Working with Automated Reasoning Tools – The THF0 Language –

Christoph Benzmüller and Geoff Sutcliffe

SS08, Block Course at Saarland University, Germany

Overview

- Motivation
- The TPTP THF0 Language
  - Syntax
  - Type checking
  - Semantics
  - Infrastructure
  - Problem collection
- Extensions
- Outlook

Motivation

We want to foster

- development of higher-order ATPs
- friendly and constructive competition
  - amongst higher-order ATPs
  - between higher-order ATPs and first-order ATPs
- applications of higher-order ATPs
- standard interfaces to higher-order ATPs from
  - proof assistants, ontology/knowledge engineering environments, ...

Our approach

- set-up of a higher order TPTP infrastructure

Motivation

We want to foster

- development of higher-order ATPs
- friendly and constructive competition
  - amongst higher-order ATPs
  - between higher-order ATPs and first-order ATPs
- applications of higher-order ATPs
- standard interfaces to higher-order ATPs from
  - proof assistants, ontology/knowledge engineering environments, ...

Our approach

- set-up of a higher order TPTP infrastructure

Motivation

We want to foster

- development of higher-order ATPs
- friendly and constructive competition
  - amongst higher-order ATPs
  - between higher-order ATPs and first-order ATPs
- applications of higher-order ATPs
- standard interfaces to higher-order ATPs from
  - proof assistants, ontology/knowledge engineering environments, . . .

Our approach

- set-up of a higher order TPTP infrastructure

Motivation

We want to foster

- development of higher-order ATPs
- friendly and constructive competition
  - amongst higher-order ATPs
  - between higher-order ATPs and first-order ATPs
- applications of higher-order ATPs
- standard interfaces to higher-order ATPs from
  - proof assistants, ontology/knowledge engineering environments, . . .

Our approach

- set-up of a higher order TPTP infrastructure

Motivation

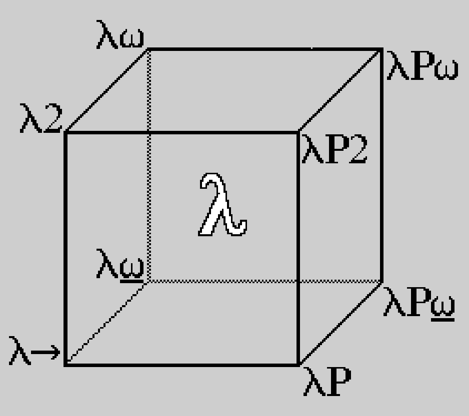
We want to foster

- development of higher-order ATPs
- friendly and constructive competition
  - amongst higher-order ATPs
  - between higher-order ATPs and first-order ATPs
- applications of higher-order ATPs
- standard interfaces to higher-order ATPs from
  - proof assistants, ontology/knowledge engineering environments, ...

Our approach

- set-up of a higher order TPTP infrastructure

Many Higher Order Logics



Starting point: **Church's simple type theory (STT)**

THF0 for Church's STT

- Syntax
  $\longrightarrow$ conservative extension of FOF
  $\longrightarrow$ Prolog readability
- Type Checking
  $\longrightarrow$ via translation to Twelf
- Semantics
  $\longrightarrow$ not fixed
  $\longrightarrow$ annotations in file headers

Syntax of THF0

- Conservative extension of FOF
- Prolog readability

Disadvantages

- BNF rather complicated
- some sacrifices, e.g., @-operator

Advantages

- reuse of existing infrastructure

Syntax of THF0

- Conservative extension of FOF
- Prolog readability

Disadvantages

- BNF rather complicated
- some sacrifices, e.g., @-operator

Advantages

- reuse of existing infrastructure

Syntax of THF0

- Conservative extension of FOF
- Prolog readability

Disadvantages

- BNF rather complicated
- some sacrifices, e.g., @-operator

Advantages

- reuse of existing infrastructure

THF0: Simple Types

o	Booleans	`$o`
ι	individuals	`$i`
$\alpha \rightarrow \beta$	function types	`a>b`
$u, \ldots$	opt. base types	`thf(u,type,(u:$tType)).`

THF0: Simple Types

o	Booleans	`$o`
ι	individuals	`$i`
$\alpha \to \beta$	function types	`a>b`
$u, \ldots$	opt. base types	`thf(u,type,(u:$tType)).`

