Some motivation and a disclaimer

Other NCLs: Even more (system-specific) formats

- difficult to evaluate/compare within a specific logic
- syntax transformation needed
  - this might change difficulty of problems

There are standards though:

- ILTP/QMLTP syntax (only for intuitionistic/modal logic)
- DFG syntax (has some capacity for expressing different logics)
- Knowledge Interchange Format (KIF) (only first-order)
- Common Logic (CL) (first- and higher-order, XML based)
- OMDoc (representation of mathematical knowledge)
- ...

All of them: Fixing a particular semantics/logic

Goal

TPTP is widely accepted in (classical) theorem proving

- Why else would you be here today?

Our goal: 'gracefully' extend to non-classical

- Minimal syntactic changes
- Uniform syntax for all non-classical logics
- Consistency throughout TPTP dialects
- User-friendly syntax
  (easy reading and writing of problems)
- Developer-friendly syntax
  (easy parsing, minimal no. of cases to consider)

Important: We only provide the syntax, not the semantics!

Goal

TPTP is widely accepted in (classical) theorem proving

- Why else would you be here today?

Our goal: 'gracefully' extend to non-classical

- Minimal syntactic changes
- Uniform syntax for all non-classical logics
- Consistency throughout TPTP dialects
- User-friendly syntax
  (easy reading and writing of problems)
- Developer-friendly syntax
  (easy parsing, minimal no. of cases to consider)

Important: We only provide the syntax, not the semantics!

Goal

TPTP is widely accepted in (classical) theorem proving

- Why else would you be here today?

Our goal: ’gracefully’ extend to non-classical

- Minimal syntactic changes
- Uniform syntax for all non-classical logics
- Consistency throughout TPTP dialects
- User-friendly syntax
  (easy reading and writing of problems)
- Developer-friendly syntax
  (easy parsing, minimal no. of cases to consider)

Important: We only provide the syntax, not the semantics!

Typed first-order logic (TXF): Recap on TPTP syntax

```
tff(dog_decl,type, dog: $tType ).
tff(human_decl,type, human: $tType ).
tff(owner_of_decl,type, owner_of: dog > human ).
tff(bit_decl,type, bit: (dog * human * $int) > $o ).
tff(hates_decl,type, hates: (human * human) > $o ).

tff(hate_the_multi_biter_dog,axiom,
   ! [D: dog,H: human,N: $int] :
     ( ( H != owner_of(D) & bit(D,H,N) & $greater(N,1) )
    => hates(H,owner_of(D)) ) ).
```

Higher-order logic (THF): Recap on TPTP syntax

```
thf(dog_decl,type, dog: $tType ).
thf(human_decl,type, human: $tType ).
thf(owner_of_decl,type, owner_of: dog > human ).
thf(owns_decl,type, owns: human > dog > $o ).

thf(owns_defn,definition,
   ( owns = ( ^ [H: human,D: dog] : ( H = ( owner_of @ D ) ) ) ) ).

thf(hate_the_multi_biter_dog,axiom,
   ! [Huddle: dog > $o]: ?[Group: human > $o]:
    ![D: dog]: ? [H: human]:
    ( (Huddle @ D) & (Group @ H) & (owns @ H @ D) ) ).
```

Extend languages with new operator:

- New kind of connective:

```
{ connective_name }
```

- `connective_name` is either TPTP-defined: e.g. `{ $necessary }`, `{ $possible }`, `{ $knows }`, ...
- or `connective_name` is system-defined: e.g. `{ $$future }`, `{ $$obligation }`, `{ $$permission }`, ...

Resulting languages:

- Non-classical extended first-order form (NXF)
  - first-order application style:

```
{ connective_name }(a,b)
```

Extend languages with new operator:

- New kind of connective:

```
{ connective_name }
```

- `connective_name` is either TPTP-defined: e.g. `{ $necessary }`, `{ $possible }`, `{ $knows }`, ...
- or `connective_name` is system-defined: e.g. `{ $$future }`, `{ $$obligation }`, `{ $$permission }`, ...

Resulting languages:

- Non-classical extended first-order form (NXF)
  - first-order application style:

```
{ connective_name }(a,b)
```

Non-Classical TPTP

Extend languages with new operator:

- New kind of connective:

```
{ connective_name }
```

- `connective_name` is either TPTP-defined: e.g. `{ $necessary }`, `{ $possible }`, `{ $knows }`, ...
- or `connective_name` is system-defined: e.g. `{ $$future }`, `{ $$obligation }`, `{ $$permission }`, ...

