A | Aan de slag met IDP

0 | Installatie

Het IDP systeem kan zowel online als offline gebruikt worden. Voor de online versie surf je naar: $http://adams.cs.kuleuven.be/idp/server.html$

Voor Unix en Mac OS systemen zijn er kant-en-klare installatiepakketten voorzien. Deze hoeven enkel uitgepakt te worden. Voor windowssystemen raden we aan gebruik te maken van een docker.

Unix	$https://dtai.cs.kuleuven.be/krr/files/releases/idp/idp-linux-latest.tar.gz$
Mac OS	$https://dtai.cs.kuleuven.be/krr/files/releases/idp/3.7.0/idp-3.7.0-Darwin.tar.gz$
Windows	$https://dtai.cs.kuleuven.be/krr/files/releases/idp/README-DOCKER.md$

Het online systeem is voorzien van een eenvoudige IDE, je kan tevens een offline versie van deze IDE downloaden voor je lokaal systeem (vergeet het pad naar de IDP installatie niet aan te passen): $https://sourceforge.net/projects/idp/files/idp-ide/$

1 | Booleaanse formules

Je kent booleaanse formules natuurlijk al lang uit diverse programmeertalen. Het volgende stukje Java code:

```
public static void main(String[] a) {
    // Declaraties
    boolean A;
    boolean B;
    boolean C;

    // Toekenningen
    A = true;
    B = false;
    C = false;

    // Booleaanse formule
    System.out.println(A && (B || !C));
    }
```

berekent bijvoorbeeld of de booleaanse formule $A \wedge (B \vee \neg C)$ voldaan is voor de volgende toekenning van waarheidswaarden aan de booleaanse veranderlijken A, B en C:

A	waar
B	onwaar
C	onwaar

Welk resultaat zou dit programma produceren? ....

We kunnen deze berekening ook laten uitvoeren door het IDP systeem, mbv. de inferentietaak $sat(theory, structure)$, als volgt:

```
vocabulary V {
    A
    B
    C
}

structure S: V {
    A = true
    B = false
    C = false
}

theory T: V {
    A & (B | ~C).
}

procedure main() {
    print(sat(T, S))
}
```

Oef. 1 ▶ Maak een bestand `abc.idp` aan met bovenstaande code en voer bovenstaand programma uit. Welk resultaat produceert IDP? Komt dit overeen met wat je verwachtte van het Java-programma? ....

Oef. 2 ▶ Bereken zelf nog een mogelijke toekenning van waarheidswaarden waarvoor de formule voldaan is en controleer mbv. IDP dat dit effectief het geval is.

A	....
B	....
C	....

Oef. 3 ▶ Kijk nu ook eens na wat er gebeurt bij de volgende toekenning van waarheidswaarden:

A	waar
B	onwaar
C	waar

Oef. 4 ▶ Maak een programma `xyz.idp` met een booleaanse formule in termen van booleaanse variabelen X, Y en Z die waar is als en slechts als *precies twee* van de variabelen waar zijn.

Hier zijn nog even de verschillende logische operatoren:

ASCII notatie	Wiskundig symbool
&	$\wedge$
|	$\vee$
~	$\neg$

Gebruik IDP om na te gaan dat de toekenning $S1$ van waarheidswaarden wel de gewenste eigenschap heeft, maar die van $S2$ en $S3$ niet (maak hiervoor drie aparte structuren aan):

	S1	S2	S3
X	waar	onwaar	waar
Y	waar	waar	waar
Z	onwaar	onwaar	waar

2 | Implicaties

Er zijn een aantal nuttige afkortingen om eenvoudiger ingewikkelde booleaanse formules te kunnen opschrijven:

- De *implicatie* wordt gelezen als een *als-dan* constructie en is een afkorting voor "ofwel niet X, ofwel Y".
- De *equivalentie* wordt gelezen als een *als-en-slechts-als* constructie en is een afkorting voor een combinatie van implicaties.

Notatie	Symbool	Betekenis	Volledig
X => Y	$X \Rightarrow Y$	als X, dan Y	$\neg X \vee Y$
X <=> Y	$X \Leftrightarrow Y$	X als en slecht als Y	$(X \Rightarrow Y) \wedge (X \Leftarrow Y)$

Oef. 5 ▶ Beschouw nu de volgende toekenning van waarheidswaarden:

P	waar
Q	waar
R	onwaar

Vul in de volgende tabel de overeenkomstige waarheidswaarde van de gegevens booleaanse formules in. In geval van twijfel kan je je eigen mening controleren met behulp van IDP.

$P \Rightarrow Q$	....
$R \Rightarrow Q$	....
$P \Rightarrow R$	....
$P \Leftrightarrow Q$	....
$P \Leftrightarrow R$	....
$(P \wedge Q) \Leftrightarrow (P \vee R)$	....
$(P \vee Q) \Leftrightarrow (P \wedge R)$	....
$(P \Rightarrow Q) \Leftrightarrow (P \Leftarrow R)$	....

3 | Model expansie

In de vorige oefeningen heb je IDP steeds laten controleren of een toekenning van waarheidswaarden ervoor zorgt dat een booleaanse uitdrukking voldaan is. Je kan IDP ook zelf een dergelijke toekenning van waarheidswaarden laten zoeken mbv. de inferentietaak $modelexpand(theory, structure)$. Het resultaat is een tabel met oplossingen (modellen genaamd in IDP) die een uitbreiding zijn op jouw structuur en voldoen aan alle formules in je theorie.

```
procedure main() {
    printmodels(modelexpand(T, S))
}
```

Om het resultaat af te drukken gebruiken we een aangepaste $print$-functie, die alle modellen op een ordelijke manier af print of "unsatisfiable" weergeeft als er geen modellen gevonden werden.

Oef. 6 ▶ Maak een bestand `pqr.idp` aan waarin je IDP een oplossing laat zoeken voor de laatste formule uit oef. 5, als je enkel weet dat R onwaar is.

P	....
Q	....
R	onwaar

Is dit de enige mogelijke oplossing? Nee, de oplossing uit de vorige opgave (P en Q waar en R onwaar) was bijvoorbeeld ook een juiste oplossing.

Oef. 7 ▶ Je kan IDP ook laten proberen om meerdere (of alle) modellen te berekenen, ipv. slechts één, door de optie $nbmodels$ te wijzigen in het gewenste aantal (of "0" voor alle) modellen:

```
procedure main() {
    stdoptions.nbmodels = 3
    ...
}
```

Laat IDP alle mogelijke toekenningen van waarheidswaarden voor P, Q en R berekenen die voldoen aan de formule.

P	....	....	....	....
Q	....	....	....	....
R	....	....	....	....

Begrijp je waarom dit allemaal correcte oplossingen zijn voor de gegeven formule?

4 | Types en constanten

Men kan in IDP ook types declareren. IDP kent enkel *enumeratie*-types, die een opsomming zijn van een aantal mogelijke waarden. Een type wordt gedeclareerd in het vocabularium en (typisch) gedefinieerd in de structuur:

```
vocabulary V {
    type Dag
}

structure S: V {
    Dag = { Maandag; Dinsdag; Woensdag; Donderdag;
                          Vrijdag; Zaterdag; Zondag }
}
```

Men kan dan constanten declareren van een bepaald type. Dit zijn uitdrukkingen (genaamd *termen*), die je in een formule kan gebruiken om te verwijzen naar een object van dit type. In de bijhorende structuur kan je dan vastleggen naar welk object de constante juist verwijst. Essentieel is een constante dus gewoon een naam voor een bepaald object. Zo introduceert het volgende voorbeeld de term $Kerstmis$ als een naam voor het object `Maandag`.

Als je in je structuur niet vastlegt naar welk object een bepaalde constant verwijst, dan zal de modelexpansie-operatie van IDP hiervoor zelf een object van het juiste type kiezen, zodanig dat met deze keuze aan alle formules van de theorie voldaan is. In het onderstaande voorbeeld gebeurt dit met de constante $Nieuwjaar$.

```
vocabulary V {
    type Dag
    Kerstmis: Dag
    Nieuwjaar: Dag
}

structure S: V {
    Dag = { Maandag; Dinsdag; Woensdag; Donderdag;
                          Vrijdag; Zaterdag; Zondag }
    Kerstmis = Maandag
}
```

Oef. 8 ▶ Vertrek van het bestand `dagen.idp` en laat IDP alle mogelijke modellen berekenen. Er zijn zeven verschillende oplossingen mogelijk, $Nieuwjaar$ kan op eender welke dag van de week vallen aangezien we hier nog niets over gezegd hebben.

In je formules kan je termen (en dus in het bijzonder ook constanten) gebruiken als argumenten van relatiesymbolen. In de volgende sectie gaan we hier dieper op in, maar nu vermelden we alvast twee speciale relatiesymbolen, die (on)gelijkheid tussen twee termen voorstellen:

Notatie	Symbool
=	$=$
~=	$\neq$

Gebruik het gelijkheidssymbool om aan de theorie een regel toe te voegen die zegt dat $Nieuwjaar$ steeds op dezelfde dag als $Kerstmis$ valt. Hoeveel mogelijke oplossingen worden er nu nog gevonden? ....

