Extending ACL2 with SMT solvers

Yan Peng & Mark Greenstreet

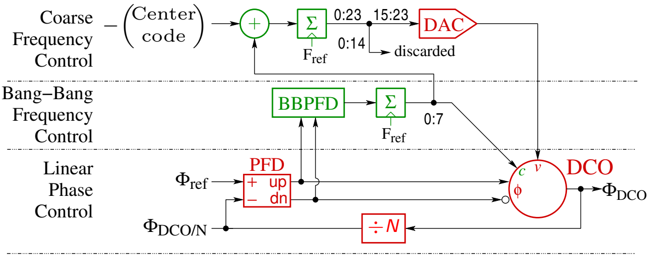
University of British Columbia

October 2nd, 2015

Smtlink handles tedious details of proofs so you can focus on the interesting parts.

Contents

The digital Phase-Locked Loop example[CNA10]



- A PLL is a feedback control system that, given an input reference clock f_{ref}, it outputs a clock at a frequency f_{DCO} that's N times of the input clock frequency and aligned with the reference in phase.
- Analog/Mixed-Signal design are composed of both analog and digital circuits.

Modelling the digital PLL

- The digital PLL is naturally modelled using non-linear recurrences that update the state variables on each rising edge of ϕ_{ref}.

$$
\begin{aligned}
c(i+1) &= next_c(c(i), v(i), \phi(i)) \\
v(i+1) &= next_v(c(i), v(i), \phi(i)) \\
\phi(i+1) &= next_\phi(c(i), v(i), \phi(i))^1
\end{aligned}
$$

[1] Three state variables: capacitance setting c (digital), supply voltage v (linear), phase correction ϕ (time-difference of digital transitions).

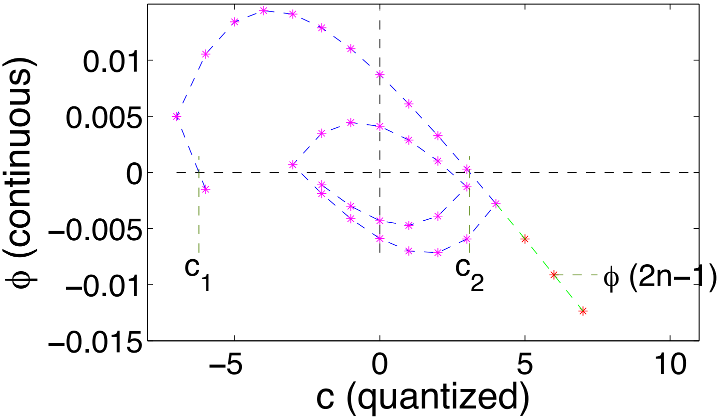
Modelling the digital PLL

- In more details,

$$
\begin{aligned}
c(i+1) &= \mathrm{saturate}(c(i) + g_c\,\mathrm{sgn}(\phi(i)), c_{\min}, c_{\max}) \\
v(i+1) &= \mathrm{saturate}(v(i) + g_v(c_{center} - c(i)), v_{\min}, v_{\max}) \\
\phi(i+1) &= \mathrm{wrap}(\phi(i) + (f_{dco}(c(i), v(i)) - f_{ref}) - g_\phi \phi(i)) \\
f_{dco}(c, v) &= \frac{1+\alpha v}{1+\beta c} f_0 \\
\mathrm{saturate}(x, lo, hi) &= \min(\max(x, lo), hi) \\
\mathrm{wrap}(\phi) &= \mathrm{wrap}(\phi + 1), \qquad \text{if } \phi \leq -1 \\
&= \phi, \qquad \text{if } -1 < \phi < 1 \\
&= \mathrm{wrap}(\phi - 1), \qquad \text{if } 1 \leq \phi
\end{aligned}
$$

- Turns out to be a relatively large system of non-linear arithmetic formulas.

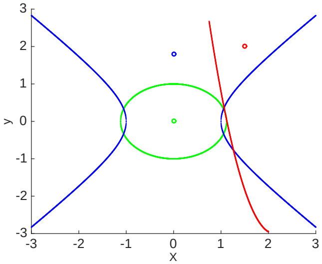
Convergence

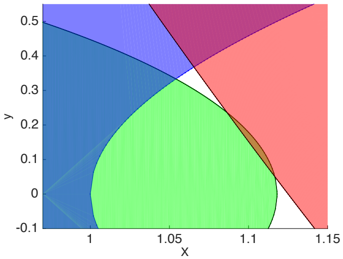


- Requires reasoning about sequences of states.
- We want to show that each crossing of $\phi = 0$ is closer to the origin than the previous one.