Resulting languages:

- Non-classical extended first-order form (NXF)
  - first-order-like application style:

```
{ connective_name } @ (a,b)
```

- Non-classical higher-order form (NHF)
  - canonical higher-order application style (curried):

```
{ connective_name } @ a @ b
```

Extend languages with new operator:

- New kind of connective:

  ```
  { connective_name }
  ```

- `connective_name` is either TPTP-defined: e.g. `{ $necessary }`, `{ $possible }`, `{ $knows }`, ...
- or `connective_name` is system-defined: e.g. `{ $$future }`, `{ $$obligation }`, `{ $$permission }`, ...

Resulting languages:

- Non-classical extended first-order form (NXF)
  - first-order-like application style:

    ```
    { connective_name } @ (a,b)
    ```

- Non-classical higher-order form (NHF)
  - canonical higher-order application style (curried):

    ```
    { connective_name } @ a @ b
    ```

Example in NXF:

```
tff(possible_dog_bit_owner,axiom,
   {$dia} @ (? [D: dog] : bit(D,owner_of(D),1)) ).

tff(jon_says_necessary_truth,axiom,
   ! [S: $o] : ( says(jon,S) => {$box} @ (S) ) ).
```

Example in NHF:

```
thf(possible_jon_owns_biter,axiom,
   ! [D: dog] :
    ( ( bit @ D @ jon @ 1 )
    => ( {$dia} @ ( owns @ jon @ D ) ) ) ).

thf(jon_says_he_must_feed_odie,axiom,
   says @ jon @ ({$box} @ (feeds @ jon @ odie)) ).
```

Example in NXF:

```
tff(possible_dog_bit_owner,axiom,
   {$dia} @ (? [D: dog] : bit(D,owner_of(D),1)) ).

tff(jon_says_necessary_truth,axiom,
   ! [S: $o] : ( says(jon,S) => {$box} @ (S) ) ).
```

Example in NHF:

```
thf(possible_jon_owns_biter,axiom,
   ! [D: dog] :
     ( ( bit @ D @ jon @ 1 )
    => ( {$dia} @ ( owns @ jon @ D ) ) ) ).

thf(jon_says_he_must_feed_odie,axiom,
   says @ jon @ ({$box} @ (feeds @ jon @ odie)) ).
```

Parameterized connectives

Optional parameters: Every NCL connective may be parameterized

- For logics with families of operators, e.g. ...
  - multi-modal logics: $\Box_i$
  - term-modal logics: $[t]\varphi$
  - propositional dynamic logic: $[p \cup q]\varphi$
  - epistemic logic: $K_A\varphi$, $C_{\{A,B,C\}}\phi$, ...

Representation: key-value arguments

```
{ connective_name(param1 := value1, param2 := value2, ...) }
```

- ... where the params are functors,
- and the values are arbitrary terms

Allow hashed (#ed) index value as first argument:

```
{ connective_name(#index, param1 := value1, param2 := value2, ...) }
```

Parameterized connectives

Optional parameters: Every NCL connective may be parameterized

- For logics with families of operators, e.g. ...
  - multi-modal logics: $\Box_i$
  - term-modal logics: $[t]\varphi$
  - propositional dynamic logic: $[p \cup q]\varphi$
  - epistemic logic: $K_A\varphi$, $C_{\{A,B,C\}}\phi$, ...

Representation: key-value arguments

```
{ connective_name(param1 := value1, param2 := value2, ...) }
```

- ... where the `params` are functors,
- and the `values` are arbitrary terms

Allow hashed (#ed) index value as first argument:

```
{ connective_name(#index, param1 := value1, param2 := value2, ...) }
```

Parameterized connectives

Optional parameters: Every NCL connective may be parameterized

- For logics with families of operators, e.g. ...
  - multi-modal logics: $\Box_i$
  - term-modal logics: $[t]\varphi$
  - propositional dynamic logic: $[p \cup q]\varphi$
  - epistemic logic: $K_A\varphi$, $C_{\{A,B,C\}}\phi$, ...

Representation: key-value arguments

```
{ connective_name(param1 := value1, param2 := value2, ...) }
```

- ... where the params are functors,
- and the values are arbitrary terms

Allow hashed (#ed) index value as first argument:

```
{ connective_name(#index, param1 := value1, param2 := value2, ...) }
```

Parameterized exmages


```
tff(alice_knows_its_possible_odie_bit_jon,axiom,
   {$knows(#alice)} @ ({$dia} @ (bit(odie,jon,1)) ).
```

```
tff(jon_says_common_knowledge,axiom,
   ! [S: $o] :
     ( says(jon,S) => {$common($agents:=[alice,bob,claire])} @ (S) ) ).
```

```
thf(alice_knows_jon_owns_a_dog,axiom,
   {$knows(#alice)} @
     ? [D: dog] : ( owns @ jon @ D ) ).
```

```
thf(alice_and_bob_know_jon_might_lie,axiom,
   ! [S: $o] :
     ( (says @ jon @ S )
    => {$common($agents:=[alice,bob])} @ ({$dia} @ ~ S) ) ).
```