THF0: Terms and Formulas

X_α	variables	`![X:($i>$i)]: ...`
p_α	constants	`thf(p,type, ( p:($i>$o) ) ).`
$\neg$	logical conn.	`~`
$\vee, \wedge, \Leftrightarrow, \Rightarrow, \ldots$		`|,&,<=>, =>, ...`
$\forall^\alpha, \exists^\alpha, \Pi^\alpha, \Sigma^\alpha$		`!,?,!!,??`
$(p\, q)$	application	`(p @ q)`
$(\lambda X_\alpha\ t)$	λ-abstraction	`^[X:a]:t`

THF0: Terms and Formulas

X_α — variables — `![X:($i>$i)]: ...`

p_α — constants — `thf(p,type, ( p:($i>$o) ) ).`

$\neg$ — logical conn. — `~`

$\vee,\ \wedge,\ \Leftrightarrow,\ \Rightarrow,\ \ldots$ — `|,&,<=>, =>, ...`

$\forall^\alpha,\ \exists^\alpha,\ \Pi^\alpha,\ \Sigma^\alpha$ — `!,?,!!,??`

$(p\,q)$ — application — `(p @ q)`

$(\lambda X_\alpha\ t)$ — λ-abstraction — `^[X:a]:t`

THF0: Terms and Formulas

X_α	variables	`![X:($i>$i)]: ...`
p_α	constants	`thf(p,type, ( p:($i>$o) ) ).`
$\neg$	logical conn.	`~`
$\vee, \wedge, \Leftrightarrow, \Rightarrow, \ldots$		`|,&,<=>, =>, ...`
$\forall^\alpha, \exists^\alpha, \Pi^\alpha, \Sigma^\alpha$		`!,?,!!,??`
$(p\, q)$	application	`(p @ q)`
$(\lambda X_\alpha\ t)$	λ-abstraction	`^[X:a]: t`

THF0: Terms and Formulas

X_α	variables	`![X:($i>$i)]: ...`
p_α	constants	`thf(p,type,` `( p:($i>$o) ) ).`
$\neg$	logical conn.	`~`
$\vee,\ \wedge,\ \Leftrightarrow,\ \Rightarrow,\ \ldots$		`|,&,<=>, =>, ...`
$\forall^\alpha,\ \exists^\alpha,\ \Pi^\alpha,\ \Sigma^\alpha$		`!,?,!!,??`
$(p\,q)$	application	`(p @ q)`
$(\lambda X_\alpha\ t)$	λ-abstraction	`^[X:a]: t`

THF0: Terms and Formulas

X_α	variables	`![X:($i>$i)]: ...`
p_α	constants	`thf(p,type,` `( p:($i>$o) ) ).`
$\neg$	logical conn.	`~`
$\vee,\ \wedge,\ \Leftrightarrow,\ \Rightarrow,\ \ldots$		`|,&,<=>, =>, ...`
$\forall^\alpha,\ \exists^\alpha,\ \Pi^\alpha,\ \Sigma^\alpha$		`!,?,!!,??`
$(p\,q)$	application	`(p @ q)`
$(\lambda X_\alpha\ t)$	λ-abstraction	`^[X:a]:t`

Example: SET171^1.p

```
%------------------------------------------------------------------
%----Signatures for basic set theory predicates and functions.
thf(const_in,type,(
    in: $i > ( $i > $o ) > $o  )).

thf(const_intersection,type,(
    intersection: ( $i > $o ) > ( $i > $o ) > ( $i > $o ) )).

thf(const_union,type,(
    union: ( $i > $o ) > ( $i > $o ) > ( $i > $o ) )).
```

Example: SET171^1.p

```
%----Some axioms for basic set theory.
thf(ax_in,axiom,(
    ( in
    = ( ^ [X: $i, S: ( $i > $o )] :
          ( S @ X ) ) ) )).

thf(ax_intersection,axiom,(
    ( intersection
    = ( ^ [S1: ( $i > $o ), S2: ( $i > $o )]: ^ [U: $i] :
          ( ( in @ U @ S1 )
          & ( in @ U @ S2 ) ) ) ) )).

thf(ax_union,axiom,(
    ( union
    = ( ^ [S1: ( $i > $o ), S2: ( $i > $o )]: ^ [U: $i] :
          ( ( in @ U @ S1 )
          | ( in @ U @ S2 ) ) ) ) )).
```

Example

```
%----The distributivity of union over intersection.
thf(thm_distr,conjecture,(
    ! [A: ( $i > $o ),B: ( $i > $o ),C: ( $i > $o )] :
      ( ( union @ A @ ( intersection @ B @ C ) )
      = ( intersection @ ( union @ A @ B )
                       @ ( union @ A @ C ) ) ) )).
%------------------------------------------------------
```

Beware

You can talk lots of rubbish in THF0 . . .

