README

In het begin heb ik nagedacht over waar de 1-UIP-clause wordt aangemaakt en teruggegeven, want als we op deze plek die clause zeg maar kunnen onderscheppen in de code en deze zodanig kunnen aanpassen zodat we daarvan de geminimaliseerde versie kunnen maken, dan zijn we klaar. De plek in de code waar dit gebeurt is de ‘analyze-conflict’ procedure. In plaats van dat we de conflicterende clause dan teruggeven, doen we eerst nog een oproep naar een andere procedure die instaat voor het minimaliseren van de 1-UIP-clause, namelijk ‘minimize-1-uip-clause’.

Voordat ik deze implementeerde moest ik een aantal preprocesstappen voorzien die tijdens het sat-solving genoeg informatie opslaan en bijhouden zodanig dat de ‘minimize-1-uip-clause’ het verdere werk kan doen. Hiervoor hebben we het ‘interpretation-adt’ uitgebreid met de volgende toevoegingen.

* We houden in het record-type een extra vector bij (marks) die bijhoudt of een bepaalde literal gemarkeerd is of niet (literal indexed vector). Dit is ook de eerste stap van het gehele minimalisatieproces. Dit met de bedoeling zodat we later weten welke de literals zijn van de 1-UIP-clause. Elke locatie van de vector bevat dan een cons-cel met in de car of hij al dan niet gemarkeerd is en in de cdr een lijst met de antecedenten. Deze lijst bestaat dan nog is uit cons-cellen met de literal + zijn rede in. Op basis van de rede moeten we later nog een beslissing maken.
* Daarbij hebben we ook nog hulpprocedures voorzien (mark-literals!, mark-lit! en marked?) die respectievelijk alle literals van een gegeven clause gaan markeren, één enkele literal markeren en teruggeeft of een gegeven literal gemarkeerd is of niet.
* Verder zijn er nog twee hulpprocedures (set-antecedents! en get-lit-antecedents). set-antecedents! gaat voor een gegeven literal zijn antecedenten bijhouden en get-lit-antecedents gaat voor een gegeven literal zijn antecedenten teruggeven.

Natuurlijk moeten we nog zoeken naar de plekken in de code waar we de antecedente literals gaan opslaan. Hiervoor ben ik gaan kijken naar het begin van het proces en heb ik dit handmatig doorlopen om te weten op welke plekken we actie moeten ondernemen. Meer specifiek ben ik opzoek gegaan naar de plaatsen in de code waar een literal waar wordt gemaakt. Dit kan in drie gevallen:

* Een literal wordt waargemaakt omdat de clause maar uit één literal bestaat. Dit gebeurt aan het begin van het proces.
* Een literal kan waargemaakt worden omdat deze a.d.h.v. andere literals gepropageerd kan worden.
* Een literal wordt waar omdat we zelf de beslissing nemen.

Vanaf het moment dat we een literal waar moeten maken in één van de drie gevallen, houden we bij welke literal dat is en met welke rede hij waar wordt gemaakt (unit, prop, dec).

Dit doen we door deze info te consen aan een opbouwende lijst ‘antecedents’ en deze dan vervolgens te setten d.m.v. de hulpprocedure.

Als we dan uiteindelijk onze 1-UIP-clause hebben, dan minimaliseren we deze door het volgende te doen:

1. Eerst markeren we de literals van onze clause door de ‘mark-literals!’ op te roepen met onze interpretatie.
2. Daarna gebruiken we een remp om uit de lijst van literals van de 1-UIP-clause die literals te verwijderen die verwijderd mogen worden door recursief de opgebouwde antecedentenlijst van die literal af te gaan. Als we een gemarkeerde literal tegenkomen tijdens het afdalen, dan verwijderen we die literal. Indien hij waar is door een beslissing laten we deze in de lijst staan. LET OP, WE DOEN DIT ENKEL VOOR DE LITERALS DIE GEPROPAGEERD GEWEEST ZIJN, WANT DAT ZIJN DE LITERALS DIE ANTECEDENTEN HEBBEN.
3. Uiteindelijk geven we deze geminimaliseerde clause terug uit de procedure.