Parameterized exmages

```
tff(alice_knows_its_possible_odie_bit_jon,axiom,
   {$knows(#alice)} @ ({$dia} @ (bit(odie,jon,1)) ).
```

```
tff(jon_says_common_knowledge,axiom,
   ! [S: $o] :
     ( says(jon,S) => {$common($agents:=[alice,bob,claire])} @ (S) ) ).
```

```
thf(alice_knows_jon_owns_a_dog,axiom,
   {$knows(#alice)} @
     ? [D: dog] : ( owns @ jon @ D ) ).
```

```
thf(alice_and_bob_know_jon_might_lie,axiom,
   ! [S: $o] :
     ( (says @ jon @ S )
    => {$common($agents:=[alice,bob])} @ ({$dia} @ ~ S) ) ).
```

Parameterized exmages

```
tff(alice_knows_its_possible_odie_bit_jon,axiom,
   {$knows(#alice)} @ ({$dia} @ (bit(odie,jon,1)) ).
```

```
tff(jon_says_common_knowledge,axiom,
   ! [S: $o] :
     ( says(jon,S) => {$common($agents:=[alice,bob,claire])} @ (S) ) ).
```

```
thf(alice_knows_jon_owns_a_dog,axiom,
   {$knows(#alice)} @
     ? [D: dog] : ( owns @ jon @ D ) ).
```

```
thf(alice_and_bob_know_jon_might_lie,axiom,
   ! [S: $o] :
     ( (says @ jon @ S )
    => {$common($agents:=[alice,bob])} @ ({$dia} @ ~ S) ) ).
```

Parameterized exmaples

```
tff(alice_knows_its_possible_odie_bit_jon,axiom,
   {$knows(#alice)} @ ({$dia} @ (bit(odie,jon,1)) ).
```

```
tff(jon_says_common_knowledge,axiom,
   ! [S: $o] :
     ( says(jon,S) => {$common($agents:=[alice,bob,claire])} @ (S) ) ).
```

```
thf(alice_knows_jon_owns_a_dog,axiom,
   {$knows(#alice)} @
     ? [D: dog] : ( owns @ jon @ D ) ).
```

```
thf(alice_and_bob_know_jon_might_lie,axiom,
   ! [S: $o] :
     ( (says @ jon @ S )
    => {$common($agents:=[alice,bob])} @ ({$dia} @ ~ S) ) ).
```

Object-level syntax is not enough

From which logic does the formula $p \vee q$ come from?

Classical logic? Wrong! I meant intuitionistic logic.

From which logic does the formula $\Box\phi \rightarrow \phi$ come from?

Modal logic **K**? Wrong! I meant **S5** ...

- Formula syntax alone not enough to let ATP systems know which logic we're in
- Introduce: Logic specification

```
tff(formula_name, logic, logic_name == [ properties ] ).
```

where ...

- *logic_name* is the name of the logic family,
- *properties* is a comma-separated sequence of key-value pairs

Object-level syntax is not enough

From which logic does the formula $p \vee q$ come from?

Classical logic? Wrong! I meant intuitionistic logic.

From which logic does the formula $\Box\phi \to \phi$ come from?

Modal logic **K**? Wrong! I meant **S5** ...

- Formula syntax alone not enough to let ATP systems know which logic we're in
- Introduce: Logic specification

```
tff(formula_name, logic, logic_name == [ properties ] ).
```

where ...

- *logic_name* is the name of the logic family,
- *properties* is a comma-separated sequence of key-value pairs

Object-level syntax is not enough

From which logic does the formula $p \vee q$ come from?

Classical logic? Wrong! I meant intuitionistic logic.

From which logic does the formula $\Box\phi \rightarrow \phi$ come from?

Modal logic **K**? Wrong! I meant **S5** ...

- Formula syntax alone not enough to let ATP systems know which logic we're in
- Introduce: Logic specification

```
tff(formula_name, logic, logic_name == [ properties ] ).
```

where ...

- *logic_name* is the name of the logic family,
- *properties* is a comma-separated sequence of key-value pairs

Object-level syntax is not enough

From which logic does the formula $p \vee q$ come from?

Classical logic? Wrong! I meant intuitionistic logic.

From which logic does the formula $\Box\phi \to \phi$ come from?

