CHECKER

written by Andrzej Trybulec
compiled by Freek Wiedijk

Abstract

This is a compilation of a series of e-mail messages by Andrzej Trybulec. The basic inference step (`by`) of the Mizar system is described.

1

Let us start with two things:

We have to check an inference[1] of the form

$$\frac{b_1 \quad \cdots \quad b_k}{a}$$

(b_i are referenced premises, maybe by '`then`'. `Then`/`hence` is only syntactic sugar.)

The CHECKER is a disprover, i.e. it negates the conclusion, puts it among premises and tries to get contradiction. So it has to check:

$$\frac{b_1 \quad \cdots \quad b_k \quad \neg a}{\bot}$$

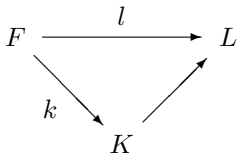
The second point that is important: Mizar works with a system of semantic correlates. The concept of semantic correclates had been introduced by Roman Suszko in his works on non-Fregean logic. He does it syntactically, but on the semantic levels one can present them in the following way:

Let F be algebra of formulae, let it be a universal free algebra with usual set of connectives: $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\neg$. I would like to have also nullary connectives: $\top$ and $\bot$ (or in the Mizar notation maybe '`contradiction`' for $\bot$, we have no notation for $\top$). And we need quantifiers: $\forall$ and $\exists$, I would prefer to think about them as families of operations, two operations for any variable, one of course must explain what a variable is, let us postpone this.

[1] 'inference' is basically an uncountable word in English, but when I used 'an inference' nobody protested, so I think we may use it.

Let L be corresponding Lindebaum algebra (it should be called Tarski-Lindenbaum algebra, but usually it is called just Lindenbaum), i.e. the algebra of logically equivalent classes of formulae, and let $l : F \to L$ be canonical mapping.

Then by an algebra of semantic correlates we mean a factorization of the mapping l



Of course, as special cases we get as algebras of semantic correlates F and L (with k equal to the identity and l resp.) Now, for practical application we want k be a computable function, with low complexity.

In Mizar, as yet, the system of semantic correlates is close to abstract syntax, but:

All propositional connectives but $\wedge$, $\neg$ and $\top$, are eliminated using well known rules (de Morgan laws etc.), the only explanation needed: the equivalence $a \leftrightarrow b$ is equal to

$$(a \to b) \wedge (b \to a)$$

i.e.

$$\neg(a \wedge \neg b) \wedge \neg(b \wedge \neg a)$$

The other possibility:

$$(a \wedge b) \vee (\neg a \wedge \neg b)$$

i.e.

$$\neg(\neg(a \wedge b) \wedge \neg(\neg a \wedge \neg b))$$

seems to be less useful.

There are additional laws.

1. The conjunction is associative (but not commutative).
2. $\top$ is the unity of conjunction.
3. The negation is an involution: $\neg\neg a = a$.

We just started to experiment with the distributivity of the universal quantifier with respect to conjunction.

I think that it is enough for the beginning, just observe that some of logical laws are obvious, because we work with semantic correlates.

Any way, because CHECKER is a disprover, and because of the semantic correlates used, the inference

$$\frac{\forall x.\, a(x)}{a(c)}$$

is basically the same as

$$\frac{a(c)}{\exists x.\, a(x)}$$

* * *

> *It is not very clear to me why it makes a difference that conjunction is not commutative (does it make a difference for what* CHECKER *accepts?)*

Yes, the inference

$$\frac{\forall x \exists y.\, P[x,y] \wedge Q[x,y]}{\exists y.\, P[x_0,y] \wedge Q[x_0,y]}$$

is accepted

$$\frac{\forall x \exists y.\, P[x,y] \wedge Q[x,y]}{\exists y.\, Q[x_0,y] \wedge P[x_0,y]}$$

is not. Some people, at least one (Pauline N. Kawamoto), believe that it is serious shortcoming.