5 | Relaties

Een relatie is een functie die een aantal argumenten van een bepaald type neemt en een booleaanse waarde produceert. Als de waarde `true` geproduceerd wordt, zeggen we dat de argumenten tot de relatie behoren, en als `false` geproduceerd wordt, dan behoren de argumenten er niet toe. We kunnen een relatie met n argumenten met andere woorden ook zien als een verzameling van tuples met n elementen.

Bijvoorbeeld, hier wordt een relatie $Weekend$ met 1 argument van het type `Dag` gedefinieerd, zodat $Weekend$ dus een verzameling van dagen is. In de bijhorende structuur leggen we vast dat dit de verzameling van de dagen `Zaterdag` en `Zondag` is:

```
vocabulary V {
    type Dag
    Weekend(Dag)
}

structure S: V {
    Dag = { Maandag; Dinsdag; Woensdag; Donderdag;
                         Vrijdag; Zaterdag; Zondag }
    Weekend = { Zaterdag; Zondag }
}
```

Oef. 9 ▶ Stel dat we ons niet meer kunnen herinneren op welke dag Pasen valt, maar er wel van overtuigd zijn dat het altijd in een weekend is. Voeg de relatie $Weekend$ en een constante $Pasen$ van het type Dag toe aan je programma en specifieer dat Pasen in het weekend moet vallen:

```
theory T {
    ...
    Weekend(Pasen).
}
```

Laat IDP al eens de verschillende mogelijkheden berekenen.

Oef. 10 ▶ We herinneren ons plots ook dat $Pasen$ nooit op een `Zaterdag` valt, waardoor we IDP kunnen laten uitrekenen op welke dag Pasen dan wel valt. Breid je formule omtrent $Pasen$ uit met je nieuwe kennis.

Wanneer je nu IDP laat zoeken naar mogelijke oplossingen, doet dit niet wat je verwachtte. Het probleem is dat `Zaterdag` gedefinieerd is als een domeinelement in onze structuur, maar een theorie hangt niet samen met een structuur (enkel met een vocabularium) en kent de inhoud van een specifieke structuur dan ook niet. Aangezien een theorie toegepast kan worden op verschillende structuren en deze structuren verschillende definities van types kunnen bevatten, zouden er problemen optreden wanneer we in een theorie spreken over een specifiek domeinelement dat in de ene structuur wel zit en in een andere niet.

IDP is voorzien van een speciale constructie voor types die “onveranderlijk” zijn, dwz. types die in alle structuren voor een bepaald vocabularium altijd dezelfde elementen bevatten (zoals in ons voorbeeld het type Dag), zodat deze wel gekend zijn in de theoriën die gebruik maken van het vocabularium:

```
vocabulary V {
    type Dag constructed from { Maandag, Dinsdag, Woensdag,
                        Donderdag, Vrijdag, Zaterdag, Zondag }
    ...
}
```

Deze declaratie doet twee dingen: ze zorgt ervoor dat elke structuur voor het vocabularium V het type dag zal interpreteren door de zeven objecten `Maandag` t/m `Zondag`; en ze introduceert in het vocabularium V ook zeven constanten `Maandag` t/m `Zondag`, waarbij de constante `Maandag` een naam is voor het object `Maandag`, enzovoort voor de andere zes constanten.

Oef. 11 ► Pas je type Dag aan en controleer dat IDP je nu wel kan zeggen wanneer Pasen valt.

Relaties kunnen ook meer dan één argument hebben. In volgend voorbeeld, kan de relatie $Geeft$ gebruikt worden om te specifiëren dat een bepaalde docent een bepaald vak geeft:

```
vocabulary V {
    type Docent constructed from { DVT, JVE, TGO }
    type Vak constructed from { ApplicatieArch, ArtificieleInt,
    EmbeddedDinges, ProductCert }
    Geeft(Docent, Vak)
}

structure S: V {
    Geeft = { JVE, ArtificieleInt;
              JVE, ApplicatieArch;
              TGO, EmbeddedDinges;
              DVT, ProductCert         }
}
...
```

Oef. 12 ► Vertrek van het bestand `vakken.idp`. Voeg een constante $BuisVak$ van het type Vak toe en specifieer in je theorie dat dit vak gegeven wordt door een bepaalde docent (kies zelf dewelke). Laat IDP alle mogelijke modellen zoeken.

6 | Kwantificatie

Met behulp van een constante kan je een eigenschap uitdrukken die geldt voor één specifiek object (namelijk dat object waarvoor de constante een naam is). Daarnaast kan je ook variabelen gebruiken om uit te drukken dat een bepaalde eigenschap moet gelden voor *minstens één* object x van een bepaald type (zonder dat je zelf vastlegt welk object dit dan wel is) of voor *elk* object x van een bepaald type. (**Let op:** dit soort van variabelen lijkt helemaal niet op een variabele uit een programmeertaal als C, maar heeft meer weg van variabelen uit wiskunde.) Een variabele introduceer je met behulp van ofwel de *existentiële kwantor* $\exists$ of de universele kwantor $\forall$. Een variabele hoeft niet x te heten, maar mag eender welke naam hebben (bij voorkeur beginnend met een kleine letter).

Notatie	Symbool	Betekenis
?x[T]:	$\exists x[T]:$	Er bestaat een object x van type T zodat
!x[T]:	$\forall x[T]:$	Voor elk object x van type T geldt dat

Bijvoorbeeld, zo drukken we uit dat elke docent minstens één vak geeft:

```
∀doc[Docent]: ∃vak[Vak]: Geeft(doc,vak).
```

Als IDP zelf het type van een variabele kan afleiden, hoef je dit zelf niet expliciet op te geven. Aangezien er in het vocabularium staat dat het eerste argument van de relatie $Geeft$ (en dus ook `doc`) van het type $Docent$ moet zijn en het tweede (en dus ook `vak`) van het type Vak, mag je deze eigenschap dus ook zo schrijven:

```
∀doc: ∃vak: Geeft(doc,vak).
```

Oef. 13 ▶ Voeg bovenstaande uitdrukking toe aan je theorie. Controleer dat je structuur voldoet aan je theorie. Voeg zelf een formule toe die zegt dat elk vak door minstens één docent gegeven wordt en controleer dat je structuur nog steeds voldoet aan je theorie.

Oef. 14 ▶ Pas je structuur zodanig aan dat het programma `unsatisfiable` wordt omwille van de eerste formule (elke docent geeft minstens één vak) in je theorie. (Controleer dit door na te gaan dat als je deze eerste formule uitcommentarieert, het geheel weer `satisfiable` wordt.)
Doe dit ook eens voor de tweede formule (elk vak wordt door minstens één docent gegeven).

Oef. 15 ▶ Voeg een relatie $Gemakkelijk$ toe, die bedoeld is om aan te geven welke vakken gemakkelijk zijn. Voeg aan je theorie een formule toe die zegt dat minstens één vak gegeven door `JVE` gemakkelijk is. Controleer dat de uitvoer van IDP inderdaad is wat je zou verwachten.

Oef. 16 ▶ (Zet vorige formule in commentaar en) Voeg een formule toe die zegt dat *elk* vak dat gegeven wordt door `JVE` gemakkelijk is.

Tip ▶ Als dit niet meteen lukt, kan je eerst eens proberen om volgende tabel in te vullen: Zie je het verband tussen de relaties $Geeft$ en $Gemakkelijk$?

	Geeft(JVE,vak)	Gemakkelijk(vak)
vak = AI	Waar	....
vak = AA	....	....
vak = ED	....	....
vak = PC	....	....

Oef. 17 ▶ Voeg nu een formule toe die zegt dat elk vak dat door `DVT` gegeven wordt niet gemakkelijk is.

Extra 1 ▶ Er zijn twee voor de hand liggende manieren om op te schrijven dat elk vak dat door `DVT` gegeven wordt niet gemakkelijk is: ofwel met een "$\forall x$" ofwel met de negatie van een"$\exists x$" (er bestaat geen vak dat door `DVT` gegeven wordt en dat gemakkelijk is). Probeer ook deze andere manier eens uit, zet hiervoor je eerste manier in commentaar.

Map Colouring Problem.
Het *Map Colouring Problem* is een probleem waarbij je een kaart moet inkleuren met een bepaalde hoeveelheid kleuren, zodanig dat alle aangrenzende landen een verschillende kleur hebben.

Oef. 18 ▶ Stel dat we bovenstaande kaart willen inkleuren met 4 kleuren. Vertrek van het bestand `map.idp` met volgende specificatie voor de figuur:

```
vocabulary V {
    type Kleur
    type Land
    Grens(Land, Land)
}

structure S: V {
    Land = { Belgie; Nederland; Duitsland; Luxemburg;
                    Frankrijk; Zwitserland; Oostenrijk }
    Kleur = { Rood; Oranje; Geel; Groen; Blauw; }
    Grens = { (Nederland,Belgie); (Nederland,Duitsland);
              (Belgie,Frankrijk); (Belgie,Luxemburg);
              (Belgie,Duitsland); (Luxemburg,Frankrijk);
              (Luxemburg, Duitsland); (Frankrijk,Duitsland);
              (Frankrijk,Zwitserland); (Duitsland,Zwitserland);
              (Duitsland,Oostenrijk); (Zwitserland,Oostenrijk) }
}
```

Voeg een relatie $ToegekendeKleur$ toe om een kleur toe te kennen aan elk land en schrijf een theorie om het Map Colouring Problem op te lossen.
Wanneer je gespecifieerd hebt dat twee aangrenzende landen een verschillende kleur moeten hebben/niet dezelfde kleur mogen hebben, doet je programma nog niet helemaal wat je wilt. Wat is er aan de hand?