Modal logic **K**? Wrong! I meant **S5** ...

- Formula syntax alone not enough to let ATP systems know which logic we're in
- Introduce: Logic specification

```
tff(formula_name, logic, logic_name == [ properties ] ).
```

where ...

- *logic_name* is the name of the logic family,
- *properties* is a comma-separated sequence of key-value pairs

Object-level syntax is not enough

From which logic does the formula $p \vee q$ come from?

Classical logic? Wrong! I meant intuitionistic logic.

From which logic does the formula $\Box\phi \rightarrow \phi$ come from?

Modal logic **K**? Wrong! I meant **S5** ...

- Formula syntax alone not enough to let ATP systems know which logic we're in
- Introduce: Logic specification

> tff(*formula_name*, logic, *logic_name* == [*properties*]).

where ...

- *logic_name* is the name of the logic family,
- *properties* is a comma-separated sequence of key-value pairs

Object-level syntax is not enough

From which logic does the formula $p \vee q$ come from?

Classical logic? Wrong! I meant intuitionistic logic.

From which logic does the formula $\Box\phi \rightarrow \phi$ come from?

Modal logic **K**? Wrong! I meant **S5** ...

- Formula syntax alone not enough to let ATP systems know which logic we're in
- Introduce: Logic specification

> tff(*formula_name*, logic, *logic_name* == [*properties*]).

where ...

- *logic_name* is the name of the logic family,
- *properties* is a comma-separated sequence of key-value pairs

Object-level syntax is not enough

From which logic does the formula $p \lor q$ come from?

Classical logic? Wrong! I meant intuitionistic logic.

From which logic does the formula $\Box\phi \rightarrow \phi$ come from?

Modal logic **K**? Wrong! I meant **S5** ...

- Formula syntax alone not enough to let ATP systems know which logic we're in
- Introduce: Logic specification

tff(*formula_name*, logic, *logic_name* == [*properties*]).

where ...

- *logic_name* is the name of the logic family,
- *properties* is a comma-separated sequence of key-value pairs

Object-level syntax is not enough

From which logic does the formula $p \vee q$ come from?

Classical logic? Wrong! I meant intuitionistic logic.

From which logic does the formula $\Box\phi \rightarrow \phi$ come from?

Modal logic **K**? Wrong! I meant **S5** ...

- Formula syntax alone not enough to let ATP systems know which logic we're in
- Introduce: Logic specification

> `tff(`*formula_name*`, logic, `*logic_name*` == [ `*properties*` ] ).`

where ...

- *logic_name* is the name of the logic family,
- *properties* is a comma-separated sequence of key-value pairs

Case study: Modal logics

Example formulas of modal logic

Mono-modal:

- $\Box \text{raining} \rightarrow \Diamond \text{raining}$
- $\forall P\, (\Diamond \text{rich}(P) \lor \Diamond \neg \text{rich}(P))$
- $\neg \Box (\exists X\, \text{rich}(X))$

Multi-modal:

- $\Box_a \text{raining} \rightarrow \Diamond_b \text{raining}$
- $\forall P\, (\Diamond_b \text{rich}(P) \lor \Diamond_b \neg \text{rich}(P))$
- $\neg \Box_a (\exists X\, \text{rich}(X))$

Example formulas of modal logic

Mono-modal:

- $\Box \text{raining} \rightarrow \Diamond \text{raining}$
- $\forall P\,(\Diamond \text{rich}(P) \lor \Diamond \neg \text{rich}(P))$
- $\neg \Box (\exists X\, \text{rich}(X))$

Multi-modal:

- $\Box_a \text{raining} \rightarrow \Diamond_b \text{raining}$
- $\forall P\,(\Diamond_b \text{rich}(P) \lor \Diamond_b \neg \text{rich}(P))$
- $\neg \Box_a (\exists X\, \text{rich}(X))$

Representation in TPTP

Connectives (mono-modal)

- `{ $box }`
- `{ $dia }`

Connectives (multi-modal)

- `{ $box(#i)}`
- `{ $dia(#i)}`

Examples from above:

```
tff(1, axiom, { $box } @ (raining) => { $dia } @ (raining) ).
tff(2, axiom, ![P]: ( { $dia } @ (rich(P)) | ~({ $dia } @ (rich(P))) ).
tff(3, axiom, ~ { $box } @ ( ? [X]: rich(X) ) ).
```

We also offer user-friendly names: `{ $necessary }`, `{ $possible }`, `{ $knows }`, ...