> *Apparently, there is some relation between $\forall$ and $\exists$ on the level of semantic correlates as well: apparently $\forall$ and $\neg\exists\neg$ in your F map to the same element of your K. So do you eliminate all the $\exists$'s as well, like the disjunctions?*

Yes, I forgot to write about it.

> *Another question: do you eliminate propositional connectives only on the outer level (so outside the quantifiers) or inside the quantifiers as well?*

All of them, in the scope of a quantifier, too.

2

The next step is:

Given

$$\frac{p_1 \quad \cdots \quad p_k}{\bot}$$

CHECKER conjugate all premises (p_i – an arbitrary formula) and creates a disjunctive normal form of the conjunction. Two remarks:

1. It is a normal form created from atomic (all kinds of them) and universal sentences; because we needed a more general concept of normal form that is usually used it prompted me to define it in Mizar (`NORMFORM`) and I had proved that it is a Heyting algebra (`HEYTING1`); recently we develop theory of such lattices – we may claim that some part of CHECKER is already described in Mizar `:-)`

2. Among all possible normal forms, CHECKER chooses what we call the standard normal form – just the normal form that we get using so many times distributivity of disjuction with respect to conjunction as necessary. We experimented earlier with the canonical normal form (all disjuncts of the same length) it gives a bit stronger concept of obviousness, and the regular normal form (it is element of a Heyting algebra), but it complicates things and the gain is comparatively small.

Now we get something like

$$\frac{(q_{1,1} \wedge \ldots \wedge q_{1,k_1}) \vee \ldots \vee (q_{n,1} \wedge \ldots \wedge q_{n,k_n})}{\bot}$$

and of course CHECKER tries to refute the consecutive disjuncts:

$$\frac{q_{1,1} \wedge \ldots \wedge q_{1,k_1}}{\bot} \quad \ldots \quad \frac{q_{n,1} \wedge \ldots \wedge q_{n,k_n}}{\bot}$$

In this way we get rid of the propositional calculus, if the original inference is obvious on the propositional level, the disjunctive normal form is empty, nothing to refute and the inference is accepted.

Of course we got more: different cases are separated now. They are checked separately and errors are reported independently: one sometimes get many

```
*4
```

or rather

```
*4,4,4,4,4,4,4
```

It is an error, we should report e.g. which disjuncts are not refuted, but it is not easy to compute the index of the disjunct, and we need a `#.LOG` file anyway.

Of course in most cases we have just one disjunct.

* * *

> *You say this like it doesn't matter for the result in which order you apply the distributivity laws. That probably is the case then, although it isn't completely obvious to me.*

It is a consequence of the associativity. However we do more: the absorption law is used to remove too long disjuncts. And because in fact disjunctive normal form (in CHECKER) is rather a set (ordered collection, then not list) of disjuncts, that are boolean valued partial functions, then on this level conjunction is commutative.

> *We experimented earlier with the canonical normal form (all disjuncts of the same length)*
>
> *You mean with all atoms in all the disjuncts the same, but depending on which disjunct it is, with or without a negation? Some subset of the 2^n disjuncts?*

Yes, perfect.

3

CHECKER in fact consists of three passes, as yet we talked about the first pass that we call the PRECHECKER, the next is EQUALIZER (because the main role of it is to cope with the equality calculus), the last we call UNIFIER, that is a bad name, because we do not allow for substitution of a variable to a variable. We have no better name. Let us move to UNIFIER.

In a disjunct CHECKER chooses a universal sentence (if any, if there is no universal sentence, the inference is not accepted), which we call a 'main premise', in it removes external universal quantifiers, and change bound variables to free.

It is convenient to think about resulting formula as a $\wedge/\vee$ tree, because de Morgan laws hold for semantic correlates, we may push down negations to leaves.

The remaining premises are called auxiliary premises.