Example: polynomial inequalities

Do you sometimes find it frustrating to prove a theorem like this?

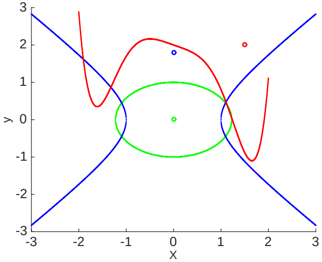


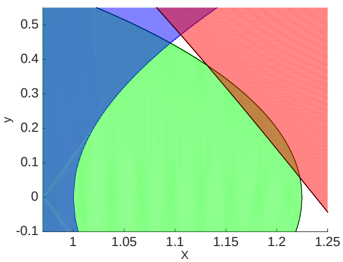



```
(defthm poly-ineq-example-a
  (implies (and (rationalp x) (rationalp y)
                (<= (+ (* 4/5 x x) (* y y)) 1)
                (<= (- (* x x) (* y y)) 1))
           (<= y (- (* 3 (- x 17/8) (- x 17/8)) 3))))
```

Example: higher order polynomial inequalities

Maybe this? With a higher order term?

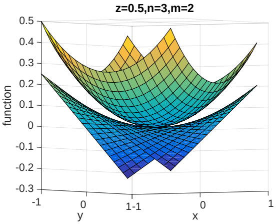


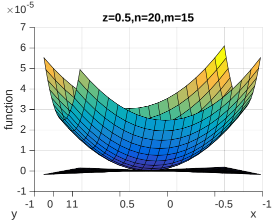



```
(defthm poly-ineq-example-b
  (implies (and (rationalp x) (rationalp y)
                (<= (+ (* 2/3 x x) (* y y)) 1)
                (<= (- (* x x) (* y y)) 1))
           (<= y (+ 2 (- (* 4/9 x)) (- (* x x x x)) (*
    1/4 x x x x x x)) )))
```

Example: exponential functions

Or even this one with exponential functions?



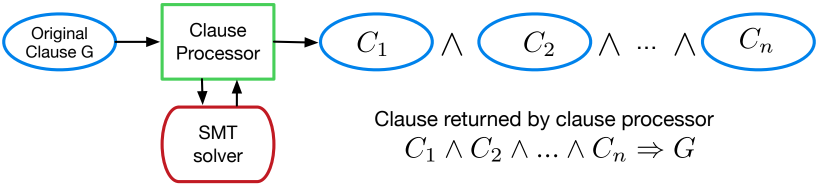



```
(defun ||x^2+y^2||^2 (x y) (+ (* x x) (* y y)))
(defthm poly-of-expt-example
  (implies (and (rationalp x) (rationalp y) (rationalp z)
                (integerp m) (integerp n)
                (< 0 z) (< z 1) (< 0 m) (< m n))
           (<= (* 2 (expt z n) x y)
               (* (expt z m) (||x^2+y^2||^2 x y) ))))
```

Motivation

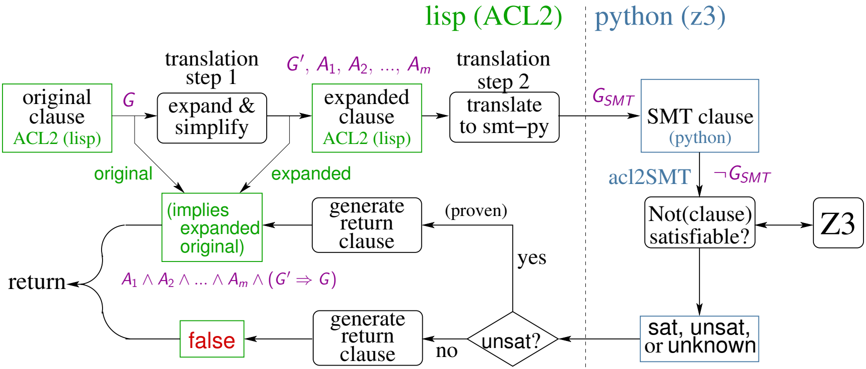
1. Motivation: provide better proof capabilities for AMS and other physical systems.
2. ACL2 provides extensive support for induction proofs and for structuring large, complicated proofs.
3. Z3 has automatic procedures for solving arithmetic formulas.
    - No direct support for induction.
    - Need to avoid "too much information" – important to give Z3 the relevant facts to keep the problems tractable.