Representation in TPTP

Connectives (mono-modal)

- `{ $box }`
- `{ $dia }`

Connectives (multi-modal)

- `{ $box(#i)}`
- `{ $dia(#i)}`

Examples from above:

```
tff(1, axiom, { $box } @ (raining) => { $dia } @ (raining) ).
tff(2, axiom, ![P]: ( { $dia } @ (rich(P)) | ~({ $dia } @ (rich(P)))) ).
tff(3, axiom, ~ { $box } @ ( ? [X]: rich(X) ) ).
```

We also offer user-friendly names: `{ $necessary }`, `{ $possible }`, `{ $knows }`, ...

Representation in TPTP

Connectives (mono-modal)

- `{ $box }`
- `{ $dia }`

Connectives (multi-modal)

- `{ $box(#i)}`
- `{ $dia(#i)}`

Examples from above:

```
tff(1, axiom, { $box } @ (raining) => { $dia } @ (raining) ).
tff(2, axiom, ![P]: ( { $dia } @ (rich(P)) | ~({ $dia } @ (rich(P)))) ).
tff(3, axiom, ~ { $box } @ ( ? [X]: rich(X) ) ).
```

We also offer user-friendly names: `{ $necessary }`, `{ $possible }`, `{ $knows }`, ...

Representation in TPTP

Connectives (mono-modal)

- `{ $box }`
- `{ $dia }`

Connectives (multi-modal)

- `{ $box(#i)}`
- `{ $dia(#i)}`

Examples from above:

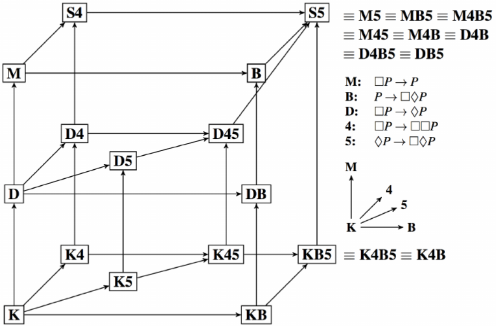
```
tff(1, axiom, { $necessary } @ (raining) => { $possible } @ (raining) ).
tff(2, axiom, ![P]: ( { $possible } @ (rich(P)) | ~({ $possible } @ (rich(P)))) ).
tff(3, axiom, ~ { $necessary } @ ( ? [X]: rich(X) ) ).
```

We also offer user-friendly names: `{ $necessary }`, `{ $possible }`, `{ $knows }`, ...

UNIVERSITÄT GREIFSWALD
Wissen lockt. Seit 1456

Modal logic: A family of many different logics

- Many parameters exist to create more specific modal logics
- Popular example: Properties of the box operator



Logic dimensions of modal logic

1. **Axiomatization of $\Box_i$**
2. **Quantification**
3. **Rigidity**

Logic dimensions of modal logic

1. **Axiomatization of $\Box_i$**
   - What properties does the box operators have?
   - Depending on the application domain

   Some popular axiom schemes:

Name	**Axiom scheme**	**Condition on R_i**	**Corr. formula**
K	$\Box_i(s \supset t) \supset (\Box_i s \supset \Box_i t)$	—	—
B	$s \supset \Box_i \Diamond_i s$	symmetric	$wR_iv \supset vR_iw$
D	$\Box_i s \supset \Diamond_i s$	serial	$\exists v. wR_iv$
T/M	$\Box_i s \supset s$	reflexive	wR_iw
4	$\Box_i s \supset \Box_i \Box_i s$	transitive	$(wR_iv \land vR_iu) \supset wR_iu$
5	$\Diamond_i s \supset \Box_i \Diamond_i s$	euclidean	$(wR_iv \land wR_iu) \supset vR_iu$
...	...	...	...

2. **Quantification**

3. **Rigidity**

Logic dimensions of modal logic

1. **Axiomatization of $\Box_i$**
   - What properties does the box operators have?
2. **Quantification**
   - What is the meaning of $\forall$?
   - Several popular choices exist
     (1) Varying domains: No restrictions
     (2) Constant domains: $\mathcal{D}_w = \mathcal{D}_v$ for all worlds $w, v \in W$
     (3) Cumulative domains: $\mathcal{D}_w \subseteq \mathcal{D}_v$ whenever $(w, v) \in R^i$
     (4) Decreasing domains: $\mathcal{D}_w \supseteq \mathcal{D}_v$ whenever $(w, v) \in R^i$
3. **Rigidity**

Logic dimensions of modal logic

1. **Axiomatization of $\Box_i$**
   - What properties does the box operators have?
2. **Quantification**
   - What is the meaning of $\forall$?
3. **Rigidity**
   - Do all constants $c \in \Sigma$ denote the same object at every world?
   - Several popular choices exist
     (1) Flexible constants: $\mathcal{I}_w$ may vary for each world w
     (2) Rigid constants: $\mathcal{I}_w(c) = \mathcal{I}_v(c)$
     for all worlds $w, v \in W$ and all $c \in \Sigma$

Logic dimensions of modal logic

1. **Axiomatization of $\Box_i$**
   - What properties does the box operators have?
2. **Quantification**
   - What is the meaning of $\forall$?
3. **Rigidity**
   - Do all constants $c \in \Sigma$ denote the same object at every world?