In the main premise we have on leaves (possibly negated) basic formulae (atomic or universal) with free variables. For every leaf CHECKER looks for auxiliary premises with different sign (i.e. not negated if the formula on the leaf is negated and non negated if it is) and the same structure, i.e. such auxiliary premises that differs only on places in which in the formula on the leaf a free variable occurs. (I hope I got the syntax correct `:-)`)

So on every leaf we get a list of substitutions (partial maps from free variables to ground terms) after which the formula of the leaf is contradictory with an auxiliary premise.

Then we use an algebra of substitutions: if $\sigma(f)$ is the list of substitutions of the formula f, then

$$\begin{aligned}\sigma(f \wedge g) &= f \cup g \\ \sigma(f \vee g) &= \{s_1 \cup s_2 : s_1 \in \sigma(f),\ s_2 \in \sigma(g) \text{ and } s_1 \cup s_2 \text{ is a function}\}\end{aligned}$$

The last condition means that it does not try to substitute two different ground terms for the same variable.

Similarly as in the case of the propositional calculus list of substitutions are standarized, bigger substitutions are removed. I believe I forgot to write that when we create disjunctive normal form we remove contradictory disjuncts, that corresponds to the condition that $f \cup g$ is a function.

Then if $\sigma(m)$, where m is the main formula, is non empty, the set of premises is contradictory and the inference is accepted.

The lattice of list of substitutions is described in `SUBSTLAT` and the proof that it is a Heying algebra in `HEYTING2`. In fact the lattices of normal forms are special cases (up to isomorphism) of lattices of substitutions.

* * *

> *Something I wondered about: is* CHECKER *a big part of Mizar's source code?*

The unit CHECKER is 15% of the code (about 150 kBytes), but some procedures needed in checker, or shared with other units are in other units.

> *So I guess* UNIFIER *just tries all universal sentences, one by one? Is there a heuristic which one to start with, or does* UNIFIER *just try them in order from left to right? (I understand that this doesn't matter for the semantics of '*`by`*', but I wondered about this.)*

Usually there is one universal premise, but if there is more the order is accidental.

4

Going back to the CHECKER. Maybe you observed that the rules for main premise are not precise. I wrote, purposefully, 'external universal quantifiers' but the phrase has at least two meanings:

- the universal quantifiers on the beginning of the proposition
- the universal quantifiers that are not in the scope of an existential quantifer (here we should presume that negations are pushed to leaves, but this time leaves are nodes labelled by atomic sentences rather and not universal)

In some old CHECKERs we used the first approach, now the second is used. But then (if we do nothing else) the inference

$$\frac{\forall xy.\, P[x,y]}{\forall y.\, P[a,y]}$$

is not accepted. It is in fact

$$\frac{\forall xy.\, P[x,y] \quad \exists y.\, \neg P[a,y]}{\bot}$$

Because of that we have an additional rule: PRECHECKER extracts existential sentences from disjuncts in DNF and does existential elimination (if I recall the name), we rather say it uses the choice rule automatically. And after introducing new constants, creates again DNF.

Then the inference above is accepted.

The second problem, CHECKER standarize premises, i.e. it uses the distributivity of universal quantifier with respect to conjunction (and what's the same distributivity of existential quantifier with respect to disjunction). It is a first step to change semantic correlates in Mizar (I believe I already wrote about that). Of course in implementation we use sort of normalized form, and it means that quantifiers are pushed down. I mean that instead of

$$\forall x.\, P[x] \wedge Q[x]$$

we have in fact

$$(\forall x.\, P[x]) \wedge (\forall x.\, Q[x])$$

After introducing this (it is comparatively new) we revised MML, but the gain was very small. I do not remember the numbers, but it was negligable. However some steps like

$$\frac{\forall x.\, P[x] \leftrightarrow Q[x]}{\forall x.\, P[x] \rightarrow Q[x]}$$

had been eliminated. It caused also the need to eliminate fictitious variables. This I do not like, but because of

$$\forall xy.\, P[x] \wedge Q[y]$$

it was necessary.