- `& = ~`
- `![X:$i]: X`
- `(((^[X:$i]: (X @ X)) @ (^[X:$i]: (X @ X)))`

because **TPTP does not provide type checking**

Type Checking

- How do we do it?

Intuitive Type Checking Rules for THF0

$$\frac{}{\texttt{\$i :: \$tType}} \qquad \frac{}{\texttt{\$o :: \$tType}} \qquad \frac{}{\Gamma \vdash \texttt{\$true :: \$o}} \qquad \frac{\texttt{thf(\_,type,c : A)}}{\Gamma \vdash \texttt{c :: A}}$$

$$\frac{\texttt{A :: \$tType} \quad \texttt{B :: \$tType}}{\texttt{A > B :: \$tType}} \qquad \frac{\texttt{X :: A} \text{ in } \Gamma}{\Gamma \vdash \texttt{X :: A}}$$

$$\frac{\Gamma, \texttt{X :: A} \vdash \texttt{S :: B}}{\Gamma \vdash \texttt{\^{} [X : A] : S :: A > B}} \qquad \frac{\Gamma \vdash \texttt{F :: A > B} \quad \Gamma \vdash \texttt{S :: A}}{\Gamma \vdash \texttt{F @ S :: B}}$$

$$\frac{\Gamma \vdash \texttt{F :: A > \$o}}{\Gamma \vdash \texttt{!! F :: \$o}} \qquad \frac{\Gamma, \texttt{X :: A} \vdash \texttt{F :: \$o}}{\Gamma \vdash \texttt{! [X : A]: F :: \$o}} \qquad \frac{\Gamma \vdash \texttt{F :: \$o}}{\Gamma \vdash \texttt{\~{} F :: \$o}}$$

$$\frac{\Gamma \vdash \texttt{F :: \$o} \quad \Gamma \vdash \texttt{G :: \$o}}{\Gamma \vdash \texttt{F \& G :: \$o}} \qquad \frac{\Gamma \vdash \texttt{S :: A} \quad \Gamma \vdash \texttt{S' :: A}}{\Gamma \vdash \texttt{S = S' :: \$o}}$$

Type Checking

Type checking of THF0 is handled **via transalation to logical framework LF**

- LF particularly suited for the task
- context, substitution, α-renaming naturally handled in LF
- LF is well suited also for further (ongoing) extensions of THF0
  - polymorphism
  - dependent types
  - representation of proofs and proof checking

We work with the **Twelf** tool [PfenningSchürmann99]

Type Checking of THF0 in Twelf

THF0	LF
types	terms of type `$tType`
terms of type `A`	terms of type `$tm A`
formulas	terms of type `$tm $o`

Base signature for THF0 in Twelf

```
$tType : type.                 ($tType introduced as LF type)
$tm    : $tType -> type.       (LF type $tm A, holds terms of type A)
$i     : $tType.
$o     : $tType.
>      : $tType -> $tType -> $tType.
```

Type Checking of THF0 in Twelf

```
^       : ($tm A -> $tm B) -> $tm (A > B).
@       : $tm(A > B) -> $tm A -> $tm B.
$true   : $tm $o.
$false  : $tm $o.
~       : $tm $o -> $tm $o.
&       : $tm $o -> $tm $o -> $tm $o.
==      : $tm A -> $tm A -> $tm $o.
!=      : $tm A -> $tm A -> $tm $o.
!       : ($tm A -> $tm $o) -> $tm $o.
?       : ($tm A -> $tm $o) -> $tm $o.
!!      : ($tm(A > $o)) -> $tm $o.
??      : ($tm(A > $o)) -> $tm $o.
```