Tip ▶ Als je IDP niet uitdrukkelijk vertelt dat elk land effectief een kleur moet krijgen, zal je merken dat hij sommige landen gewoon ongekleurd laat. Je zal dus een formule moeten toevoegen die uitdrukt dat er aan elk land een kleur moet worden gegeven.

Tip ▶ Je zal ook merken dat, als je dit niet uitdrukkelijk verboden hebt, IDP een land meerdere kleuren kan geven. Je kan opnieuw op twee manieren te werk gaan om te specifiëren dat elk land slechts één kleur toegekend kan worden:

- “Er bestaan geen twee kleuren zodat een land beide kleuren toegekend wordt”
- “Voor elke twee kleuren die toegekend worden aan een land, geldt dat deze één en dezelfde kleuren moeten zijn”

Extra 2 ▶ Probeer beide manieren eens uit om te specifiëren dat elk land slechts één kleur toegekend kan worden (zet hiervoor de manier die je reeds gebruikte in commentaar). Zie je het verband tussen existentiële en een universele kwantificatie?

$$\ldots\ \exists x:\ \ldots\ R(x) \quad\Longleftrightarrow\quad \ldots\ \forall x:\ \ldots\ R(x)$$

7 | Functies

In de vorige oefening was er een functioneel verband tussen de landen en hun kleur: elk land werd een kleur en exact één kleur toegekend. We hebben IDP dit moeten vertellen door expliciet enkele beperkingen extra op te nemen in onze theorie. We kunnen dit nog op een andere manier doen, namelijk door gebruik te maken van een functie (ipv. een relatie).
Let op: de term "functie" wordt hier gebruikt in zijn wiskundige betekenis en doelt dus op een afbeelding van tuples van elementen van één of meer verzameling(en) op elementen van een andere verzameling.

Bijvoorbeeld, er is een functioneel verband tussen personen en geboortejaren, aangezien iedereen exact één geboortejaar heeft (meerdere personen kunnen natuurlijk wel hetzelfde geboortejaar hebben). Ook kan de grootte van een persoon op diens verjaardag worden uitgedrukt als functie van de persoon en de verjaardag. Elke persoon heeft immers één bepaalde lengte op zijn of haar verjaardag, maar verschillende personen kunnen wel dezelfde lengte hebben:

```
vocabulary V {
    type Persoon
    type Jaartal
    type Grootte
    Geboortejaar(Persoon): Jaartal
    Grootte(Persoon, Jaartal): Grootte
}

structure S: V {
    Persoon = { Jos; Jef; Marie }
    Jaartal = { 2010; 2011 }
    Grootte = { 0..100; }
    Geboortejaar = { Jos -> 2010; Jef -> 2011; Marie -> 2010 }
    Grootte = {
        Jos,   2010 -> 50; Jos,   2011 -> 70;
        Jef,   2010 -> 0;  Jef,   2011 -> 55;
        Marie, 2010 -> 45; Marie, 2011 -> 65;
    }
}
```

Let op: Wanneer je een functie zelf invult in je structuur, moet deze steeds compleet gedefinieerd zijn, maw. er moet ook effectief voor elk mogelijk tuple in het domein van de functie gespecifieerd worden op welke waarde dit wordt afgebeeld.

Oef. 19 ▶ Hernoem je vorige oplossing `map.idp` naar `mapfunc.idp` en verander je relatie $ToegekendeKleur$ in een functie die een land mapt op een kleur. Je hebt nu geen constraints meer nodig die zeggen dat elk land een kleur moet hebben en dat een land slecht één kleur kan hebben.

8 | Integer types

Een integer type in IDP is niets anders dan een enumeratietype waarop bepaalde aritmetische bewerkingen ($+, -, *, /, \%$) gedefiniëerd zijn. Het ingebouwde type ‘`int`” kan best niet rechtstreeks gebruikt worden, aangezien de grenzen hiervan onbepaald zijn. De declaratie “`type MySubType isa MyType`” kan gebruikt worden om subtypes te definiëren. Bijvoorbeeld, het volgende programma (`octaal.idp`) berekent welke twee cijfers het getal `25` vormen in het octaal talstelsel, gebruikmakend van een subtype `Octaal` van het type `int`. Probeer dit voorbeeld eens uit.

```
vocabulary V {
    type Octaal isa int
    Cijfer1: Octaal
    Cijfer2: Octaal
}

structure S: V {
    Octaal = { 0; 1; 2; 3; 4; 5; 6; 7 }
}

theory T: V {
    Cijfer1 * 8 + Cijfer2 = 25.
}

procedure main() {
    printmodels(modelexpand(T, S))
}
```

De definitie van het type $Octaal$ mag trouwens ook afgekort worden als:

```
Octaal = { 0..7 }
```

Of dit mag bijvoorbeeld ook:

```
Octaal = { 0..4; 5; 6..7 }
```

Oef. 20 ► Vertrek van het bestand `diet.idp` en schrijf een programma dat de volgende letterpuzzel oplost:

```
  LESS
  FOOD
+ ----
  DIET
```

De regels bij zo’n puzzel zijn de volgende:

- Elke letter stelt een cijfer van 0 t/m 9 voor.
- Elke letter staat voor een ander cijfer (reeds voorzien in je theorie).
- Een getal mag nooit beginnen met het cijfer nul.

Laat IDP eens uitrekenen of er een oplossing van de volgende vorm bestaat en vul aan:

L	E	S	F	O	D	I	T
…	6	2	1	…	…	…	9

Oef. 21 ► Maak een bestand `cde.idp` aan en los volgende letterpuzzel op, waarbij klinkers even cijfers voorstellen en medeklinkers oneven cijfers:

```
    AI
    BA
+  ---
   CDE
```

Maak hiervoor gebruik van één relatie $Even$ met waarden `0, 2, 4, 6 en 8`.

A	B	C	D	E	I
…	…	…	…	6	…

Extra 3 ► Je merkte aan de implementatie van de regel “elke letter staat voor een ander cijfer” dat het gebruik van constanten voor de verschillende letters niet ideaal is, zeker wanneer je met een groter aantal letters werkt. Aangezien elke letter op exact één waarde mapt, kan je hier perfect een functie voor gebruiken.
Hernoem je oplossing `diet.idp` voor de eerste letterpuzzel naar `dietfunc.idp`. Vervang je constanten door één functie $Waarde$ die letters mapt op cijfers. Vergeet hierbij niet om de constanten ook uit je structuur te verwijderen. Specifieer dmv. kwantificatie dat twee letters niet dezelfde waarde mogen hebben. Om te controleren of je tot dezelfde oplossing komt dan vooheen, kan je, voor nu, de waarden van de letters als beperkingen opnemen in je theorie.

Extra 4 ► Zoals gezien in 7.Functies, kan een functie niet slechts gedeeltelijk gedefinieerd worden in een structuur. Moeten de gegeven waarden dan als harde beperkingen opgenomen worden in je theorie? Dit is geen goede oplossing. De gegeven waarden van deze puzzel maken namelijk geen deel uit van de logica die nodig is om de puzzel op te lossen. Wanneer we een andere toekenning van waarden aan letters willen doen, zou dit dan in de theorie aangepast moeten worden. Idealiter worden de gegevens in een nieuwe structuur geplaatst, zodat de theorie onaangepast gebruikt kan worden voor verschillende toekenningen. Kan je, met wat je tot nu toe geleerd hebt, een manier bedenken om de gegeven toekenning op te nemen in je IDP-programma zodat de theorie niet steeds moet aangepast worden? Doe de nodige toevoegingen in `dietfunc.idp`.

Extra 5 ► In de tweede letterpuzzel lijkt het overbodig om de relatie $Even$ te definiëren in je structuur, aangezien deze wiskundig gedefinieerd kan worden en onveranderlijk is. Zet je toekenning van waarden $0, 2, \ldots$ in commentaar in `cde.idp` en voeg een formule toe aan je theorie die de relatie $Even$ definieert mbv. een universele kwantor.

9 | Aggregaten

Eén van de features van IDP die we nog niet behandeld hebben, zijn aggregaten: functies die werken op een verzameling van objecten. De algemene vorm van zo'n aggregaat ziet er als volgt uit:

$$agg\{\ x_1\ x_2\ \ldots\ x_n:\ \phi:\ t\}$$

Met *agg* de aggregaatfunctie, x_i de variabelen (zowel van ϕ als t), ϕ een formule en t een term. De betekenis van zo'n aggregaat is dat we de functie *agg* (bv. `sum` of `product`) toepassen op de verzameling van alle instantiaties van de term t waarvoor aan ϕ voldaan is.