Starting with a clause processor



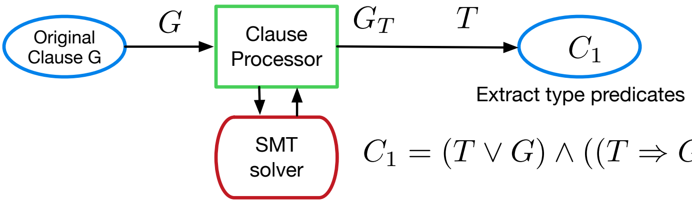
- Verified clause processor & trusted clause processor. We use a trusted clause processor for the integration.
- We utilize clauses C_1, C_2 ... C_n to get ACL2 to check many of the steps of our translation.

Two-step translation architecture



- First translation step: clause transformation
- Second translation step: transliteration

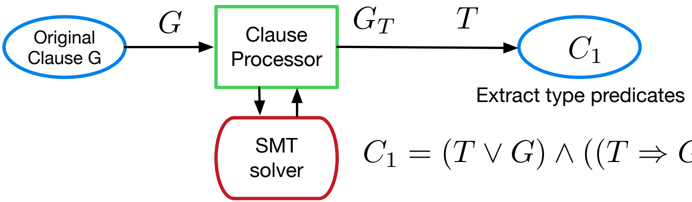
Extract type predicates



$$C_1 = (T \vee G) \wedge ((T \Rightarrow G_T) \Rightarrow G)$$

- ACL2 is not typed while Z3 is typed.
- It is common for the users to include type-recognizers in the hypotheses.
- We are currently translating `rationalp` in ACL2 into `reals` in Z3.

Extract type predicates



$$C_1 = (T \vee G) \wedge ((T \Rightarrow G_T) \Rightarrow G)$$

G

```
(implies (and (rationalp x) (rationalp y) (rationalp z)
              (integerp m) (integerp n)
              (< 0 z) (< z 1) (< 0 m) (< m n))
         (<= (* 2 (expt z n) x y)
             (* (expt z m) (||x^2+y^2||^2 x y) )))
```

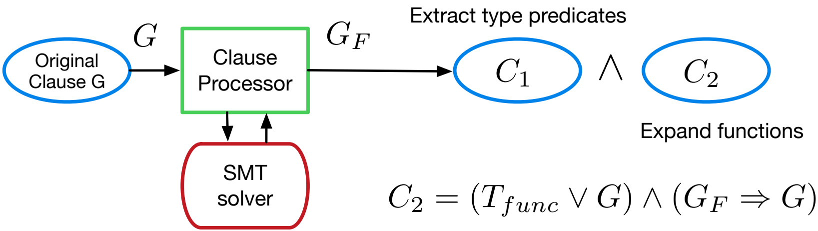
T

```
(and (rationalp x) (rationalp y) (rationalp z)
     (integerp m) (integerp n))
```

G_T

```
(implies (and (< 0 z) (< z 1) (< 0 m) (< m n))
         (<= (* 2 (expt z n) x y)
             (* (expt z m) (||x^2+y^2||^2 x y) )))
```

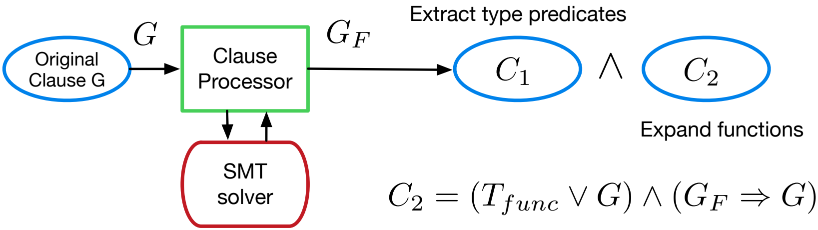
Expand functions



$$C_2 = (T_{func} \vee G) \wedge (G_F \Rightarrow G)$$

- Functions are expanded into primitive functions.
- Recursive functions are expanded to a user specified level then replaced with a variable of appropriate type.
- Uninterpreted functions stay the same.