For translation of axioms and conjectures:

```
$istrue : $tm $o -> type.
```

Type Checking of THF0 in Twelf

- same ASCII notation used in Twelf (except for equality)
- THF0 binders ^, ! and ? realized via the λ-binder of Twelf
- in summary, translation to Twelf is rather straightforward

Type Checking of THF0 in Twelf

Translation of our example to Twelf

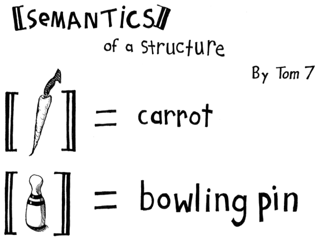
```
in           : $tm($i > ($i > $o) > $o).
intersection : $tm(($i > $o) > ($i > $o) > ($i > $o)).
union        : $tm(($i > $o) > ($i > $o) > ($i > $o)).
axiom        : $istrue in == ^[X : $tm $i] ^[S : $tm($i > $o)](S @ X).
axiom        : $istrue intersection ==
                   ^[S1: $tm($i > o)] ^[S2: $tm($i > $o)] ^[U: $tm $i]
                    ((in @ U @ S1) & (in @ U @ S2)).
axiom        : $istrue union ==
                   ^[S1: $tm($i > $o)] ^[S2: $tm($i > $o)] ^[U: $tm $i]
                    (in @ U @ S1) | (in @ U @ S2).
conjecture   : $istrue
              ![A: $tm($i > $o)] !![B: $tm($i > $o)] ![C: $tm($i > $o)]
                 ((union @ A @ (intersection @ B @ C))
                 == (intersection @ (union @ A @ B) @ (union @ A @ C))).
```

Type Checking of THF0 in Twelf

A THF0 problem is well typed if its translation to Twelf is.

Semantics of STT

- not fixed a priorily
- via annotations in problem file



Semantics of STT (options)

- Standard semantics
- Henkin semantics (without choice and description)
- Henkin semantics with choice or description
- Non-extensional Henkin semantics [Benzm.BrownKohlhase2004]
- Non-classical semantics
  - Kripke semantics for equational reasoning in the simply typed lambda-calculus (Kripke Lambda Models) [MitchellMoggi1991]
  - Semantics for full impredicative higher-order intuitionistic logic [LiptonNieva2007]
- ...

Semantics of STT

Default (assumed if no further information is provided):

- Henkin semantics (fully extensional)
- without choice and description

Infrastructure

Systems that directly read or produce THF0

- LEO-II
- Has-Casl
- "Mizar"

TPTP2X THF0 translation tools for

- Twelf
- OmDoc
- TPS

TPTP4X

- forthcoming

Infrastructure

Already online at Systems on TPTP

- LEO
- TPS

Problem Collection

We have started to collect THF0 examples:

- few (12) in TPTP library
- $>$ 150 in LEO-II library

We will provide

- small test problems
- problems stemming from applications
- challenge problems

A Small Challenge Problem

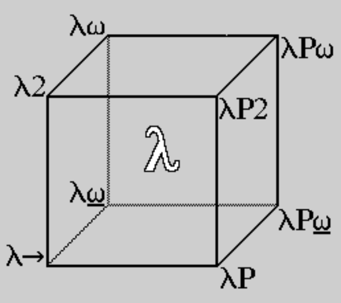
```
% Hi Chad, Christoph,
%
% I just saw something in a lecture by Martin Hofmann at Marktoberdorf,
% which (if I understand it correctly) is a fairly small fact of pure
% higher-order logic. This is the equivalence of two characterizations
% of the smallest "quasi-PER" containing a given binary relation R, one
% the obvious inductive characterization
%
%  forall a b. (forall S. (forall x y. R x y ==> S x y) /\
%                         (forall w x y z. S x y /\ S z y /\ S z w ==> S x w)
%                         ==> S a b)
%             <=>
%             (forall P Q. (forall x y. R x y ==> (P x <=> Q y))
%                           ==> (P a <=> Q b))
%
% Just wondered if there was any hope of an automated proof? Disclaimer:
% I don't completely guarantee that I got it right, and I suspect the
% proof may be non-obvious.
%
% John.
%-------------------------------------------------------------------------
```