Het aggregaat "kardinaliteit" (genoteerd met #) telt hoeveel verschillende instantiaties er zijn waarvoor de formule voldaan is. In dit geval hebben we de term t niet nodig. Bijvoorbeeld, in de oefening met de verschillende dagen en het weekend (`dagen.idp`), kunnen we gaan tellen hoeveel dagen er in het weekend vallen, als volgt:

```
#{d[Dag]: Weekend(d)}
```

Deze uitdrukking stelt het *aantal* elementen `d` voor, waarvoor de formule $Weekend(d)$ waar is, maw. het aantal dagen die in het weekend vallen. We kunnen dit nu naar wens gaan gebruiken in formules, bijvoorbeeld:

```
1 < #{ d[Dag]: Weekend(d) } ∧ #{ d[Dag]: Weekend(d) } < 3.
```

Dit mag trouwens ook:

```
1 < #{ d[Dag]: Weekend(d) } < 3.
```

Veronderstel dat we ons programma met vakken en docenten uitbreiden met het concept studiepunten:

```
vocabulary V {
    type Docent constructed from { DVT, JVE, TGO }
    type Vak constructed from { ApplicatieArch, ArtificieleInt,
                               EmbeddedDinges, ProductCert, Graphics }
    Geeft(Docent, Vak)

    type StudiePunten isa int
    Punten(Vak): StudiePunten
}

structure S: V {
    StudiePunten = { 0..10 }
    Punten = { ApplicatieArch -> 4; ArtificieleInt -> 6;
           EmbeddedDinges -> 5; Graphics -> 5; ProductCert -> 6; }
}
```

Oef. 22 ▶ Vertrek van het bestand `docenten.idp` en laat IDP een relatie `Geeft(Docent,Vak)` berekenen, zodanig dat:

- `JVE` zeker niet de vakken `ProductCert` of `EmbeddedDinges` geeft;
- `TGO` het vak `Graphics` geeft;
- Elk vak exact één docent heeft;
- Elke docent behalve `DVT` minstens 2 vakken geeft;

(Voor de laatste twee items kan je gebruik maken van het kardinaliteitsaggregaat.)

Hier is nog even een overzicht van de verschillende vergelijkingsoperatoren:

ASCII notatie	Wiskundig symbool
=	$=$
~=	$\neq$
<	$<$
=<	$\leq$
>	$>$
>=	$\geq$

Als we een verzameling van variabelen x hebben van een `int` type, dan kunnen we met die x-en meer doen dan enkel de kardinaliteit berekenen. IDP kent vijf verschillende aggregaatfuncties:

Notatie	Betekenis
#	Kardinaliteit
sum	Som
$prod$	Product
min	Minimumwaarde
max	Maximumwaarde

Oef. 23 ▶ Voeg aan je theorie een formule toe die zegt dat het totale aantal studiepunten van de vakken die door dezelfde docent gegeven worden maximaal 10 mag zijn.

10 | Optimalisatie

IDP kan ook gebruikt worden om optimalisatie-problemen op te lossen: in plaats van eender welk model van je theorie te berekenen, zal IDP dan een model berekenen dat de waarde van een bepaalde numerieke term minimaliseert. Dit gebeurt door middel van de inferentietaak $minimize(theory, structure, term)$.
Hiervoor definieer je eerst de term die je wil minimaliseren en vervolgens laat je IDP de modellen zoeken waarin deze term zijn minimale waarde aanneemt (voor maximalisatie neem je de negatie van de gewenste term).

Dit ziet er dus als volgt uit:

```
vocabulary V {
    ...
}

structure S: V {
    ...
}

theory T: V {
    ...
}

term t: V {
    ...
}

procedure main() {
    printmodels(minimize(T, S, t))
}
```

Oef. 24 ► In een vorige opgave hebben we het *Map colouring problem* bekeken. Er bestaat hiervan ook een optimalisatie variant, waarbij het de bedoeling is om het aantal gebruikte kleuren te minimaliseren. Hernoem je oplossing `mapfunc.idp` naar `mapopt.idp` en voeg een term toe die het aantal effectief gebruikte kleuren telt. Laat IDP nu deze term minimaliseren.

Oef. 25 ▶ Gebruik de structuur S_euro uit `europa.idp`, die hoort bij volgende kaart:

Tip ▶ Je kan een structuur ook opnemen in je IDP-programma dmv. een $include$. Je hoeft de gegeven structuur dus niet te kopiëren naar je eigen IDP-bestand. Om een externe structuur toe te voegen aan je programma, moet het vocabularium, waar de structuur bij hoort, reeds gedefinieerd zijn:

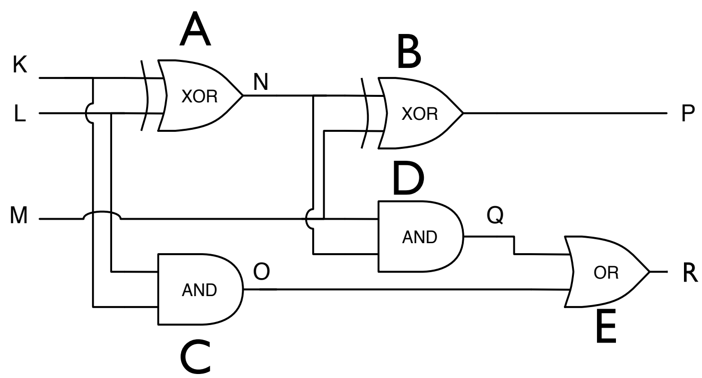
```
vocabulary V {
    ...
}

include "bestandsnaam.idp"
```

Hoeveel verschillende kleuren heb je minstens nodig om deze kaart te kleuren? ....
(Je kan je uitkomst eventueel controleren op bovenstaande kaart.)

B | Analyse van logische circuits

We gaan IDP gebruiken voor het analyseren van logische circuits.



1 | Voorstelling van circuits in IDP

Veronderstel dat we bovenstaand logisch circuit beschrijven met volgende structuur:

```
structure Circuit: CircVoc {
    Gate = { A; B; C; D; E }
    Wire = { K; L; M; N; O; P; Q; R }
    Output = { A ↦ N; B ↦ P; C ↦ O; D ↦ Q; E ↦ R }
    FirstInput = { A ↦ K; B ↦ N; C ↦ L; D ↦ M; E ↦ Q }
    SecondInput = { A ↦ L; B ↦ M; C ↦ K; D ↦ N; E ↦ O }
    Xor = { A; B }
    And = { C; D }
    Or = { E }
}
```

Oef. 1 ▶ Vertrek van het bestand `circuit.idp` en schrijf de bijhorende definitie van het vocabularium $CircVoc$.

Oef. 2 ▶ Om de toestand van het circuit te beschrijven, gebruiken we een relatie $On(Wire)$, die aangeeft welke lijnen aan staan (de andere staan dan uit). Voeg deze toe aan het vocabularium. Reken eens uit welke interpretatie er bij deze relatie hoort in het geval dat, van de invoerdraden K, L en M, enkel K en M aan staan.

K	L	M	N	O	P	Q	R
x		x	...	...	...	...	...

Voeg de interpretatie die je net berekend hebt toe aan de structuur.

Oef. 3 ▶ Voeg dan een theorie-blok toe, waarin je beschrijft hoe de twee invoerlijnen van, respectievelijk, een And-, Xor-, of Or-poort zich verhouden tot de bijhorende uitvoerlijn. Om je op weg te helpen, is hier alvast de beschrijving van een And-poort:

$$\forall \texttt{x: And(x)} \Rightarrow \texttt{(On(FirstInput(x))} \land \texttt{On(SecondInput(x))} \Leftrightarrow \texttt{On(Output(x)))}.$$

Controleer of je theorie inderdaad voldaan is in je structuur. Probeer dit ook eens uit met wat andere waarden voor de relatie On.

2 | Simulatie

We kunnen IDP gebruiken om de werking van het circuit te simuleren.

Oef. 4 ▶ Gegeven een bepaalde waarde van de invoerlijnen van het circuit (K, L en M, in dit geval), berekenen we de waarde van de andere lijnen. Om dit voor elkaar te krijgen, hebben we in de eerste plaats een relatie $InputWire$ nodig, die aangeeft welke de invoerlijnen precies zijn. Voeg deze relatie toe aan het vocabularium en de structuur.

Oef. 5 ▶ We zouden nu ook de waarde van de invoerlijnen moeten toevoegen aan de structuur. Maar we willen dit natuurlijk doen zonder dat we ook de waarde van de andere draden moeten opgeven. Met ons huidige vocabularium zal dit niet lukken, aangezien we hier enkel maar de relatie On hebben die de waarde van alle lijnen beschrijft. Los dit probleem op door het introduceren van een nieuwe relatie $InputWireOn$, die de toestand van enkel maar de invoerlijnen van het circuit weergeeft. Wat moet er nu gebeuren met de relatie On die we al hadden?

