UNIVERSITÄT GREIFSWALD
Wissen lockt. Seit 1456

The NTF format for non-classical logics

Alexander Steen, University of Greifswald
TPTPTP 2024, Nancy, France

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

Goal

TPTP is the standard in (classical) theorem proving

Our goal: 'gracefully' extend to non-classical logics

- 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)

This talk: Describe extensions to TPTP (focus on language)

Goal

TPTP is the standard in (classical) theorem proving

Our goal: ’gracefully’ extend to non-classical logics

- 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)

This talk: Describe extensions to TPTP (focus on language)

Goal

TPTP is the standard in (classical) theorem proving

Our goal: ’gracefully’ extend to non-classical logics

- 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)

This talk: Describe extensions to TPTP (focus on language)

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 language: NTF (non-classical typed form)

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

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

- Non-classical typed 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 language: NTF (non-classical typed form)

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

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

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

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

NTF: 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 language: NTF (non-classical typed form)

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

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

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

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

Non-Classical TPTP

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 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) ) ).
```

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) ) ).
```

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) ) ).
```

Note on representation in TPTP

A note on the format

- NTF is a result of different deliberate design decisions:
  - Minimal parser extensions if TXF (or THF) is already supported
  - Prolog parsing compability (long-standing principle)
  - Syntax should be general enough to cover many (complicated) NCLs
  - Distinction of object-language and meta-language components

UNIVERSITÄT GREIFSWALD
Wissen lockt. Seit 1456

A note on the format

- NTF is a result of different deliberate design decisions:
  - Minimal parser extensions if TXF (or THF) is already supported
  - Prolog parsing compability (long-standing principle)
  - Syntax should be general enough to cover many (complicated) NCLs
  - Distinction of object-language and meta-language components

Advantages:
(1) n-ary operators remain n-ary: `{$box(#i)} @ (phi)` for $\Box_i \varphi$

(2) operators always use the same number of arguments (also in case NCL has e.g. indexed and unindexed box)

(3) no typing issues (meta-level objects – like index *i* – may not be part of the term/formula language)

A note on the format

- NTF is a result of different deliberate design decisions:
  - Minimal parser extensions if TXF (or THF) is already supported
  - Prolog parsing compability (long-standing principle)
  - Syntax should be general enough to cover many (complicated) NCLs
  - Distinction of object-language and meta-language components
- NTF clearly is (a bit) more complex than single-purpose languages
  - If we only need to support (indexed) modal operators, things are syntactically simpler
  - If we only need to support unary (binary) NCL operators, things are syntactically simpler
  - ...

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 $p \vee q$ 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 $p \vee q$ 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 $p \vee q$ 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

As a start: Focused on (quantified) modal logics

Examples

Example formulas of modal logics

Mono-modal:

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

Multi-modal:

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

Example formulas of modal logics

Mono-modal:

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

Multi-modal:

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

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 }`, ...

Quantified Modal Logics in TPTP

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)}`

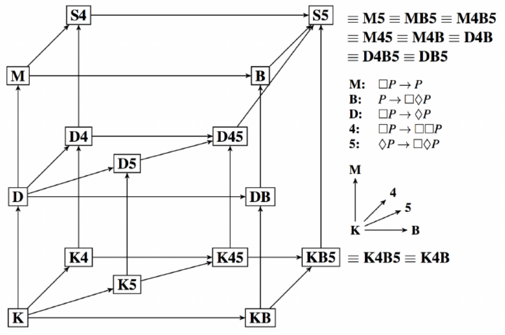
Examples from above:

```
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 }`, ...