Infrastructure: Challenge Problem

```
thf(r,constant,(r:($i>$i>$o))).

thf(thm,theorem,
 (![A:$i,B:$i]:
   ((![S:($i>$i>$o)]:
      (((![X:$i,Y:$i]: ((r @ X @ Y) => (S @ X @ Y)))
        &
        (![W:$i,X:$i,Y:$i,Z:$i]: (((S @ X @ Y) & (S @ Z @ Y) & (S @ Z @ W))
                                  => (S @ X @ W))))
       => (S @ A @ B)))
    <=>
    (![P:($i>$o),Q:($i>$o)]:
      ((![X:$i,Y:$i]: ((r @ X @ Y) => ((P @ X) <=> (Q @ Y))))
       =>
       ((P @ A) <=> (Q @ B))))))).
```

Extensions: The THF Language (ongoing)



- We want to subsequently cover logics with richer type systems
- Want to represent and check proofs

Extensions: THF Language (ongoing)

```
thf(list,type, ( list: ( $tType > $tType ) ) ).

thf(nil,type, ( nil: ( !> [ A: $tType ] : ( list @ A ) )).
thf(cons,type, ( cons: ( !> [ A: $tType ] :
                         ( A > ( list @ A) > ( list @ A) ) ) )).

thf(conj,conjecture,(
     ( ( cons @ $o ) @ $true @ ( nil @ $o ) )
       !=
       ( nil @ $o ) )).
```

How can you support us?

- Send us problems
- Adapt your systems
- Proof assistants: you may want to provide a special command like
  - `convert-subproblem-to-THF-and-email-to-Geoff`

Outlook

- want strong progress of higher-order TPTP infrastructure within next 12 months (EU project THFTPTP)
- higher-order CASC at next CADE
- want problems from
  - existing libraries of higher-order theorem provers
  - verification projects such as Verisoft
  - ontologies such as SUMO
  - challenge problems
  - re-representations of existing FO problems in TPTP
  - etc.
- first TPTP THF0 release: January 2009 (TPTP v4.0.0)

Example 1 – SET171

```
%------------------------------------------------------------------------
%----Signatures for basic set theory predicates and functions.
thf(const_in,type,(
    in: $i > ( $i > $o ) > $o  )).

thf(const_intersection,type,(
    intersection: ( $i > $o ) > ( $i > $o ) > ( $i > $o ) )).

thf(const_union,type,(
    union: ( $i > $o ) > ( $i > $o ) > ( $i > $o ) )).
```

Example – SET171

```
%----Some axioms for basic set theory. These axioms define the set
%----operators as lambda-terms. The general idea is that sets are
%----represented by their characteristic functions.
thf(ax_in,axiom,(
    ( in
    = ( ^ [X: $i,S: ( $i > $o )] :
          ( S @ X ) ) ) )).

thf(ax_intersection,axiom,(
    ( intersection
    = ( ^ [S1: ( $i > $o ),S2: ( $i > $o ),U: $i] :
          ( ( in @ U @ S1 )
          & ( in @ U @ S2 ) ) ) ) )).

thf(ax_union,axiom,(
    ( union
    = ( ^ [S1: ( $i > $o ),S2: ( $i > $o ),U: $i] :
          ( ( in @ U @ S1 )
          | ( in @ U @ S2 ) ) ) ) )).

%----The distributivity of union over intersection.
thf(thm_distr,conjecture,(
    ! [A: ( $i > $o ),B: ( $i > $o ),C: ( $i > $o )] :
      ( ( union @ A @ ( intersection @ B @ C ) )
      = ( intersection @ ( union @ A @ B ) @ ( union @ A @ C ) ) ) )).
%------------------------------------------------------------------------
```

Example – Cantor's Theorem

```
thf(surjectiveCantorThm,conjecture,(
   ~ ( ? [G: $i > $i > $o] :
        ! [F: $i > $o] :
        ? [X: $i] :
          ( ( G @ X )
          = F ) ) )).
```

Example – Knights and Knaves (chris original)

```
%------------------------------------------------------------------
%----Type declarations
thf(islander,type,(
    islander: $i )).

thf(knight,type,(
    knight: $i )).

thf(knave,type,(
    knave: $i )).

thf(says,type,(
    says: $i > $o > $o )).

thf(zoey,type,(
    zoey: $i )).

thf(mel,type,(
    mel: $i )).

thf(is_a,type,(
    is_a: $i > $i > $o )).
```