Oef. 6 ▶ Vanzelfsprekend moet er nog een verband zijn tussen $InputWireOn$ en On. Voeg aan je theorie een formule toe die het vereiste verband uitdrukt. Gebruik deze theorie dan om te berekenen wat de waarde van de andere lijnen zal zijn als enkel de invoerlijnen K en M aan staan (maw. L staat uit). Als alles goed is, zou dit natuurlijk hetzelfde resultaat moeten opleveren als in de vorige oefening. Probeer nu eens te berekenen wat het resultaat zal zijn als enkel L aan staat en ga na dat je effectief het juiste resultaat krijgt.

$InputWireOn$	On
K, M	....
L	....

3 | Omgekeerde simulatie

In de vorige oefening hebben we de werking van het circuit gesimuleerd door te kijken welke uitvoer geproduceerd wordt voor een gegeven invoer. IDP laat ons echter ook toe om, gebruikmakend van hetzelfde model van het circuit, een simulatie in omgekeerde richting uit te voeren.

Oef. 7 ▶ Op basis van een gehoopte uitvoer van het circuit, kunnen we nagaan welke verschillende invoerwaarden allemaal aanleiding zouden geven tot deze uitvoer. Net zoals we in de vorige oefening nood hadden aan een relatie $InputWireOn$ om de toestand van een specifiek deel van de lijnen te kunnen beschrijven, hebben we hiervoor een gelijkaardige relatie $OutputWireOn$ nodig. Voeg relaties $OutputWire$ en $OutputWireOn$ toe aan het vocabularium, en voeg de gepaste interpretatie voor de relatie $OutputWire$ toe aan de structuur.

Oef. 8 ▶ Voeg aan de theorie een formule toe die de relatie tussen On en $OutputWireOn$ definieert. Ga nu na welke verschillende waardes voor de invoerlijnen allemaal aanleiding geven tot de uitvoertoestand waarin P en R allebei aan staan. Probeer dit ook eens uit voor een uitvoertoestand waarin enkel P aan staat.

$OutputWireOn$	$InputWireOn$
P, R	....
P	....

4 | Validatie

In deze oefening gaan we eens kijken naar het probleem van *validatie*: om te controleren of het circuit wel correct werkt, voeren we een aantal metingen uit.

Oef. 9 ▶ Om te beginnen, beschouwen we een heel eenvoudig geval, waarbij we een bepaalde invoer aan het circuit geven en dan de toestand van een aantal lijnen meten, met de bedoeling om na te gaan of deze toestand overeenkomt met onze verwachtingen. Voeg aan het vocabularium een relatie $ObservedOn$ (de lijnen waarvan er geobserveerd is dat ze aan staan) en een relatie $ObservedOff$ (de lijnen waarvan er geobserveerd is dat ze uit staan) toe. (Draden die noch in $ObservedOn$, noch in $ObservedOff$ voorkomen, zijn dus precies die draden die we niet hebben opgemeten.) Druk in je theorie uit dat de geobserveerde waardes overeenkomen met de toestand van de geobserveerde lijnen.

Oef. 10 ▶ Gebruik IDP om na te gaan of het circuit correct werkt als we volgende observaties maken:

(1)	Invoer: K en L staan aan; M staat uit Observaties: we observeren dat N uit staat en dat O aan staat	....
(2)	Invoer: K en M staan aan; L staat uit Observaties: we observeren dat Q uit staat en dat N aan staat	....

Sla je programma (`circuit.idp`) op en hernoem naar `circuitdiag.idp` alvorens verder te gaan.

5 | Diagnose

Als uitbreiding op de vorige oefening rond validatie, beschouwen we nu het probleem van *diagnose*: in plaats van enkel te bepalen of het hele circuit al dan niet goed werkt, willen we nu ook weten welke componenten er mogelijk defect zijn.

Oef. 11 ▶ Voeg een relatie $Broken$ toe aan het vocabularium. In je vorige theorie heb je gedefinieerd wat de waarde van de uitvoerlijn van een poort zou moeten zijn in functie van zijn invoerlijnen. In deze oefening gaan we dit veranderen, en in plaats daarvan zeggen dat:

> *"Voor elke And-poort* **die niet defect is***, moet het zo zijn dat zijn uitvoerlijn aan staat als en slechts als zijn twee invoerlijnen aan staan."*

Pas je formule op deze manier aan en doe hetzelfde met de formules voor Or- en Xor-poorten.

Oef. 12 ▶ Neem nu opnieuw de tweede reeks observaties uit de vorige opgave (K en M staan aan, geobserveerd dat Q uit staat en N aan) en kijk wat voor uitvoer IDP produceert.

Laat IDP ook eens zoeken naar alle mogelijke oplossingen.

Je ziet dat IDP heel veel verschillende diagnoses vindt. Dit is ook logisch, aangezien een defecte component eender welke waarde voor zijn uitvoerlijn kan produceren, en we dus nooit zeker kunnen zijn dat een component niet defect is. Hoewel het resultaat van IDP dus in principe correct is, is het niet zo nuttig.

Oef. 13 ▶ Wat we liever zouden hebben, is dat IDP enkel maar *minimale* diagnoses zoekt, dwz. dat we enkel maar gaan veronderstellen dat een component defect is, als we effectief observaties hebben die betekenen dat hij niet correct kán werken. Introduceer hiervoor een *term* $NbBroken$, die het aantal defecte componenten telt, en laat deze door IDP minimaliseren. Bestudeer het effect van deze uitdrukking op de uitvoer en overtuig jezelf ervan dat dit resultaat nu inderdaad veel nuttiger is dan het vorige.

Minimale diagnoses
....

Oef. 14 ▶ Kijk ook eens naar de eerste reeks observaties uit de vorige opgave (K en L staan aan, geobserveerd dat N uit staat en O aan).

Vergelijk de uitvoer die je hiervoor krijgt als je IDP vraagt om alle model expansies te berekenen met de uitvoer als je de term $NbBroken$ laat minimaliseren. Begrijp je waarom dit zo is?

Minimale diagnoses
....

Sla je programma op en hernoem naar `circuittest.idp` alvorens verder te gaan.

6 | Testen

In de vorige oefening hebben we telkens geprobeerd om een diagnose te stellen aan de hand van één reeks van observaties voor dezelfde invoer. We kunnen natuurlijk ook trachten om meerdere testen uit te voeren en al deze informatie samen te gebruiken om tot een diagnose te komen.

Oef. 15 ▶ Veronderstel dat we volgende 3 testen hebben uitgevoerd op het circuit:

(1) Invoer: K aan, L en M uit
Observaties: P en R staan aan

(2) Invoer: L aan, K en M uit
Observaties: P en R staan aan

(3) Invoer: M aan, K en L uit
Observaties: P staat aan en R staat uit

Om deze drie verschillende tests te kunnen voorstellen, moet je een type $Test$ toevoegen, met als elementen 1, 2 en 3. Een aantal relaties zullen dan ook een bijkomend argument van type $Test$ krijgen. Voer deze veranderingen door.

Oef. 16 ▶ Zoek de minimale diagnoses van het circuit, gegeven de drie bovenstaande tests. Hierbij wordt er natuurlijk verondersteld dat de status van een bepaalde poort (dwz. defect of niet defect) dezelfde blijft gedurende al de testen.

Minimale diagnoses
....

C | (Inductieve) Definities

1 | Definities

Een speciale en veelvoorkomende vorm van kennis zijn *definities*. Een voorbeeld van een definitie:

> *"Een student slaagt voor een vak als en slechts als hij/zij een score van minstens 10/20 haalt op dit vak."*

Deze definitie definieert het concept "geslaagd zijn" in termen van het concept "examenscore".

Oef. 1 ▶ In `geslaagd.idp` vind je een vocabularium en een structuur met wat informatie over examenscores van enkele vakken:

```
vocabulary V {
    type Vak
    type Score isa nat
    ExamenResultaat(Vak): Score
    Geslaagd(Vak)
}

structure S: V {
    Vak = { AI; EmbeddedDinges; Graphics; }
    Score = { 0 .. 20 }
    ExamenResultaat = { AI ↦ 12;
                        EmbeddedDinges ↦ 8;
                        Graphics ↦ 17;    }
}
```

Maak een theorie die de definitie van *Geslaagd* uitdrukt. Bereken de model expansies van deze theorie tov. de bovenstaande structuur. Als je theorie correct is, zou er natuurlijk maar één model mogen zijn.

In bovenstaande oefening heb je de IDP operatoren die je al kende gebruikt. Daarnaast beschikt IDP echter ook over een speciale notatie die specifiek dient om definities mee op te schrijven. Deze specifieke notatie biedt een drietal voordelen over het gebruik van de algemene operatoren, die we hieronder één voor één zullen bespreken.

Een definitie in IDP bestaat uit een verzameling regels, omsloten door accolades. Elke regel heeft de volgende vorm:

$$\forall x_1\ x_2\ \ldots\ x_n :\ A\ \leftarrow\ F.$$

waarbij A een atomaire formule is en F een formule. Elk van deze regels beschrijft één specifiek geval waarin het gedefinieerde concept waar is.