* * *

This strengthening of the CHECKER caused some funny phenomena. Somebody, I believe Markus Wenzel (the author of Isar), asked about the use of Mizar for 'pure logic', I prepared some axiomatic files for it and wanted to prove

$$((\forall x.\, \mathit{Works}[x]) \rightarrow \mathit{WFS}) \rightarrow \exists x.\, (\mathit{Works}[x] \rightarrow \mathit{WFS})$$

(the last pair of parentheses is not necessary, I put them to avoid misunderstanding). Before proving it (the simplest prove is just to separate cases: either everybody works, then from the assumption we have Well Fare State or somebody does not work, then he is the bastard for which it is true that if he works we

have Well Fare State. Students usually propose the prime minister as the candidate, but it is wrong – the tautology is not intuitionistic, and the constructive proof is impossible `:-)`.

I am lazy so first I put semicolon after the sentence, and I observed with some horror that it had been accepted. What happens: after distributing quantifers CHECKER got

$$\frac{\neg(((\forall x.\, Works[x]) \to WFS) \to \exists x.\, (Works[x] \to WFS))}{\bot}$$

i.e.

$$\frac{(\neg\forall x.\, Works[x]) \vee WFS \quad \neg\exists x.\, (\neg Works[x] \vee WFS)}{\bot}$$

i.e.

$$\frac{\neg Works[x_0] \vee WFS \quad \neg(\exists x.\, \neg Works[x]) \vee \neg WFS}{\bot}$$

Sorry, I was convinced that we have to distribute quantifers to get it accepted. Now writing down this example I am not certain. What do you think?

* * *

> *I understand what* PRECHECKER *does (it brings things in some canonical form, and leads to a set of conjunctions from each of which falsity should be derived.)*

It is OK. However, it does a bit more. E.g. uses the choice rule by default, i.e. it introduces local constants, if an existential sentence is among premises.

> *I also think I understand* UNIFIER*: it skolemizes (= introduces new function symbols for existentials) and then takes one universally quantified formula from the conjunction which it tries to instantiate to get a contradiction.*

No, we do not use skolemization, so processing basically 'stops' on existential quantifiers. To be more precise: an existential sentence is treated as a whole and only contradiction between it (after substantiation) we a universal sentence is checked.

> *So that leaves* EQUALIZER*. That pass does 'congruence closure' on the equalities in the conjunction. But how does it interact with* UNIFIER*? And what happens to equalities under the universal quantifier that's being instantiated (are those added to the congruence too?)*

It is true. EQUALIZER introduce what we call 'aggregated constants' or 'equating classes' (we should work on the terminology) and uses them for instantiation. An aggregated constant is characterized by the list of its possible forms and their types (all expressed using aggregated constants).

> *But how thus this interfere with the 'calculus of substitutions' that you told me about?*

Just these aggregated constants are substituted.

Actually it does not add the equalities (from universal sentences), but it does a forward reasoning, it adds inequalities that are consequences of premises, e.g., if among premises occurs

$$(a + b) + c \neq (b + a) + c$$

it adds

$$a + b \neq b + a$$

To avoid calling prechecker again, we have 'one difference rule', that says that the inequality is added, if it is the only inequality between two unequal terms (or the only difference between two atomic sentence with opposite valuation).

> *Now I don't understand any more. If you add* in*equalities, then you can't represent them as list of possible forms? Or can you?*

Now I do not understand. Maybe ...

Sentences in Mizar are not terms, we have three classes of (small constructions): formulae, terms and types, and they are processed in a different way. We just keep a list of sentences, we may add their consequences to the list.

The EQUALIZER (maybe it is its main role) is also a gate through which we smuggle requirements and similar 'additional but practical' rules.

> *I suppose you mean things like 'symmetry' and so on?*

Yes.

* * *

> *So terms and types are represented by aggregated constants, and formulae are not? (I mean: equivalent formulae are not represented by one 'aggregated formula'?)*

Only terms. There is not a concept of equality of type (well, you may say that the types θ_1 and θ_2 are equal if the sentence

$$\forall x : set.\ (x : \theta_1 \leftrightarrow x : \theta_2)$$

it does not seem useful) The equivalence calculus was discussed, but we think that the gain of building it in is minimal.