Example – Knights and Knaves (chris original)

```
%----A very special island is inhabited only by knights and knaves.
thf(kk_6_1,axiom,(
    ! [X: $i] :
      ( ( is_a @ X @ islander )
     => ( ( is_a @ X @ knight )
        | ( is_a @ X @ knave ) ) ) )).

%----Knights always tell the truth.
thf(kk_6_2,axiom,(
    ! [X: $i] :
      ( ( is_a @ X @ knight )
     => ( ! [A: $o] :
            ( says @ X @ A )
       => A ) ) )).

%-----Knaves always lie.
thf(kk_6_3,axiom,(
    ! [X: $i] :
      ( ( is_a @ X @ knave )
     => ( ! [A: $o] : ( says @ X @ A )
       => ~ A ) ) )).
```

Example – HW Verification

```
thf(hw_not,type,(hw_not:($o>$o>$o))).
thf(hw_and,type,(hw_and:($o>$o>$o>$o))).
thf(hw_or,type,(hw_or:($o>$o>$o>$o))).
thf(hw_nand_spec,type,(hw_nand_spec:($o>$o>$o>$o))).
thf(hw_nand_imp,type,(hw_nand_imp:($o>$o>$o>$o))).

thf(hw_not,axiom,(hw_not =
    (^[I:$o,O:$o]: (O = ~ I)))).
thf(hw_and,axiom,(hw_and =
    (^[I1:$o,I2:$o,O:$o]: (O = (I1 & I2))))).
thf(hw_or,axiom,(hw_or :=
    (^[I1:$o,I2:$o,O:$o]: (O = (I1 | I2))))).

thf(hw_nand_spec,axiom,(hw_nand_spec =
     (^[I1:$o,I2:$o,O:$o]: (O = ~ (I1 & I2))))).
thf(hw_nand_imp,axiom,(hw_nand_imp =
    (^[I1:$o,I2:$o,O:$o]: (?[H:$o]:
       (hw_and @ I1 @ I2 @ H) & (hw_not @ H @ O))))).

thf(imp_fulfils_spec,conjecture,
    (![I1:$o,I2:$o,O:$o]:
      (hw_nand_imp @ I1 @ I2 @ O) => (hw_nand_spec @ I1 @ I2 @ O))).
```

Example – HW Verification (definitions)

```
thf(hw_not,definition,(hw_not :=
    (^[I:$o,O:$o]: (O = ~ I)))).
thf(hw_and,definition,(hw_and :=
    (^[I1:$o,I2:$o,O:$o]: (O = (I1 & I2))))).
thf(hw_or,definition,(hw_or :=
    (^[I1:$o,I2:$o,O:$o]: (O = (I1 | I2))))).

thf(hw_nand_spec,definition,(hw_nand_spec :=
     (^[I1:$o,I2:$o,O:$o]: (O = ~ (I1 & I2))))).
thf(hw_nand_imp,definition,(hw_nand_imp :=
    (^[I1:$o,I2:$o,O:$o]: (?[H:$o]:
       (hw_and @ I1 @ I2 @ H) & (hw_not @ H @ O))))).

thf(imp_fulfils_spec,conjecture,
    (![I1:$o,I2:$o,O:$o]:
      (hw_nand_imp @ I1 @ I2 @ O) => (hw_nand_spec @ I1 @ I2 @ O))).
```

Example – HW Verification (definitions + timings)

```
thf(hw_not,definition,(hw_not :=
    (^[I:($i>$o),O:($i>$o)]: (![T:$i]: ((O @ T) = ~ (I @ T)))))).
thf(hw_and,definition,(hw_and :=
    (^[I1:($i>$o),I2:($i>$o),O:($i>$o)]: (![T:$i]:
       ((O @ T) = ((I1 @ T) & (I2 @  T))))))).
thf(hw_or,definition,(hw_or :=
    (^[I1:($i>$o),I2:($i>$o),O:($i>$o)]: (![T:$i]:
       ((O @ T) = ((I1 @ T) | (I2 @ T))))))).

thf(hw_nand_spec,definition,(hw_nand_spec :=
    (^[I1:($i>$o),I2:($i>$o),O:($i>$o)]:
        (![T:$i]: ((O @ T) = ~ ((I1 @ T) & (I2 @ T)))))))).
thf(hw_nand_imp,definition,(hw_nand_imp :=
    (^[I1:($i>$o),I2:($i>$o),O:($i>$o)]: (?[H:($i>$o)]:
       (hw_and @ I1 @ I2 @ H) & (hw_not @ H @ O))))).

thf(imp_fulfils_spec,conjecture,
    (![I1:($i>$o),I2:($i>$o),O:($i>$o)]:
      (hw_nand_imp @ I1 @ I2 @ O) => (hw_nand_spec @ I1 @ I2 @ O))).
```