De definitie van *Geslaagd* kan bijvoorbeeld geschreven worden als:

```
{
```
$$\forall x : \; Geslaagd(x) \;\leftarrow\; ExamenResultaat(x) \geqslant 10.$$
```
}
```

Je leest deze definitie als volgt:

> *Het concept* Geslaagd *is als volgt gedefinieerd:*
>
> - *Als het examenresultaat van een vak minstens 10 is, ben je geslaagd voor dit vak.*
> - *In geen enkele andere situatie, ben je geslaagd voor een vak.*

Merk op dat—zoals blijkt uit het laatste puntje—een definitie dus altijd een *volledige* opsomming geeft van alle gevallen waarin het gedefinieerde concept waar is. In alle gevallen die niet opgesomd zijn, is het concept maw. dus niet waar.

Oef. 2 ▶ Maak een nieuwe theorie ($T2$) aan die de definitie van $Geslaagd$ bevat.

Een eerste voordeel van deze notatie is dat ze duidelijk maakt *welk* concept juist gedefinieerd wordt in termen van welke andere concepten: door het feit dat de *Geslaagd* links van de pijl staat, is het duidelijk dat dit de relatie is die gedefinieerd wordt.

Een tweede voordeel volgt uit het feit dat een definitie uit meerdere regels kan bestaan. Hierdoor kan de interne structuur van een definitie beter bewaard worden. Beschouw de volgende definitie:

> *Een student slaagt voor een vak in de volgende gevallen:*
>
> - *Als hij hier een score van minstens 10/20 op haalt*
> - *Als hij een score van minstens 8/20 haalt en het vak tolereert.*

Oef. 3 ▶ Voeg een relatie $Getolereerd$ toe aan het vocabularium en de structuur en pas de definitie van $Geslaagd$ aan. Doe dit zowel voor de "definitie" die je zelf gemaakt had in Oef. 1, als voor de definitie met de speciale definitie-notatie in Oef. 2. Welk van beide versies lijkt je het gemakkelijkst om te onderhouden en welke sluit het beste aan bij de definitie zoals we ze hierboven in het Nederlands gaven? ....

Extra 6 ▶ Veronderstel dat je een definitie hebt die er zo uitziet:

```
{
```
$$\forall x_1 \; x_2 \; \ldots \; x_n : \; A \;\leftarrow\; Formule_1.$$
$$\ldots$$
$$\forall x_1 \; x_2 \; \ldots \; x_n : \; A \;\leftarrow\; Formule_n.$$
```
}
```

Zou je deze definitie in het algemeen kunnen omzetten naar een IDP formule die geen gebruik maakt van de speciale definitie-notatie?

$\forall x_1 \; x_2 \; \ldots \; x_n :$ ....

2 | Definities door Inductie

Hierboven hebben we relaties gedefinieerd. Het is eveneens mogelijk om functies te definiëren. We illustreren dit door de volgende rij te definiëren.

$$3\ 5\ 7\ 9\ 11\ 13\ 15\ ...$$

IDP kent geen lijsten of arrays, maar we kunnen een dergelijke rij voorstellen als een functie die een index afbeeldt op een waarde:

```
vocabulary V {
    type Index isa int
    type Waarde isa int
    Rij(Index): Waarde
}

structure S: V {
    Index = { 0 .. 10 }
    Waarde = { 0 .. 20 }
}

theory T: V{
    ∀i[Index]: Rij(i) = 3 + 2*i.
}
```

We kunnen dezelfde rij ook nog op een andere manier definiëren, namelijk door onderstaande twee regels:

- $Rij(0) = 3$;
- Voor alle $i > 0$, geldt dat $Rij(i) = Rij(i - 1) + 2$.

Dit soort van definitie—waarbij we het begin van de rij gebruiken om het vervolg van de rij te definiëren—noemen we een *inductieve definitie*. Dergelijke definities bestaan typisch uit één of meerdere basisgevallen, zoals het eerste item hierboven, en een aantal inductieve gevallen, zoals het tweede item.

Oef. 4 ▶ Vervang in `rij.idp` de definitie van Rij door een inductieve definitie.

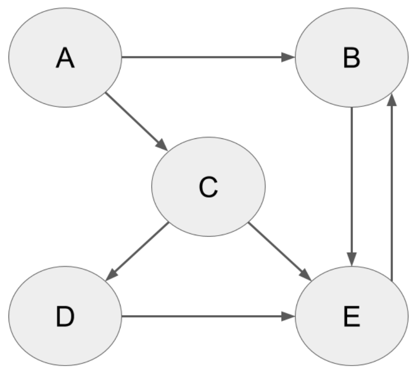
Bij de eenvoudige rij hierboven, konden we kiezen tussen een inductieve of een niet-inductieve definitie. Er zijn echter ook rijen die alleen maar op een inductieve manier gedefinieerd kunnen worden. Een voorbeeld hiervan is de welbekende Fibonnacci rij:

$$0\ 1\ 1\ 2\ 3\ 5\ 8\ 13\ 21\ 34\ 55...$$

Oef. 5 ▶ Schrijf een inductieve definitie van een functie Fib die de rij van Fibannacci getallen voorstelt. Maak hiervoor een nieuwe theorie T_fib aan. Test dat de eerste 11 elementen van deze rij inderdaad overeenkomen met wat hierboven staat.

Een ander klassiek voorbeeld van een inductieve definitie is die van de bereikbaarheid (*reachability*) in een grafe.

> *Een knoop x is bereikbaar vanuit knoop y in de volgende gevallen:*
>
> - *Als er een boog is van y naar x;*
> - *Als er een derde knoop z bestaat, zodat z bereikbaar is vanuit y en x bereikbaar is vanuit z.*



Bovenstaande grafe kunnen we voorstellen door volgend vocabularium en structuur:

```
vocabulary V {
    type Knoop
    Boog(Knoop, Knoop)
}

structure S: V {
    Knoop = { A; B; C; D; E }
    Boog = { A,B ; A,C ; B,E ; C,D ; C,E ; D,E ; E,B }
}
```

Oef. 6 ► Vertrek van `grafe.idp` en maak een theorie met daarin een definitie van de bereikbaarheidsrelatie en test uit dat ze bij dit voorbeeld het juiste resultaat geeft.

Extra 7 ► In een vorige extra oefening heb je erover nagedacht hoe je definities ook kan uitdrukken *zonder* gebruik te maken van de speciale definitie-notatie van IDP. Probeer dit eens te doen met je definitie van de bereikbaarheidsrelatie. Je zal zien dat dit niet correct werkt (je krijgt teveel modellen). Begrijp je hoe dit komt?

Weetje: Men kan wiskundig bewijzen dat het onmogelijk is om de bereikbaarheidsrelatie te definiëren in logica zonder gebruik te maken van de speciale definitie-notatie.

3 | N-koninginnen probleem

Het N-koninginnen-probleem kan worden omschreven als volgt. Plaats N koninginnen op een $N \times N$ schaakbord, zodanig dat geen enkele koningin een andere kan slaan. Een koningin kan een ander schaakstuk slaan als dit schaakstuk op één lijn bevindt (horizontaal, verticaal of diagonaal) met het andere schaakstuk.

```
vocabulary V {
    type Rij isa nat
    type Kolom isa nat

    Koningin(Rij, Kolom)
    Slaan(Rij, Kolom, Rij, Kolom)
}

structure S: V {
    Rij = {1..8}
    Kolom = {1..8}
}

theory T: V{
    #{r k: Koningin(r,k)} = 8.
    ∀r1 r2 k1 k2: (r1≠r2∨k1≠k2) ∧
        Koningin(r1,k1) ∧ Koningin(r2,k2) ⇒ ¬Slaan(r1, k1, r2, k2).

}
```

Oef. 7 ► Vertrek van het bestand `koningin.idp` en schrijf een definitie die aangeeft wanneer twee koninginnen elkaar kunnen slaan.

D | Oplossen van puzzels

We gaan IDP gebruiken om een Sudoku op te lossen.

	2		3					
7	1				9	3		
	3	4	5				7	
9			6					
	4						6	
					4			9
	8				6	7	4	
		3	4				5	8
					1		9	

1 | Vocabularium

Oef. 1 ▶ Maak een vocabularium aan waarin je types Rij, $Kolom$ en $Getal$ definieert.

Oef. 2 ▶ Voeg aan dit vocabularium de relatie $Gegeven$ toe, die weergeeft welke getallen er al ingevuld zijn in het rooster en een functie $Oplossing$, die aangeeft welk getal op elke plaats van het rooster moet staan.

Merk op: $Gegeven$ is een relatie en $Oplossing$ een functie. De reden hiervoor is dat $Oplossing$ een waarde invult op elke plaats in het rooster, terwijl $Gegeven$ maar voor enkele vakjes in het rooster geldt.

Oef. 3 ▶ Voeg een relatie $Vierkant$ toe die je gaat gebruiken om te definiëren dat twee vakjes in hetzelfde 3×3 vierkant zitten.

2 | Structuur

Oef. 4 ▶ Voeg de mogelijke waarden toe die Rij, $Kolom$ en $Getal$ kunnen aannemen.

De $Gegeven$ getallen (rij, kolom, waarde) hebben we voor jou al op de juiste plek in het rooster ingevuld.

3 | Theorie

Ga bij elke stap na of wat je hebt toegevoegd ook de gewenste effecten heeft.

Oef. 5 ▶ Voeg een regel toe die zegt dat als een bepaald getal $Gegeven$ is, het ook in de $Oplossing$ moet zitten.