$\longrightarrow$ at least $10 \times 4 \times 2 = 80$ distinct logics

Use logic specification to encode specific logic

`tff(`*formula_name*`, logic, $modal == [ `*properties*` ] ).`

- `$modalities` for the properties of $\Box_i$
- `$domains` for the properties of $\mathcal{D}_w$
- `$designation` for the properties of $\mathcal{I}_w$

Allowed values:

Use logic specification to encode specific logic

> `tff(`*formula_name*`, logic, $modal == [` *properties* `] ).`

- `$modalities` for the properties of $\Box_i$
- `$domains` for the properties of $\mathcal{D}_w$
- `$designation` for the properties of $\mathcal{I}_w$

Allowed values:

`$modalities:` `$modal_system_`X or [`$modal_axiom_`Y_1`,...,$modal_axiom_`Y_n]
for ...
$X \in \{K, KB, K4, K5, K45, KB5, \ldots, S4, S5\}$,
$Y_i \in \{K, T, B, D, 4, 5, C\}$.

`$domains:` `$constant`, `$varying`, `$cumulative`, `$decreasing`

`$designation:` `$rigid`, `$flexible`

- May be assigned per modality/type/constant/...

Example

Simple example:

```
tff(simple_spec,logic,
   $modal == [
     $designation == $rigid,
     $domains == [ $constant, some_user_type == $varying ],
     $modalities == $modal_system_S5 ] ).
```

More complex example:

```
tff(complex_spec,logic,
  $modal == [
    $designation == [ $flexible, sun == $rigid ],
    $domains == [ $constant,
                  planet_type == $varying],
    $modalities == [ $modal_system_K,
                  {$box(#1)} == $modal_system_KB,
                  {$box(#2)} == [ $modal_axiom_K,
                              $modal_axiom_4 ] ] ).
```

Example

Simple example:

```
tff(simple_spec,logic,
   $modal == [
     $designation == $rigid,
     $domains == [ $constant, some_user_type == $varying ],
     $modalities == $modal_system_S5 ] ).
```

More complex example:

```
tff(complex_spec,logic,
 $modal == [
  $designation == [ $flexible, sun == $rigid ],
  $domains == [ $constant,
                planet_type == $varying],
  $modalities == [ $modal_system_K,
                {$box(#1)} == $modal_system_KB,
                {$box(#2)} == [ $modal_axiom_K,
                                $modal_axiom_4 ] ] ).
```

State of TPTP and tool support

TPTP integration

- Modal logics planned for next TPTP release (v9.X?)
- Tool available for QMLTP -> TPTP translation
- TPTP4X utility to be extended to bridge to other syntax formats
- Syntax prolog parseable
- scala-tptp-parser package available

Automation of modal logics

- Offer semantical embedding to HOL for flexible automation
- Tool available: LET (Logic Embedding Tool)
- Accessible via SystemBeforeTPTP
- Included into Leo-III, accessible via SystemOnTPTP

State of TPTP and tool support

TPTP integration

- Modal logics planned for next TPTP release (v9.X?)
- Tool available for QMLTP -> TPTP translation
- TPTP4X utility to be extended to bridge to other syntax formats
- Syntax prolog parseable
- scala-tptp-parser package available

Automation of modal logics

- Offer semantical embedding to HOL for flexible automation
- Tool available: LET (Logic Embedding Tool)
- Accessible via SystemBeforeTPTP
- Included into Leo-III, accessible via SystemOnTPTP

Automation method: Logic Embeddings

Shallow semantical embeddings

- Encode semantics of logic L into HOL
- Translate problem using encoding function $\lceil . \rceil$
- Pass translated problem to HOL systems

$$\begin{tikzcd} & \text{HOL} \arrow[dl] \arrow[dr] & \\ \text{Syntax L} \arrow[rr, dashed] & & \text{Semantics L} \end{tikzcd}$$

Off-the-shelf reasoning for non-classical logics

$$\Psi \models^{L} \varphi \qquad \text{iff} \qquad \{\lceil \psi \rceil \mid \psi \in \Psi\} \cup \mathrm{meta}(L) \models^{HOL} \lceil \varphi \rceil$$