Expand functions



$$C_2 = (T_{func} \vee G) \wedge (G_F \Rightarrow G)$$

```
(rationalp (||x^2+y^2||^2 x y))
```

T_{func}

⇑ function type clause

```
(||x^2+y^2||^2 x y)
```

⇓ function expansion

```
((lambda (VAR1 VAR2) (+ (* VAR1 VAR1) (* VAR2 VAR2))) x y)
```

Revisit the expt proof

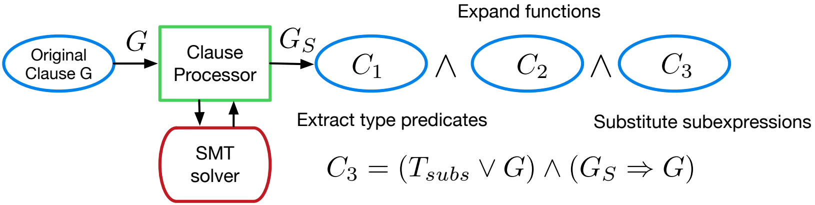
Let's take a look at the expt theorem again:

```
(defun ||x^2+y^2||^2 (x y) (+ (* x x) (* y y)))
(defthm poly-of-expt-example
  (implies (and (rationalp x) (rationalp y) (rationalp z)
                (integerp m) (integerp n)
                (< 0 z) (< z 1) (< 0 m) (< m n))
           (<= (* 2 (expt z n) x y)
               (* (expt z m) (||x^2+y^2||^2 x y) ))))
```

The reason that this is a theorem is because:

- $0 < z < 1$ and $0 < m < n \Rightarrow 0 < z^n < z^m$
- $2xy \leq x^2 + y^2$

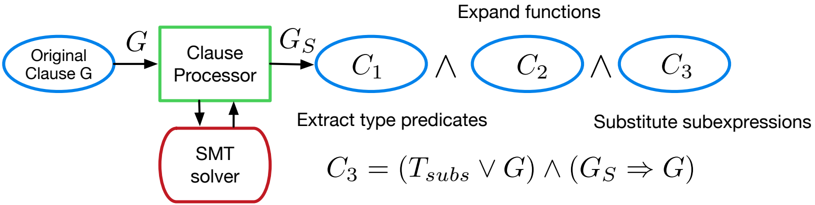
Substitute subexpressions



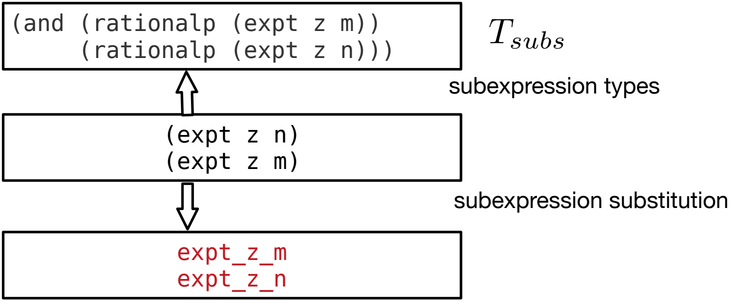
$$C_3 = (T_{subs} \vee G) \wedge (G_S \Rightarrow G)$$

- The user can substitute subexpressions with variables.

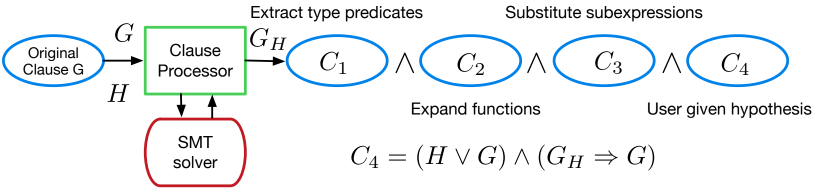
Substitute subexpressions



$$C_3 = (T_{subs} \lor G) \land (G_S \Rightarrow G)$$



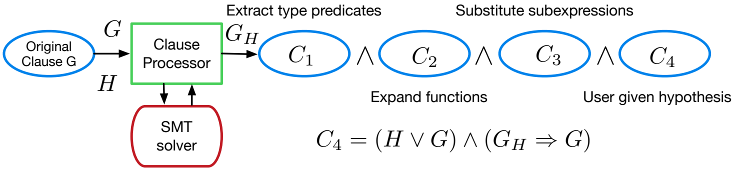
User given hypotheses



$$C_4 = (H \vee G) \wedge (G_H \Rightarrow G)$$

- The user can provide hypotheses about this theorem.
- The hypothesis feature conveys facts from the ACL2 world about these variables to the SMT solver.