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 domains**
3. **Rigid/flexible designation of symbols**
4. **Term locality**

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 domains**

3. **Rigid/flexible designation of symbols**

4. **Term locality**

Logic dimensions of modal logic

1. **Axiomatization of $\Box_i$**
   - What properties does the box operators have?
2. **Quantification domains**
   - 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. **Rigid/flexible designation of symbols**
4. **Term locality**

Logic dimensions of modal logic

1. **Axiomatization of $\Box_i$**
   - What properties does the box operators have?
2. **Quantification domains**
   - What is the meaning of $\forall$?
3. **Rigid/flexible designation of symbols**
   - 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 $c \in \Sigma$
4. **Term locality**

Logic dimensions of modal logic

1. **Axiomatization of $\Box_i$**
   - What properties does the box operators have?
2. **Quantification domains**
   - What is the meaning of $\forall$?
3. **Rigid/flexible designation of symbols**
   - Do all constants $c \in \Sigma$ denote the same object at every world?
4. **Term locality**
   - What is the domain for the interpretation of terms t?
   - At least two common possibilites:
     (1) Local terms: Interpretation of t at world w is an element of $\mathcal{D}_w$
     (2) Global terms: Interpretation of t at world w is some element from $\bigcup_{w \in W} \mathcal{D}_w$

Logic dimensions of modal logic

1. **Axiomatization of $\Box_i$**
   - What properties does the box operators have?
2. **Quantification domains**
   - What is the meaning of $\forall$?
3. **Rigid/flexible designation of symbols**
   - Do all constants $c \in \Sigma$ denote the same object at every world?
4. **Term locality**
   - What is the domain for the interpretation of terms t?

Logic dimensions of modal logic

1. **Axiomatization of $\Box_i$**
   - What properties does the box operators have?
2. **Quantification domains**
   - What is the meaning of $\forall$?
3. **Rigid/flexible designation of symbols**
   - Do all constants $c \in \Sigma$ denote the same object at every world?
4. **Term locality**
   - What is the domain for the interpretation of terms t?

$\longrightarrow$ many different 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 $\mathcal{I}_w$
- `$terms` for the locality properties

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 $\mathcal{I}_w$
- `$terms` for the locality properties

Allowed values:

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

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

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

`$terms`: `$global`, `$local`

Example

Simple example:

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

More complex example:

```
tff(complex_spec,logic,
 $modal == [
   $designation == [ $flexible, sun == $rigid ],
   $domains     == [ $constant,
                        planet_type == $varying],
   $terms       == $local,
   $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 ],
      $terms       == $global,
      $modalities  == $modal_system_S5 ] ).
```

More complex example:

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

TPTP integration

- TPTP v9.0.0 contains 147 NTF (monomodal logic) problems: 132 NXF + 15 NHF
- TPTP4X utility for NTF
- proof verification via GDV
- AGMV model verifier
- IDV derivation viewer (for NTF)
- IIV interpretation viewer for Kripke models
- `scala-tptp-parser` package available

Automation of modal logics

Existing translations to bridge to ...

- KSP
- nanoCoP-M
- MleanCoP
- any TFF/THF reasoner via Logic Embedding Tool

State of TPTP and tool support

TPTP integration

- TPTP v9.0.0 contains 147 NTF (monomodal logic) problems: 132 NXF + 15 NHF
- TPTP4X utility for NTF
- proof verification via GDV
- AGMV model verifier
- IDV derivation viewer (for NTF)
- IIV interpretation viewer for Kripke models
- `scala-tptp-parser` package available

Automation of modal logics

Existing translations to bridge to ...

- KSP
- nanoCoP-M
- MleanCoP
- any TFF/THF reasoner via Logic Embedding Tool

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)) ).
```

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)) ).
```

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)) ).
```

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.

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)) ).
```

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.

Summary

- NTF: Generic NCL syntax extension of the TPTP
  - Current focus: Modal logic
- NTF problems in TPTP v9.0.0
- Solutions as usual in the TSTP
- Many TPTP tools and infrastructure extended to NTF
- More logics to come (with <u>your</u> help?)

Not discussed here:

- There exist means of automation for the presented logics
  - Based on shallow semantical embedding to HOL
- Recent experients for quantified modal logics (QMLTP): Competitive performance wrt. native modal logic provers