Example – Knights and Knaves

A very special island is inhabited only by knights and knaves.
Knights always tell the truth, and knaves always lie.
You meet two inhabitants: Zoey and Mel.
Zoey tells you that Mel is a knave.
Mel says, 'Neither Zoey nor I are knaves.'
Can you determine who is a knight and who is a knave?

Example – Knights and Knaves (variant lucas)

```
%------------------------------------------------------------------
%----A very special island is inhabited only by knights and knaves.
thf(kk_6_1,axiom,(
   ! [X: $i] :
    ( ( is_a @ X @ knight )
      <~> ( is_a @ X @ knave ) ) )).

%----Knights always tell the truth.
thf(kk_6_2,axiom,(
    ! [X: $i] :
      ( ( is_a @ X @ knight )
     => ( ! [A: $o] :
            ( says @ X @ A )
       => A ) ) )).

%-----Knaves always lie.
thf(kk_6_3,axiom,(
    ! [X: $i] :
      ( ( is_a @ X @ knave )
     => ( ! [A: $o] : ( says @ X @ A )
       => ~ A ) ) )).
```

Example – Knights and Knaves (variant lucas)

```
%----You meet two inhabitants: Zoey and Mel.
% thf(kk_6_4,axiom,
%     ( ( is_a @ zoey @ islander )
%     & ( is_a @ mel @ islander ) )).

%----Zoey tells you that Mel is a knave.
thf(kk_6_5,axiom,
    ( says @ zoey @ ( is_a @ mel @ knave ) )).

%----Mel says, `Neither Zoey nor I are knaves.'
thf(kk_6_6,axiom,
    ( says @ mel
    @ ~ ( ( is_a @ zoey @ knave )
        | ( is_a @ mel @ knave ) ) )).

%----Can you determine who is a knight and who is a knave?
thf(query,theorem,(
    ? [Y: $i,Z: $i] :
      ( ( is_a @ mel @ Y )
      & ( is_a @ zoey @ Z ) ) )).
%------------------------------------------------------------------
```

Example – Knights and Knaves (variant chris)

```
%------------------------------------------------------------------
%----A very special island is inhabited only by knights and knaves.
thf(kk_6_1,axiom,(
    ! [X: $i] :
      ( ( is_a @ X @ islander )
     => ( ( is_a @ X @ knight )
        | ( is_a @ X @ knave ) ) ) )).

%----Knights always tell the truth.
thf(kk_6_2,axiom,(
    ! [X: $i] :
      ( ( is_a @ X @ knight )
     => ( ! [A: $o] :
            ( says @ X @ A )
       => A ) ) )).

%-----Knaves always lie.
thf(kk_6_3,axiom,(
    ! [X: $i] :
      ( ( is_a @ X @ knave )
     => ( ! [A: $o] : ( says @ X @ A )
       => ~ A ) ) )).
```

Example – Knights and Knaves (variant chris)

```
%----You meet two inhabitants: Zoey and Mel.
thf(kk_6_4,axiom,
    ( ( is_a @ zoey @ islander )
    & ( is_a @ mel @ islander ) )).

%----Zoey tells you that Mel is a knave.
thf(kk_6_5,axiom,
    ( says @ zoey @ ( is_a @ mel @ knave ) )).

%----Mel says, 'Neither Zoey nor I are knaves.'
thf(kk_6_6,axiom,
    ( says @ mel
    @ ~ ( ( is_a @ zoey @ knave )
        | ( is_a @ mel @ knave ) ) )).

%----Can you determine who is a knight and who is a knave?
thf(query,theorem,(
    ? [Y: $i,Z: $i] :
      ( ( is_a @ Y @ knight )
      & ( is_a @ Z @ knave) ) )).
%------------------------------------------------------------------
```