User given hypotheses




```
;; given hypotheses in the theorem
((lambda (expt_z_n expt_z_m)
   (and (< expt_z_n expt_z_m) (< 0 expt_z_m) (< 0 expt_z_n)))
  (expt z n) (expt z m))
```

H

hypothesis clause

```
expt_z_m
expt_z_n
```

added hypotheses

```
(and (< expt_z_n expt_z_m) (< 0 expt_z_m) (< 0 expt_z_n))
```

The expt proof

The transformed result clause G' becomes:

```
(lambda (expt_z_m expt_z_n)
 ;; bind substitution variables to their original expressions
  (implies (and (and (< 0 z) (< z 1) (< 0 m) (< m n))
                (and (< expt_z_n expt_z_m)
                     (< 0 expt_z_m) (< 0 expt_z_n)))
      (<= (* 2 expt_z_m x y)
          (* expt_z_n
             ((lambda (VAR1 VAR2)
                      (+ (* VAR1 VAR1) (* VAR2 VAR2))) x y) )))
  (expt z m) (expt z n)))
```

The returned clauses are respectively: $T \vee G$, $T_{func} \vee G$, $T_{subs} \vee G$ and $H \vee G$.

The expt proof

The clause processor hint:

```
:hints (("Goal" :clause-processor
  (Smtlink clause
    '((:expand ((:functions ((||x^2+y^2||^2 rationalp)))
                (:expansion-levels 1)))
      (:let ((expt_z_m (expt z m) rationalp)
             (expt_z_n (expt z n) rationalp)))
      (:hypothesize ((< expt_z_n expt_z_m)
                     (< 0 expt_z_m)
                     (< 0 expt_z_n)))))))
```

Trust a little, but not too much

Let G be the original clause, A be all auxiliary clauses generated during the first translation step and G' be the main clause after this step. Let G_{SMT} be the transliteration result after the second translation step. Q_1 and Q_2 are the two sets of clauses returned to ACL2.

$$\begin{aligned} Q_1 &= (G' \wedge A) \Rightarrow G \\ Q_2 &= A \vee G \end{aligned} \tag{1}$$

Since we assume that the second translation step is sound, meaning $G_{SMT} \Rightarrow G'$, and the SMT solver proves G_{SMT}, We conclude that G is a theorem.

Customization interface

```
(local
 (progn
   (defun my-smtlink-expt-config ()
     (declare (xargs :guard t))
     (change-smtlink-config *default-smtlink-config*
       :dir-interface    ;; SMT file directory
       "../z3_interface"
       :SMT-module       ;; SMT module name
       "RewriteExpt"
       :SMT-class        ;; SMT class name
       "to_smt_w_expt"
       ))
   (defattach smt-cnf my-smtlink-expt-config)))
```

- The default Smtlink and the customizable Smtlink uses different trust tags.

Customizing Smtlink

- As an example, we created a customized Smtlink that adds a partial theory of `expt` to Z3.

`(expt x 0)` $\rightarrow$ `1`

`(expt 0 n)` $\rightarrow$ `0`, if $n > 0$

`(expt x (+ n1 n2))` $\rightarrow$ `(* (expt x n1) (expt x n2))`

`(expt x (* c n))` $\rightarrow$ `(* (expt x n) (expt x n) ...)`

`(< (expt x m) (expt x n))`, if $1 < x$ and $m < n$

...

- This simplified the use of Smtlink to produce a simpler proof. The new proof is about half the length of the original.

An example from the digital Phase-Locked Loop proof

Definitions:

$$\texttt{B-term(h)} = (1 - K_t)^{-h}\left(\mu \frac{1 + \alpha(d_0 + d_v)}{1 + \beta(g_1 h + (equ_c\ \ v_0))} - 1\right)$$

$$\texttt{B-sum(n)} = \sum_{h=1}^{n} (\texttt{B-term}(h) + \texttt{B-term}(-h))$$

An example from the digital Phase-Locked Loop proof

Proof of B-term-neg and B-sum-neg using Smtlink:

```
(defthm B-term-neg
  (implies (a-bunch-of-hypothesis)
           (< (+ (B-term h v0 dv g1 Kt)
                 (B-term (- h) v0 dv g1 Kt)) 0))
  :hints (("Goal"
           :clause-processor
           (smtlink-custom-config clause
             (smt-std-hint "B-term-neg") )))
  :rule-classes :linear)

(defthm B-sum-neg
  (implies (a-bunch-of-hypothesis)
           (< (B-sum 1 n-minus-2 v0 dv g1 Kt) 0))
  :hints (("Goal" :in-theory (e/d (B-sum) (B-term)))))
```

Future work

- Support better counter-example report
  - Fetch counter-example result from the SMT solver and interpret it into ACL2 constants.
  - The clause processor can execute the counter-example to make sure they are indeed counter-examples.
- Add bounded model checking ability
  - We can use the SMT solver to build a bounded model checker that can be called through the customizable Smtlink interface.
- Typing with less typing
  - Type information can be extracted from `define`.
  - `type-alist` may contain lemmas/facts that Smtlink can send to the SMT solver to help with proofs.
- Explore other interesting applications

Summary

Smtlink handles tedious details of proofs so you can focus on the interesting parts.