Oef. 6 ▶ Definieer wanneer 2 vakjes in hetzelfde $Vierkant$ zitten. De relatie vierkant geldt niet voor een vakje met zichzelf. Waarom is dit belangrijk?

....

....

....

Oef. 7 ▶ Voeg enkele regels toe zodat in elke rij, kolom, en vierkant maximaal 1 keer hetzelfde getal voorkomt. Het is niet nodig om te specifiëren dat elke rij, elke kolom en elk vierkant alle getallen moeten bevatten. Kan je uitleggen waarom dit overbodig is?

....

....

....

4 | Zelf aan de slag

Maak groepjes van twee en kies per groepje een van volgende puzzels waarvoor jullie samen een IDP programma zullen schrijven. Deze puzzels zijn er in vier categorieën van moeilijkheid, gaande van één ster (makkelijk) tot vier sterren (moeilijk). Bij het quoteren van jullie oplossing wordt de moeilijkheidsgraad van de oefening mee in rekening gebracht: het perfect oplossen van een 4-sterren oefening levert 20/20 op, een drie-sterren oefening 18/20, een 2-sterren oefening 16/20 en een 1-ster oefening 14/20.

Overleg met de docent van de practica over welke puzzel je kiest, zodat de puzzels zo goed mogelijk over de verschillende groepjes verdeeld kunnen worden.

(Meer uitleg en de complete spelregels kan je vinden in de app “Logic Games” beschikbaar in de appstore van Android, iOS en Windows of natuurlijk elders online)

⋆ Binairo (Landscaper).

In elke cel moet een nul of een één ingevuld worden. Er mogen niet meer dan twee dezelfde cijfers direct naast of direct onder elkaar staan. Elke rij en elke kolom is uniek en bevat evenveel nullen als enen.

	1		0
		0	
	0		
1	1		0

0	1	1	0
1	0	0	1
0	0	1	1
1	1	0	0

⋆ Mosaic.

Elk vakje kan gekleurd zijn of niet. Elk gegeven getal geeft aan hoeveel gekleurde vakjes er zijn in het 3x3 vierkant rond het vakje waar het getal in staat. Hierbij wordt het vakje zelf ook meegeteld, zodat het maximale getal dat in een vakje kan staan 9 is.

	2			3
0	3	5		3
2	5	6	7	4
4	7		7	4
4	6			4

	2			3
0	3	5		3
2	5	6	7	4
4	7		7	4
4	6			4

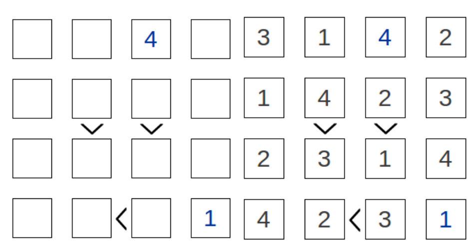
⋆ Tenner grid.

Elke rij moet de cijfers 0 t.e.m. de breedte van een rij bevatten, zodanig dat de som van elke kolom overeenkomt met de waarde aangegeven beneden de kolom. Aangrenzende vakjes (horizontaal, verticaal of diagonaal) mogen niet dezelfde waarde krijgen.

8	9	6	3	7	0	4	5	1	2
3	1	0	2	6	8	9	7	4	5
6	7	4	5	9	2	1	3	8	0
17	17	10	10	22	10	14	15	13	7

⋆ Futoshiki.

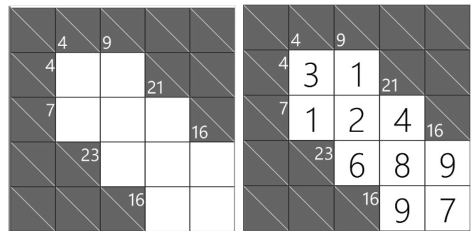
Plaats in elke rij en kolom de cijfers 1 t.e.m. de breedte van het rooster. Zorg er voor dat voldaan wordt aan de groter- en kleiner-dan-relaties die tussen de cellen staan.



★★

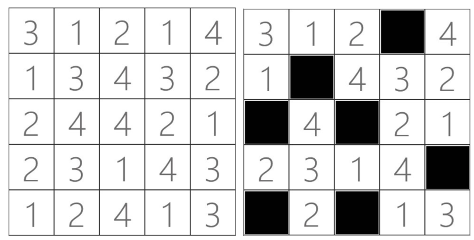
⋆⋆ **Kakuro.**

In deze puzzel is het de bedoeling om cijfers van 1 t.e.m. 9 in te vullen in een rooster. De opgavevakjes geven aan wat de som moet zijn van de cijfers die er horizontaal naast / verticaal onder staan. In dezelfde som mag elk cijfer maximaal één keer voorkomen.



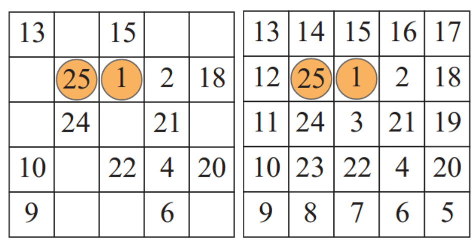
⋆⋆ **Hitori.**

Gegeven een rooster vol cijfers, kleur de vakjes zodat elk cijfer slecht één keer voorkomt in elke rij en kolom. Twee ingekleurde vakjes mogen niet aan elkaar grenzen (horizontaal of verticaal) en alle overblijvende vakjes moeten één doorlopend vlak vormen (er mag dus geen deee van de puzzel omsloten worden door gekleurde vakjes).



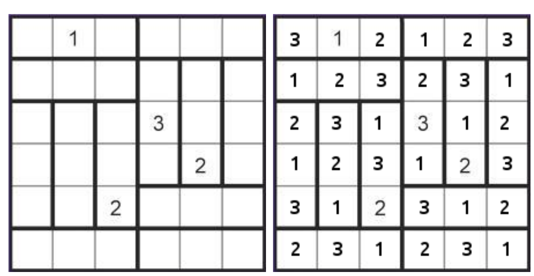
⋆⋆ **Hidato.**

In elke cel moet een nummer ingevuld worden tussen 1 en het aantal vakjes in het rooster. Opeenvolgende cijfers moeten telkens aan elkaar grenzen (horizontaal, verticaal of diagonaal).



⋆⋆ **Tatami.**

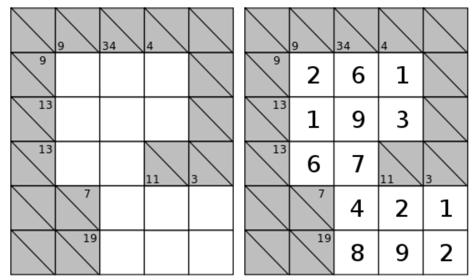
Elke rechthoek stelt een mat voor die een verschillende grootte kan hebben. Vul elke mat met de cijfers 1 tem. de grootte van de mat (elk cijfer kan dus slechts één maal voorkomen op een mat). Hierbij moet elk cijfer even vaak voorkomen per rij en kolom. Twee dezelfde cijfers mogen elkaar niet horizontaal of verticaal raken.



★★★

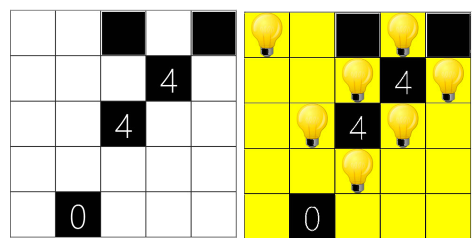
⋆⋆⋆ **Kakuro.**

In deze puzzel is het de bedoeling om cijfers van 1 t.e.m. 9 in te vullen in een rooster. De opgavevakjes geven aan wat de som moet zijn van de cijfers die er horizontaal naast / verticaal onder staan. In dezelfde som mag elk cijfer maximaal één keer voorkomen. Met als uitbreiding dat op dezelfde rij / kolom meerdere sommen kunnen staan, gescheiden door zwarte vakjes.



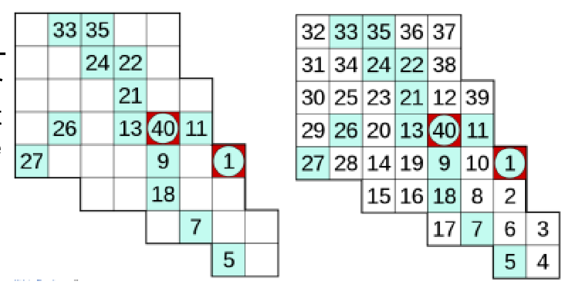
⋆⋆⋆ **Lighten up.**

Plaats een aantal lampjes in het rooster, zodat elke cel "verlicht"wordt. Een cel wordt verlicht wanneer er een lamp op een rechte lijn (horizontaal of verticaal) staat, zonder obstakels. Lampen mogen elkaar niet beschijnen en op de obstakels staat telkens vermeld hoeveel lampen eraan grenzen (horizontaal of verticaal).