> *So what happens to formulae?*

All terms in formulae and type (and of course term `:-)`) are substituted by its aggregate constants (basically by a constant I mean here a ground term).

> *Also: how are the clusters integrated into this? Are the type aggregates extended automatically by the clusters?*

Basically clusters in types are rounded up in ANALYZER, but EQUALIZER tries to round up them again, using the fact that an aggregated constant has many types (inherited from the members)

5

What I never wrote as yet:

We keep formulae on two lists: P – positively valued and N – negatively valued. On this list we keep (all kinds of) atomic formulae and universal formulae.

But:

- equalities are removed from the P list; negatively valued[2] are still on the N list
- negatively valued universal sentences (i.e. existential sentence) are removed from the N list (they had been used by PRECHECKER the choice rule).

After creating aggregated constants EQUALIZER checks if it can get a contradiction:

1. maybe two differently valued formulae become equal
2. maybe we got $E \neq E$

[2] i.e. '*un*equalities'? inequalities?, the same problem in Polish, inequality means something like '$1 < k$'

ad 1

$$\frac{P[a] \quad a = b}{P[b]}$$

after aggregation we got $E = \{a, b\}$

the P list	$P[E]$	
the N list	$P[E]$	(actually $\neg P[E]$)

so the inference is accepted

ad 2 let us check

$$(a = b \vee c = d) \wedge (a = c \vee b = d) \rightarrow a = d \vee b = c$$

so we have to refute

$$(a = b \vee c = d) \wedge (a = c \vee b = d) \wedge \neg(a = d \vee b = c)$$

DNF is

$$\begin{array}{l} a = b \wedge a = c \wedge a \neq d \wedge b \neq c \vee \\ a = b \wedge b = d \wedge a \neq d \wedge b \neq c \vee \\ c = d \wedge a = c \wedge a \neq d \wedge b \neq c \vee \\ c = d \wedge b = d \wedge a \neq d \wedge b \neq c \end{array}$$

after aggregation:

$$\begin{array}{l} E_1 = E_1 \wedge E_1 = E_1 \wedge E_1 \neq E_2 \wedge E_1 \neq E_1, \\ E_1 = E_1 \wedge E_1 = E_1 \wedge E_1 \neq E_1 \wedge E_1 \neq E_2, \\ E_1 = E_1 \wedge E_1 = E_1 \wedge E_1 \neq E_1 \wedge E_2 \neq E_1, \\ E_1 = E_1 \wedge E_1 = E_1 \wedge E_2 \neq E_1 \wedge E_1 \neq E_1 \end{array}$$

so we got contradiction, and the inference is accepted.

* * *

> *The commutativity keyword is taken into account when doing the aggregation? Are there other properties like that that affect what is aggregated?*

Actually it is before aggregation. The first step of EQUALIZER (or maybe the zero step) it collecting terms. It is like initialization of the aggregation. We want terms to be kept in one copy, so it browses the formulae and types, and terms of course to prepare a list of terms that occur in the inference. The terms themselves are substituted by (trivial) aggregated constants. But we must remember their structure, so the object representing a collected term has a field that is the list of possible forms of it.

Collecting a term

$$(a + b) + c = a + (b + c)$$

we get

$$\begin{array}{ll} E_1 = a & EqList = \{a\} \\ E_2 = b & EqList = \{b\} \\ E_3 = a + b & EqList = \{E_1 + E_2, E_2 + E_1\} \\ E_4 = c & EqList = \{c\} \\ E_5 = (a + b) + c & EqList = \{E_3 + E_4, E_4 + E_3\} \\ E_6 = b + c & EqList = \{E_2 + E_4, E_4 + E_2\} \\ E_7 = a + (b + c) & EqList = \{E_1 + E_6, E_6 + E_1\} \end{array}$$

It is a trick, it hardly can be used for the associativity, and we are in the great need to have it.