Enabling technology for

- Explorative logical analysis
- Empirical studies
- Automation prototyping

... *fruitful for, e.g., deontic logic context: Logical setting still not settled upon*

Automation method: Logic Embeddings

Shallow semantical embeddings

- Encode semantics of logic L into HOL
- Translate problem using encoding function $\lceil . \rceil$
- Pass translated problem to HOL systems

$$\begin{tikzcd} & \text{HOL} \arrow[dl] \arrow[dr] & \\ \text{Syntax L} \arrow[rr, dashed] & & \text{Semantics L} \end{tikzcd}$$

Off-the-shelf reasoning for non-classical logics

$$\Psi \models^{L} \varphi \qquad \text{iff} \qquad \{\lceil \psi \rceil \mid \psi \in \Psi\} \cup \text{meta}(L) \models^{\text{HOL}} \lceil \varphi \rceil$$

Enabling technology for

- Explorative logical analysis
- Empirical studies
- Automation prototyping

... fruitful for, e.g., deontic logic context: Logical setting still not settled upon

Automation method: Logic Embeddings

Shallow semantical embeddings

- Encode semantics of logic L into HOL
- Translate problem using encoding function $\lceil . \rceil$
- Pass translated problem to HOL systems

$$\begin{tikzcd} & \text{HOL} \arrow[dl] \arrow[dr] & \\ \text{Syntax L} \arrow[rr, dashed] & & \text{Semantics L} \end{tikzcd}$$

Off-the-shelf reasoning for non-classical logics

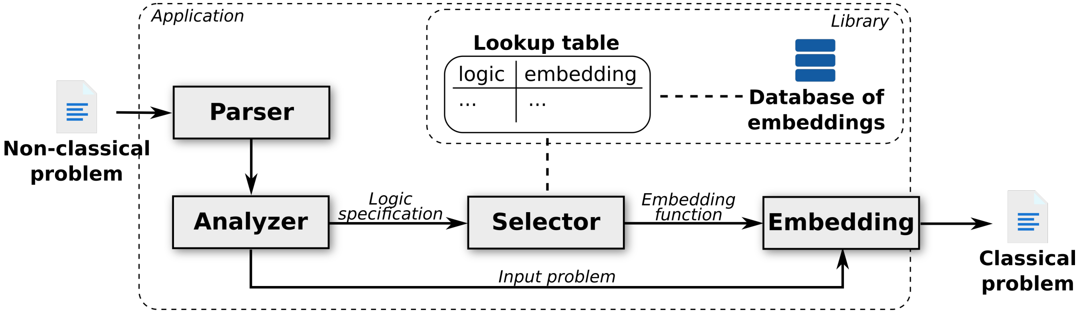
$$\Psi \models^{L} \varphi \qquad \text{iff} \qquad \{\lceil \psi \rceil \mid \psi \in \Psi\} \cup \mathrm{meta}(L) \models^{\mathrm{HOL}} \lceil \varphi \rceil$$

Enabling technology for

- Explorative logical analysis
- Empirical studies
- Automation prototyping

... fruitful for, e.g., deontic logic context: Logical setting still not settled upon

Top-level picture: Logic Embedding Tool (LET)



- Implemented in Scala, available at GitHub `leoprover/logic-embedding`
- Outputs classical TPTP THF (monomorphic or polymorphic)

Possible (exotic?) future

Dynamic epistemic logics (PAL)

- Does $[\varphi!]\varphi$ hold?

```
tff(c1, conjecture, {$box($announce := phi)} @ (phi) ).
```

- Does $[\varphi!]\,(C_{a,b,c}\,\varphi)$ hold?

```
tff(c2, conjecture, {$box($announce := phi)} @ ({common($agents := [a,b,c])} @ (phi)) ).
```

(Propositional) Dynamic logics (PDL)

- Does $[p \cup (p;q)^*]\varphi$ hold?

```
tff(c3, conjecture, {$box($program := $choice(p,$loop($sequence([p,q]))))} @ (phi) ).
```

Term modal logics

- Does $\Box_{f(x)}\varphi$ hold?

```
tff(c4, conjecture, {$box($term := f(X))} @ (phi) ).
```

Of course, concrete syntax needs to be discussed.