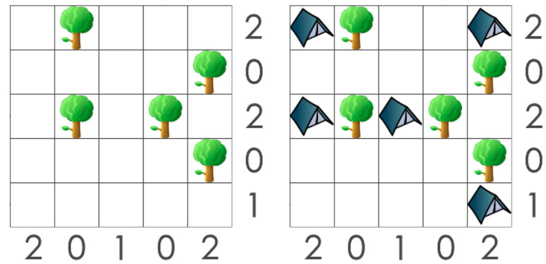
⋆⋆⋆ **Hidato.**

In elke cel moet een nummer ingevuld worden tussen 1 en het aantal vakjes in het rooster. Opeenvolgende cijfers moeten telkens aan elkaar grenzen (horizontaal, verticaal of diagonaal). Met als uitbreiding dat het rooster een willekeurige vorm kan aannemen.



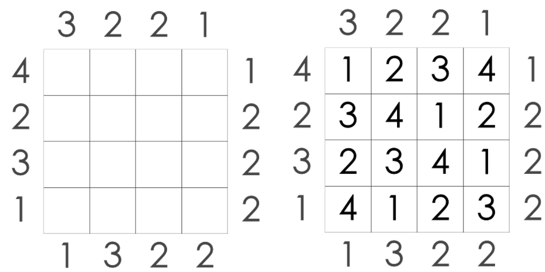
⋆ ⋆ ⋆ **Tents.**

In deze puzzel moeten er tenten worden ingevuld in het rooster. Elke tent is verbonden met exact één boom (horizontaal of verticaal). Er zijn dus evenveel tenten als bomen. Een tent mag wel naast meerdere bomen staan, maar is maar verbonden met één van deze. Twee tenten mogen nooit aan elkaar grenzen. De getallen boven en naast het rooster geven aan hoeveel tenten er in die specifieke rij of kolom staan.



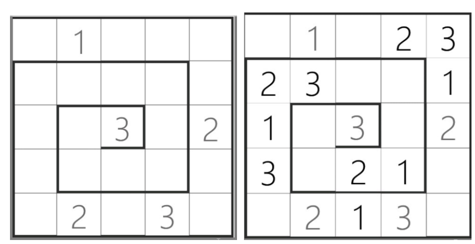
⋆ ⋆ ⋆ **Skyscrapers.**

In elke rij en kolom moeten de cijfers 1 t.e.m. het aantal rijen/kolommen worden ingevuld. Het cijfer geeft de hoogte van de wolkenkrabber weer. Aan de randen van het rooster staat vermeld hoeveel wolkenkrabbers gezien kunnen worden vanuit dit standpunt. Het kan zijn dat er al enkele wolkenkrabbers ingevuld staan.



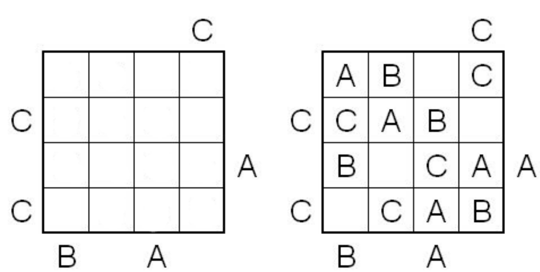
⋆ ⋆ ⋆ **Snail.**

Een spiraal vertrekt in de linkerbovenhoek en draait met de klok mee naar het middelste vakje van het rooster. Deze spiraal moet worden ingevuld door telkens de sequentie van 1, 2, 3. Lege vakjes zijn toegestaan. Het eerste cijfer in de spiraal moet altijd een 1 zijn, het laatste cijfer een 3. In elke rij en kolom moet elk cijfer exact één keer worden ingevuld.



⋆ ⋆ ⋆ **ABC.**

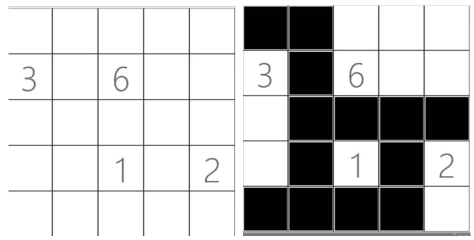
Plaats de letters A, B, en C exact één keer in elke rij en kolom. De letters links/rechts en boven/onder het rooster geven aan welke letter er het dichtst bij deze rand van de rij of kolom staat.



★★★★

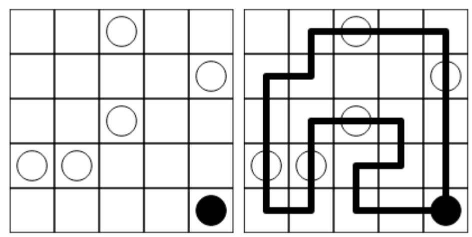
⋆⋆⋆⋆ **Nurikabe.**

Bouw één continue muur die het veld opdeelt in verschillende vlakken. De grootte van elk vlak is aangegeven met een cijfer. Verschillende vlakken mogen elkaar niet raken. De muur mag geen 2x2 vierkanten maken.



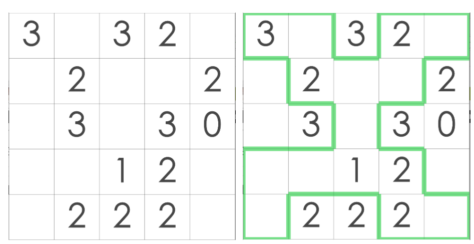
⋆⋆⋆⋆ **Masyu.**

Maak een gesloten ketting zonder aftakkingen die door het centrum van de cellen gaat. Er zijn 2 soorten parels, door de witte parels moet de ketting rechtdoor gaan, door de zwarte parels moet de ketting een 90 graden draai maken. Bovendien moet na de hoek in de zwarte parel, de draad langs beide kanten rechtdoor gaan; terwijl de draad na rechtdoor te gaan in een witte parel, langs minstens één kant een hoek moet maken.



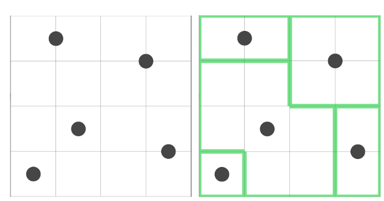
⋆⋆⋆⋆ **Slither Link.**

Maak een gesloten ketting zonder aftakkingen die tussen de vakjes van het rooster doorloopt. Elk getal in het rooster geeft weer aan hoeveel van de 4 randen van de cel de ketting grenst.



⋆⋆⋆⋆ **Galaxies.**

Het rooster bevat een aantal centra van melkwegen. Deze centra kunnen zich bevinden in het centrum van een cel, op de rand tussen 2 cellen of in de kruising van 4 cellen. Deel het rooster zo in dat elke melkweg symmetrisch is en alle melkwegen samen het hele rooster innemen.



IDP Cheat Sheet

Blokken:

```
vocabulary VocNaam {
  type Type
  type Getal isa int
  Relatie(Type,Type)
  Functie(Type,Type): Type
  Constante: Type
}
```

```
structure StrucNaam : VocNaam {
  Type = { A; B; C }
  Getal = { 1..3 }
  Relatie = { A,B ; B,C }
  Functie = { A,B -> C; B,C -> A }
  Constante = A
}
```

```
theory ThNaam: VocNaam {
  Formule.
  {
    Atoom <- Formule.     % definitie
  }
}
```

```
query QueryNaam: Vocnaam {
  { x y : P(x,y) }
}
```

```
term TermNaam: Vocnaam {
  X + Y
}
```

Logische operatoren:

Operator	IDP notatie	Wiskundige notatie
niet	~	$\neg$
en	&	$\wedge$
of	|	$\vee$
als, dan	=>	$\Rightarrow$
als en slechts als	<=>	$\Leftrightarrow$
voor alle	!	$\forall$
er bestaat	?	$\exists$

Vergelijkingsoperatoren:

Operator	IDP notatie	Wiskundige notatie
gelijk aan	=	=
verschillend van	~=	$\neq$
groter dan	>	$>$
kleiner dan	<	$<$
groter of gelijk aan	>=	$\geq$
kleiner of gelijk aan	=<	$\leq$

Varianten van de kwantoren:

$\forall$x y: P(x,y) $\Rightarrow$ Q(x,y).	$\exists$x y: P(x,y) $\wedge$ Q(x,y).
$\forall$(x y) in P: Q(x,y).	$\exists$(x y) in P: Q(x,y).
$\forall$(x y) sat P(x,y): Q(x,y).	$\exists$(x y) sat P(x,y): Q(x,y).

Betekenis afgeleide operatoren

$F \vee G$	$\neg(\neg F \wedge \neg G)$
$F \Rightarrow G$	$\neg F \vee G$
$F \Leftrightarrow G$	$\neg F \wedge \neg G \vee F \wedge G$
$\exists$x: P(x)	$\neg(\forall$x: $\neg$P(x))

Inferentietaken:

Model checking

```
procedure main() {
    print(sat(ThNaam,StrucNaam))
}
```

Query answering

```
procedure main() {
    print(query(QueryNaam,StrucNaam))
}
```

Model expansion

```
procedure main() {
    stdoptions.nbmodels = 0
    printmodels(modelexpand(ThNaam,StrucNaam))
}
```

Optimisation

```
procedure main() {
    printmodels(minimize(ThNaam, StructNaam, TermNaam))
}
```