- We have demonstrated Smtlink for AMS design verification. Other cyberphysical problems should benefit as well.
- Smtlink is designed to be extensible to support, for example: other domains, and using more of the SMT solver's capabilities.

Summary

Smtlink handles tedious details of proofs so you can focus on the interesting parts.

- It provides an architecture and examples for further research on combining the complementary strengths of ACL2 and SMT solvers.

Thank you!
Questions or thoughts?

Bibliography

J. Crossley, E. Naviasky, and E. Alon, *An energy-efficient ring-oscillator digital pll*, Custom Integrated Circuits Conference (CICC), 2010 IEEE, Sept 2010, pp. 1–4.

Primitive functions are:
`binary-+`, `unary--`, `binary-*`, `unary-/`, `equal`, `<`, `if`, `not`, and `lambda` along with the constants `t`, `nil`, and arbitrary integer constants.

An example from the digital Phase-Locked Loop proof

Definition of B-term (I've removed guards and returns to save space):

```
(define B-term-expt (Kt nco)
  (expt (gamma Kt) (- nco)))

(define B-term-rest (nco v0 dv g1)
  (1- (* (mu) (/ (1+ (* *alpha* (+ v0 dv)))
                 (1+ (* *beta* (+ (* g1 nco) (equ-c
     v0))))))))

(define B-term (nco v0 dv g1 Kt)
    (* (B-term-expt Kt nco) (B-term-rest nco v0 dv g1)))
```

An example from the digital Phase-Locked Loop proof

Definition of B-sum (I've removed guards and returns to save space):

```
(define B-sum (nco_lo nco_hi v0 dv g1 Kt)
  :measure (if (and (integerp nco_hi) (integerp nco_lo)
                    (>= nco_hi nco_lo))
               (1+ (- nco_hi nco_lo)) 0)
  (if (and (integerp nco_hi) (integerp nco_lo) (>= nco_hi nco_lo))
      (+ (B-term nco_hi v0 dv g1 Kt )
         (B-term (- nco_hi) v0 dv g1 Kt)
         (B-sum nco_lo (- nco_hi 1) v0 dv g1 Kt))
    0))
```

An example from the digital Phase-Locked Loop proof

std-smt-hint:

```
(define smt-std-hint (clause-name)
  :guard (stringp clause-name)
  ‘( (:expand ((:functions ( (B-term rationalp)
                             (B-term-expt rationalp)
                             (B-term-rest rationalp)
                             (dv0 rationalp)
                             ...
                             (fdco rationalp)
                             (gamma rationalp)
                             (m rationalp)
                             (mu rationalp)))
                (:expansion-level 1)))
     (:uninterpreted-functions ((expt rationalp rationalp
   rationalp)))
     (:python-file ,clause-name)))
```

An example from the digital Phase-Locked Loop proof

Proof of B-term-neg using Smtlink:

```
(defthm B-term-neg
  (implies (and (integerp h) (<= 1 h) (< h (/ (* 2 g1)))
                (hyp-macro g1 Kt v0 dv))
           (< (+ (B-term h v0 dv g1 Kt) (B-term (- h) v0 dv g1 Kt)) 0))
  :hints (
          ("Goal"
           :in-theory (enable B-term B-term-expt B-term-rest mu equ-c gamma dv0)
           :clause-processor
           (smtlink-custom-config clause (smt-std-hint "B-term-neg") )))
  :rule-classes :linear)
```

An example from the digital Phase-Locked Loop proof

Proof of B-sum-neg:

```
(defthm B-sum-neg
  (implies (and (integerp n-minus-2)
                (<= 1 n-minus-2)
                (< n-minus-2 (/ (* 2 g1)))
                (hyp-fn (list :v0 v0 :dv dv :g1 g1 :Kt Kt)))
           (< (B-sum 1 n-minus-2 v0 dv g1 Kt) 0))
  :hints (("Goal" :in-theory (e/d (B-sum) (B-term)))))
```