Possible (exotic?) future

Dynamic epistemic logics (PAL)

- Does $[\varphi!]\varphi$ hold?

```
tff(c1, conjecture, {$box($announce := phi)} @ (phi) ).
```

- Does $[\varphi!]\,(C_{a,b,c}\,\varphi)$ hold?

```
tff(c2, conjecture, {$box($announce := phi)} @ ({common($agents := [a,b,c])} @ (phi)) ).
```

(Propositional) Dynamic logics (PDL)

- Does $[p \cup (p;q)^*]\varphi$ hold?

```
tff(c3, conjecture, {$box($program := $choice(p,$loop($sequence([p,q]))))} @ (phi) ).
```

Term modal logics

- Does $\Box_{f(x)}\varphi$ hold?

```
tff(c4, conjecture, {$box($term := f(X))} @ (phi) ).
```

Of course, concrete syntax needs to be discussed.

Possible (exotic?) future

Dynamic epistemic logics (PAL)

- Does $[\varphi!]\varphi$ hold?

```
tff(c1, conjecture, {$box($announce := phi)} @ (phi) ).
```

- Does $[\varphi!]\,(C_{a,b,c}\;\varphi)$ hold?

```
tff(c2, conjecture, {$box($announce := phi)} @ ({common($agents := [a,b,c])} @ (phi)) ).
```

(Propositional) Dynamic logics (PDL)

- Does $[p \cup (p;q)^*]\varphi$ hold?

```
tff(c3, conjecture, {$box($program := $choice(p,$loop($sequence([p,q]))))} @ (phi) ).
```

Term modal logics

- Does $\Box_{f(x)}\varphi$ hold?

```
tff(c4, conjecture, {$box($term := f(X))} @ (phi) ).
```

Of course, concrete syntax needs to be discussed.

Possible (exotic?) future

Dynamic epistemic logics (PAL)

- Does $[\varphi!]\varphi$ hold?

```
tff(c1, conjecture, {$box($announce := phi)} @ (phi) ).
```

- Does $[\varphi!]\,(C_{a,b,c}\ \varphi)$ hold?

```
tff(c2, conjecture, {$box($announce := phi)} @ ({common($agents := [a,b,c])} @ (phi)) ).
```

(Propositional) Dynamic logics (PDL)

- Does $[p \cup (p;q)^*]\varphi$ hold?

```
tff(c3, conjecture, {$box($program := $choice(p,$loop($sequence([p,q]))))} @ (phi) ).
```

Term modal logics

- Does $\Box_{f(x)}\varphi$ hold?

```
tff(c4, conjecture, {$box($term := f(X))} @ (phi) ).
```

Of course, concrete syntax needs to be discussed.

Possible (exotic?) future

Dynamic epistemic logics (PAL)

- Does $[\varphi!]\varphi$ hold?

```
tff(c1, conjecture, {$box($announce := phi)} @ (phi) ).
```

- Does $[\varphi!]\,(C_{a,b,c}\;\varphi)$ hold?

```
tff(c2, conjecture, {$box($announce := phi)} @ ({common($agents := [a,b,c])} @ (phi)) ).
```

(Propositional) Dynamic logics (PDL)

- Does $[p \cup (p;q)^*]\varphi$ hold?

```
tff(c3, conjecture, {$box($program := $choice(p,$loop($sequence([p,q]))))} @ (phi) ).
```

Term modal logics

- Does $\Box_{f(x)}\varphi$ hold?

```
tff(c4, conjecture, {$box($term := f(X))} @ (phi) ).
```

Of course, concrete syntax needs to be discussed.

Limits and tools

Limits of the syntax

- There is bound to be a limit on what the extension syntactically express
- ... we did not yet discover it.

Can you provide us with logic representation challenges?

Tools

- *... what tools to we still need?*

Limits and tools

Limits of the syntax

- There is bound to be a limit on what the extension syntactically express
- ... we did not yet discover it.

Can you provide us with logic representation challenges?

Tools

- *... what tools to we still need?*

Summary

- Quite generic syntax extension
- Current focus: Modal logic (as proof-of-concept)
- More logics to come (with your help?)

More information at `tptp.org`:

TPTP subprojects:

- **System on TPTP** Services for solving problems and recommending systems.
- **TSTP** The TSTP (Thousands of Solutions from Theorem Provers) Solution Library is a lib automated theorem proving (ATP) systems. In particular, it contains solutions to TPTP pr
- **TMTP** The TMTP (Thousands of Models for Theorem Provers) Model Library is a library of m theorem proving (ATP) systems. In particular, it contains models for TPTP axiomatizatio
- **CASC** The TPTP is used to supply problems for the CADE ATP System Competition.
- **TPTP Proposals** We are working on new features for the TPTP, as explained in the following proposals. appreciated.
  - Extended TFF (TFX), Updated THF
  - Non-classical Logics
  - Answer Extraction
  - The TPTP Process Instruction Language
  - ATP System Building Conventions