Also we want to have

- involutiveness, like

$$-(-x) = x$$

- projectivity, like

$$\mathrm{abs}(\mathrm{abs}(x)) = \mathrm{abs}(x)$$

but when we started to implement, we tried to do it as part of the aggregation (the trick with the commutativity cannot be used, because equating $-(-E)$ to E must to deal with the situation:

$$\begin{array}{l} a = -b \\ b = -c \end{array}$$

and to equate a and c we need the aggregation.) Well, we tried but we failed, the data structures should be changed to do it.

What affect aggregation:

1. explicit equalities
2. 'default equalities' ('`take`' and '`reconsider`' introduce new constant that is equal by default to term that is taken or reconsidered)
3. processing structures (if two structures are equal corresponding fields are equal)
4. requirements; it is quite irregular, but typical is equating $x - 0$ and x (requirement `REAL`); not so typical is equating a set that is empty with the empty set. e.g. $\emptyset\ A$ (`SUBSET_1:def 3`) with $\emptyset$ (`BOOLE:def 1`)

So formulae are added to these lists by (1) 'propagating inequalities', and (2) by the properties of the predicates (symmetry, antonym, etc.), and (3) by the requirements?

Only propagating inequalities; in other cases the formula is if constructed is virtual, usually it is not.

> *For a specific example: if I have a step* $3 + 4 = 7$ *in my proof, how does* EQUALIZER *prove that?*

In the object representing a collected term there are two fields (I do not remember the names)

- a boolean field 'term has an integer value' initialized of course as false, but for numerals
- an integer field 'the integer value of it' (undefined if it has not an integer value and initialized to the value for (small) integers ('small' means '`longint`' we doubt if we should use '`int64`'))

The terms having the same integer value are equated. Then to refute

$$3 + 4 \neq 7$$

it is to refute

$$E_k \neq E_k$$

E_k being the aggregated constant of $\{3 + 4, 4 + 3, 7\}$

> *Basically clusters in types are rounded up in* ANALYZER*, but* EQUALIZER *tries to round up them again*
>
> *Do both passes use the same 'rounding up code' in the implementation? Just curious.*

I wrote 'tries' because there are shortcomings in the implementation.

I should rather write PRECHECKER rounds up the type of ground terms introduced by the Choice Rule. Even this seems not to work properly, there are rare cases that

```
...
proof
  thus ... by ++++;
end;
```

cannot be changed to

```
... by ++++;
```

and I suspect that the rounding up of these terms might be a problem. I hope to fix it when back in Bialystok. To be precise, I do not know if the problem is with rounding up, I just suspect.

The EQUALIZER just 'adjusts' the clusters. It means that it moves adjectives between types of the same aggregate constant.

The real rounding up is not done, and it needs other implementation. The problem is that the concept of 'type(s)' of an aggregated constant is a bit more complicated. If it is a functorial term, not only its aggregate constant has 'multiple type' (I mean that maybe more than one, this correspondence is very good for me, I have to develop the terminology) but also the arguments are aggregated classes and has multiple types. Funny that you asked about it.

* * *

> *. . . equating a set that is empty with the empty set.*
> *e.g.* $\emptyset$ *A (*`SUBSET_1:def 3`*) with* $\emptyset$ *(*`BOOLE:def 1`*)*
>
> *Which requirement gives the last equality?*

I believe `BOOLE` and `SUBSET` (it was built in, but now it should require `BOOLE` and `SUBSET`). The '`empty`' attribute is now introduced in `SUBSET_1`.

> *So how are things like symmetry, antonym, etc. handled? They operate on predicates, so I guess they don't affect the aggregation of terms, right? And if they don't add formulas to the lists, how are they implemented then?*

'`antonym`' and '`synonym`' are only syntactic sugar, ANALYZER eliminates them.

Yes, they don't affect aggregation of terms.

'`symmetry`' and so on ('`reflexivity`', '`irreflexivity`', '`connectedness`') are processed by both EQUALIZER and UNIFIER. It is done in this way that in all procedures (I hope all) that compares two formulae they take into account that maybe the order of arguments is different.